Doctoral Thesis Dissertation

Invoice Factoring through Blockchain Technology

by:
Nasibeh Mohammadzadeh

Supervisor:
Dr. Jose Luis Muñoz Tapia
Co-supervisor:
Dr. Sadegh Dorri Nogoorani

Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Telematics

ISG (Information Security Group)
Department of Network Engineering

July 2, 2022

Declaration of Authorship

I, Nasibeh Mohammadzadeh , declare that this thesis titled, 'Invoice Factoring through Blockchain Technology' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"Do not fear to be eccentric in opinion, for every opinion now accepted was once eccentric."

Bertrand Russell

Abstract

Invoice factoring has been a popular way to provide cash flow for businesses. The primary function of a factoring system is to prevent an invoice from being factored twice. In order to prevent double factoring, many factoring ecosystems use one or several centralized entities to register factoring agreements. However, this puts a lot of power in the hands of these centralized entities and makes it difficult for users to dispute situations in which factoring data is unavailable, wrongly recorded or manipulated by negligence or on purpose.

This thesis presents our research around the current problems of invoice factoring and our new solutions to solve this process using the blockchain technology. A public blockchain can keep a permanent, secure, ordered and transparent record of transactions which are then available for everyone at any time to view and verify.

In this thesis, we start proposing a base solution, and we gradually enhance it. In the base protocol, we propose an architecture for invoicing registration based on a general blockchain. The blockchain platform builds trust between the parties by executing transactions correctly. We employed a smart contract to complete the registration process, and prevent double factoring. The smart contract provides for auditing and dispute resolution in such a way that privacy is protected and relevant information is always available.

In the second protocol, we add a relayer to our architecture for easier onboarding. Only the relayer is required to submit blockchain transactions, and pay the corresponding fees. Other participants can proxy their transactions through the relayer, and pay the relayer in fiat money. We also enhance our identity management and authentication using the concept of verifiable credentials (VC) in order to better comply with the Know-Your-Customer (KYC) regulation. In fact, in this architecture, participants use their decentralized identifiers (DIDs) and the DIDComm protocol for asynchronous and secure off-chain interactions.

In the final protocol, we greatly enhance our smart contract with respect to the conditions it checks before registering an invoice factoring. We integrate non-interactive zero-knowledge proofs and cryptographic commitments into our solution. With these cryptographic tools in place, we can prevent a special

type of denial of service (DoS) attack and better verify invoice details without compromising privacy.

Our protocols are very efficient in terms of blockchain costs. In particular, we only need one transaction to register an invoice factoring, and most of the details are recorded in low-cost blockchain storage. Our evaluations and comparison with the literature reveals that our protocols are superior to the related works with respect to efficiency, security, privacy, and ease of use.

Acknowledgements

First and foremost, I am incredibly grateful to my supervisor, Dr. Jose Luis Muñoz Tapia, for his invaluable advice, continuous support, and patience during my PhD study. His immense knowledge and great experience have encouraged me in my academic research and daily life. I would also like to extend my special gratitude to my Co-supervisor, Dr. Sadegh Dorri Nogoorani, for the endless guidance, counsel, and mentorship. He continuously encouraged and was always willing and enthusiastic to assist in any way he could throughout my PhD program.

I wish to thank my dad and mom, who taught me to be strong and always work towards my dreams. I am forever indebted to my parents for their encouragement and support when it is most needed. They patiently supported me all this time, both emotionally and financially, and believed in my success. Without their tremendous understanding and encouragement over the past few years, it would have been impossible for me to complete my PhD study. This Thesis reflects their endless motivation, encouragement, and prayers. I am also very grateful to my brother, Hadi, and His wife, Neda, who always cheered for me and celebrated my tiniest achievements.

I am deeply grateful to my Iranian friends for their support in the realization of this Thesis. Many thanks to my colleagues and friends at UPC, with whom we exchanged ideas daily. Their kind help and support have made my study and life in Barcelona a wonderful time.

Last but not least, I would like to thank the administrative staff, Doctoral Office, International Students Office (OMI) Office, and the Doctoral Committee of the Technical University of Catalonia for their endless support and guidance throughout the time I spent in Barcelona.

Contents

Declaration of Authorship ii

Acknowledgements vii

Contents viii

List of Figures xiii

List of Tables xv

Acronyms xvii

Notation xix

1 Introduction 1
1.1 Motivation . . . 1
1.1.1 The Factoring Scenario . . . 1
1.1.2 Challenges in Invoice Factoring . . . 2
1.1.3 Using Blockchain to Prevent Double Factoring . . . 3
1.2 Objectives . . . 4
1.3 Summary of contributions . . . 5
1.4 Methodology . . . 7
1.5 Resulting publications . . . 8
1.6 Thesis Outline . . . 9

2 Background 11
2.1 Public Distributed Ledgers . . . 11
2.2 Cryptographic Primitives . . . 13
2.3 Decentralized Identifiers . . . 14
2.4 DIDComm . . . 16
2.5 Zero-Knowledge Proofs . . . 17

3 Invoice Factoring Registration Based on a Public Blockchain 19

3.1 Introduction . . . 19
3.2 Proposed Architecture . . . 20
3.2.1 Design Goals & Assumptions . . . 21
3.2.2 Setup . . . 23
3.2.2.1 Key Management . . . 23
3.2.2.2 Blockchain Certificates . . . 24
3.2.2.3 The Smart Contract . . . 26
3.2.3 Phase 1: Registration . . . 29
3.2.4 Phase 2: Factoring . . . 30
3.2.5 Phase 3: Payment . . . 35
3.3 Security Analysis . . . 36
3.3.1 Blockchain and Invoice Certificates . . . 36
3.3.2 Communications' Security . . . 36
3.3.3 Data Manipulation and Repudiation Attacks . . . 36
3.3.4 Confidentiality and Privacy . . . 37
3.3.5 Dispute Handling . . . 38
3.4 Comparison with the Related Works . . . 39
3.5 Conclusions . . . 42

4 Decentralized Factoring for Self-Sovereign Identities 45
4.1 Introduction . . . 45
4.2 Proposed Architecture . . . 46
4.2.1 Design Goals & Assumptions . . . 47
4.2.2 The Protocol . . . 48
4.2.2.1 Phase 1: Credential Registration . . . 49
4.2.2.2 Phase 2: Factoring . . . 51
4.2.2.3 Phase 3: Payment . . . 57
4.2.3 Use Case . . . 58
4.3 Evaluation . . . 60
4.3.1 Security Analysis . . . 60
4.3.1.1 Verifiable Credentials . . . 60
4.3.1.2 Data Security and Privacy . . . 61
4.3.1.3 Availability . . . 62
4.3.1.4 Dispute Handling . . . 62
4.3.2 Comparison to Related Work . . . 63
4.4 Implementation . . . 66
4.4.1 Tools and Setup . . . 66
4.4.2 The Smart Contract . . . 67
4.4.3 Functions for Testing . . . 69
4.5 Conclusions . . . 70

5 Enhanced Privacy with Zero-Knowledge Proofs 73
5.1 Introduction . . . 73
5.2 Proposed Architecture . . . 74

5.2.1 Design Goals & Assumptions 75
5.2.2 The Protocol 76
5.2.3 The Protocol Setup 76
5.2.4 The Registration Phase 77
5.2.4.1 The Invoice Certificate 77
5.2.4.2 The Compliance Certificate 79
5.2.5 The Factoring Phase 80
5.2.6 Phase 3: Payment 83
5.2.7 Denial of Service Attack 83
5.3 Related Work 84
5.4 Conclusions 87

6 Conclusions and Future Work 89

A AppendixA 91

Bibliography 95

List of Figures

1.1 Typical factoring. 2

3.1 Our architecture. 21
3.2 Summary of smart contract storage possibilities. 27
3.3 Our protocol. 28
3.4 Timestamps and periods for factoring registration and payment. 31

4.1 Transaction relayer. 47
4.2 Registration of a factor's VC. 49
4.3 Registration of an invoice for a seller (seller-invoice VC). 51
4.4 The factoring phase. 54
4.5 Payment phase. 58

List of Tables

3.1 Comparison with close related work. 39

4.1 Comparison with the related works. 63

5.1 Comparison with close related work. 86

Acronyms

API	Application Program Interface
CA	Central Authority
CLR	Certificate Revocation List
DApp	Decentralized Application
DID	Decentralized Identifier
DH	Diffie-Hellmann
DecReg	Decentralized Registry
DoS	Denial of Service
DDos	Distributed Denial of Service
DIMS	Digital Identity Management System
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
EVM	Ethereum Virtual Machine
ECC	Elliptic-curve cryptography
GSN	Gas Station Network
IDs	Identifiers
IBAN	International Bank Account Identifier
KYC	Know Your Customer
KDF	Key Derivation Function
NIZK	Non-Interactive Zero-Knowledge
PoC	Proof of Concept
PKI	Public-Key Infrastructure
PII	Personally Identifiable Information
RSA	Rivest, Shamir, and Adelman
SSI	Self-Sovereign Identity
SHA-256	Secure Hash Algorithm 256-bit
VC	Verifiable Credential
ZKP	Zero-Knowledge Proof

Notation

$\mathrm{KDF}(m)$	Secure symmetric key derived from m (deterministic)
$h(m)$	Cryptographic hash of m
$\mathrm{h}(s,m)$	A salted hash function using salt s and string m.
ID_X	Real identity of entity X
$@X$	Blockchain address of entity X
DID_X	The public DID of entity X
$\mathrm{Enc}(K,m)$	Symmetric encryption of m using key K
$\mathrm{MAC}(K,m)$	Message authentication code of m using key K (e.g. HMAC)
PU_X	The public key of X
$\mathrm{Enc}(PU_X,m)$	Asymmetric encryption of m using the public key PU_X
$\mathrm{Enc}(K_{XY},m)$	Symmetric encryption of m using a key shared between X and Y
σ_m^X	Digital signature of X over message m using a long-term key pair
$\sigma_m^{@X}$	Digital signature of X over message m using a blockchain key pair
$h(m)$	Hash of m
I	Invoice number
S	seller
B	buyer
F	factor
R	relayer
A	Accreditation authority
C	smart contract
WS	Web Service (by the buyer)

Dedicated To My Lovely Parents.

The reason for what I become today.
For their endless love, support, and encouragement.

Chapter 1

Introduction

1.1 Motivation

In business-to-business financial relationships, it is a common practice to pay for some services or products with some delay, for example, several months later. In this situation, the provider (namely the seller) might sell her future receivable finance (invoice from a buyer) with a discount to a factoring entity (namely the factor, e.g., a bank). Invoice factoring has been a popular way to provide cash flow for businesses. This financial service is continually growing; for instance, only in Europe, invoice factoring has increased from less than a billion in 2010 to 1,6 billion Euros in 2017 [1].

This thesis is based on the research of the problems of the invoice factoring processes and in providing new solutions using the blockchain technology. This chapter first presents a high-level introduction to the problems and the main ideas to solve them. Then, follows by introducing the thesis objectives, a summary of the contributions and the applied methodology. We also present a list of scientific publications that resulted from the realization of the thesis and end with an outline of the thesis structure.

1.1.1 The Factoring Scenario

As we mentioned, a factoring relationship involves three parties [2]: (i) a buyer, who is a person or a commercial enterprise to whom the services are supplied on

credit, (ii) a seller, who is a commercial enterprise which supplies the services on credit and avails the factoring arrangements, and (iii) a factor, which is a financial institution (e.g., a bank) that benefits from the discount on invoice factoring. Typical interactions between these parties are the following (see Fig. 1.1):

1. The seller sells some service or product to the buyer.
2. In return, the buyer issues an invoice to the seller with an already agreed payment due in the future (typically, several months later).
3. The seller wants to get the money earlier and sells the invoice to the factor.
4. The factor pays the invoice's cost minus the fee to the seller.
5. When the due date of the invoice is reached, the factor asks the buyer to settle the invoice.
6. The buyer pays the amount to the factor.

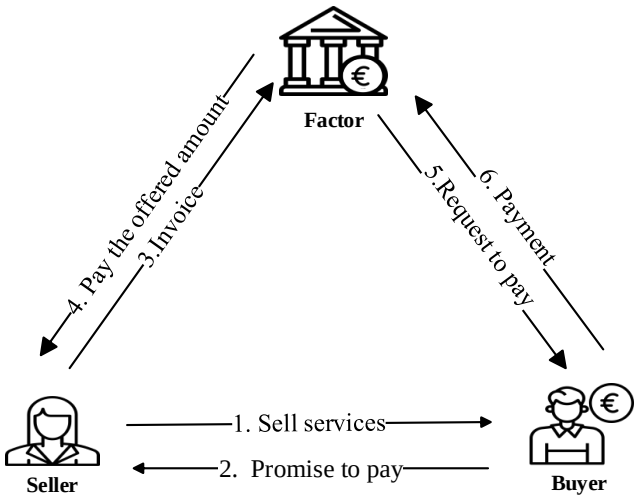

Figure 1.1: Typical factoring.

1.1.2 Challenges in Invoice Factoring

There are several issues and challenges in the traditional invoice factoring process. For example, it often requires several manual steps. Moreover, the information is dispersed among different systems and databases [3, 4]. There are

also trust issues related to factoring. The factor has to trust the buyer to pay the amount of invoice by the due deadline, and the buyer has to comply with the factoring contract between the seller and the factor. The seller may inquire about multiple financial institutions, and they may pay part of the invoice. However, the seller shall not be able to use the same invoice multiple times and receive extra money. So, the factor should verify and ensure that an invoice has not been financed yet. This issue is known as double factoring and it is the main problem that a factoring system needs to prevent [5]. In particular, double factoring is possible because there are no insights between factors about whether an invoice has already been financed or not . In general, the implication of the buyer is necessary to provide awareness between factors in whether an invoice has already been financed. Usually, we can assume that the buyer is a trusted party, since this entity does not have any economic incentives in the factoring process. This is clearly true when the buyer is an administration or a government, which is our main use case in this work.

1.1.3 Using Blockchain to Prevent Double Factoring

To prevent double factoring, many ecosystems (e.g., countries) use one or several centralized entities to register factoring agreements. However, this puts a lot of power in the hands of these centralized entities and makes it difficult for users to dispute situations in which factoring data is unavailable, wrongly recorded or manipulated by negligence or on purpose. Besides, if there are several possible centralized registries for invoice factoring, which is quite common, another problem arises. In this case, the factoring information is scattered and it is the responsibility of the buyer, the less interested entity in the factoring process, to check the records of all possible trusted third parties and make sure that the payment is made to the correct party.

In this context, a public blockchain seems a natural tool to solve these issues, because it can keep the record of factoring agreements but also prevent double factoring [6]. A distributed ledger can make the record-keeping database distributed and highly available, as well as logically unique and secure from manipulations [7]. This way, factoring agreements can be made faster with fewer errors and still carry the authenticity and credibility of manual contracts.

Several works have been proposed in the literature to implement new generation invoice factoring protocols using distributed ledger technologies [1, 5, 8]. However, none of them is completely able to fulfill the requirements that such a new generation ledger-based protocol should cope with. Concretely, ledger-based invoice factoring solutions should operate without a single point-of-failure, provide privacy and protection of personally identifiable information (PII) and business information, provide non-repudiation for handling disputes, be decentralized and secure against corruption, comply with Know-Your-Customer (KYC), be cost-effective, and provide an easy user on-boarding.

1.2 Objectives

The main objective of the thesis is to investigate the applicability of Blockchain technology to the invoice factoring process. For this purpose, three protocol models have been designed. The first protocol considers an architecture with a public blockchain and that all the interactions to complete a factoring registry are managed by a smart contract. All parties can trust the correct execution of transactions managed by the smart contract because the blockchain platform guarantees this execution. In the second protocol, the architecture is framed in a financial context; hence, strict regulatory restrictions apply to it. In particular, following the Know-Your-Customer (KYC) regulation, the involved parties need to be well identified to each other, and their agreements have to be persisted for later audits and law enforcement. In the third protocol, we use public-key cryptography and digital certificates for better key management. Some part of the data is symmetrically encrypted before being stored on-chain, while another part of the data is exchanged off-chain between the parties and we use zero-knowledge proofs and cryptographic commitments for proving existence.

In order to achieve the previous objectives, the thesis undertakes the following specific tasks:

- Develop the use case analysis diagram for defining the interaction between different entities.
- Design protocols for secure information exchange between entities.
- Develop the system architecture.

- Develop the necessary smart contracts.
- Develop and design the application.
- Do performance evaluations and security analysis of the proposed solutions.

1.3 Summary of contributions

The Protocol in Chapter 3 proposes an invoice factoring registration architecture and its associated protocol based on a public blockchain. Using a public blockchain as trust anchor significantly helps the factoring registration process, avoiding manual steps and reducing the power of trusted third parties. In our protocol, we assume that the buyer is trusted by the seller and the factor, being our main use case for buyers that are governments and administrations. Buyers make payments off-chain using fiat transfers between bank accounts. As a consequence, our architecture is a blockchain-based registration system and not a payment system based on blockchain. As a general requirement, we try to spare the buyer from as much complexity and responsibility as possible. In particular, the buyer does not need to have a digital certificate or perform any digital signatures. He just needs to provide a Web Service with some public information.

Regarding availability, there are other factoring systems (like [8]) that use distributed storage systems such as IPFS [9] to provide a certain level of data availability. In our design, we provide the buyer with the highest possible availability for the payment data. The highest availability is provided by on-chain data, which is ultra replicated. So our protocol stores relevant payment data for the buyer on the blockchain. In particular, the buyer reads on-chain data to obtain the bank account where he has to make a payment, in case the invoice has been factored. Obviously, according to our general requirement of involving the buyer as little as possible, the buyer does not have to send any transactions to the blockchain, and is just requested to read from it.

Storing data on-chain creates some challenges for our protocol: we need to provide security and privacy as well as optimizing the number of blockchain transactions and blockchain storage used by the protocol. We ensure security

and privacy, by using commitments and symmetric encryption for data stored on-chain. In particular, symmetric keys are exchanged using an asynchronous version of the well-known Diffie-Hellman protocol [10]. Regarding optimal on-chain registration, we manage to register an invoice with only one transaction and one key-value of storage.

Additionally, we provide evidence for dispute solving between the seller and the factor. Again, according to our general requirement of involving the buyer as little as possible, the buyer is not involved in dispute resolutions after the factoring is completed. As we will demonstrate, only public information and evidences registered in the blockchain are enough to solve disputes without further intervention from the buyer.

In the protocol described in Chapter 4, we propose a protocol that is optimal in terms of cost, and that is built over a public ledger, which is the most reliable, transparent, and secure type of ledger. Regarding other proposals, we address the entry barrier related to the fact that users have to manage cryptocurrencies for interacting with public ledgers. In general, many users, prefer not to use cryptocurrencies because they are highly volatile, risky, and non-compliant. While we rely on a public distributed ledger, the parties are not required to use cryptocurrencies. In particular, we include a relayer in our architecture. Nonetheless, in our protocol, the buyer is not required to invest too many resources, nor be heavily involved in the factoring process.

Additionally, the proposed protocol also enables new functionality that is not available in any other related protocol. Specifically, the protocol allows the involved parties identify each other in a secure and privacy-preserving manner to comply with the KYC regulation. They implement self-sovereign identities making use of self-managed identifiers.

The protocol also leverages the concept of Verifiable Credentials (VCs), which are credentials issued to self-sovereign identities and grant permission to the parties to participate in our invoice factoring architecture. Another advantage of using DIDs is that we can relay on new communications models that are being developed in this ecosystem, like DIDComm. DIDComm allows us to implement asynchronous and secure off-chain communications between participants, which means that a party does not need to be present at the moment that another

party sends a message. The response can be received, processed, and approved asynchronously.

In Chapter 5, we propose a final protocol that uses zero knowledge proofs to prove certain aspects of the invoice factoring procedure without revealing critical and confidential parts. The protocol is built upon our previous protocols and incorporates new cryptographic constructs, namely non-interactive zero-knowledge proofs and cryptographic commitments. The extensions enable the smart contract to thoroughly verify factoring agreements before registering them without compromising the privacy of the involved parties.

1.4 Methodology

In this section, we review the general methodology used in our three protocols.

For the design of our ledger-based invoice factoring solution, we followed the design-science research methodology [11]. Following this methodology, we reviewed the literature to better understand invoice factoring services in the first step. Our study revealed that prevention of double-factoring is the critical motivation behind such services.

In the second step, the distributed-ledger technology motivated us to search for a solution based on this technology to completely prevent double-factoring while satisfying the other requirements. We reviewed several works which implemented new generation invoice factoring protocols using distributed ledger technologies [1, 5, 8]. However, as we discuss in detail in the subsequent chapters, none of them was completely able to fulfill our objectives (mentioned in section 1.2.)

In the third step, we designed system architectures, and developed procedures and communication protocols based on the distributed ledger technology for efficient invoice factoring systems. We made use of proven and solid cryptographic primitives to make our design secure while being functional and efficient. We briefly introduce the cryptographic primitives and the technology we use in our designs in Chapter 2.

We followed an iterative and incremental process in which we started by a base design, and improved it in two increments (Chapters 4 and 5) until we

reached our final design. We had our mentioned objectives in mind during the whole process. Nonetheless, in each iteration, we most focused on specific objectives. In particular, in the base design (presented in Chapter 3), we focused on our base blockchain-based architecture, key management, non-repudiation of the factoring agreement, and implementing the smart contract in the most efficient way. In the second increment, we focused on identity management and better user on-boarding. And in the final increment, we used digital certificates for world-wide adaptability. In addition, we enhanced our protocol with zero-knowledge proofs in order to let the smart contract better verify a factoring agreement for both compliance and authenticity before registering it, while preserving the privacy of the involved parties. The level of the details verified by the smart contract vastly minimizes the chance that the buyer refuses to pay in the last stage.

The fourth step was to evaluate our solution and compare it with the related work. The analyses and comparisons are included besides each version of our protocol in its respective chapter.

1.5 Resulting publications

The thesis work has yielded several scientific publications in international journals ranked in the Journal Citation Reports (JCR). The accepted publications [12],[13] are listed below:

1. Mohammadzadeh, Nasibeh, Sadegh Dorri Nogoorani, and José Luis Muñoz-Tapia. "Invoice Factoring Registration Based on a Public Blockchain." IEEE Access 9 (2021): 24221-24233. Impact Factor:3.9/Q1.

2. Mohammadzadeh, Nasibeh, Sadegh Dorri Nogoorani, and José Luis Muñoz-Tapia. "Decentralized Factoring for Self-Sovereign Identities." Electronics 10.12 (2021): 1467. Impact Factor:2.412/Q2.

Now, we are submitting an article to Computers & Security Elsevier Journal (Impact Factor: 4.438/ Q1) entitled "Enhanced Privacy with Zero-Knowledge Proofs"

1.6 Thesis Outline

This section outlines the organizational structure of the thesis document. Specifically, the document is structured in six chapters:

- Chapter 1 of the thesis gives a high-level introduction to the thesis.
- Chapter 2 explains the required background for our protocols.
- Chapter 3 is devoted to the first protocol for invoice factoring registration based on a public blockchain. This chapter mainly discusses how to apply blockchain technology to solve the problem of double factoring.
- Chapter 4 introduces the second protocol which is developed in the context of a decentralized factoring system for self-sovereign identities. The description includes the design of an improved protocol for which we examine the security and the trust-building between the parties, again focusing on preventing double factoring.
- Chapter 5 introduces our final protocol that uses zero knowledge proofs to prove certain aspects of the invoice factoring procedure without revealing critical and confidential parts.
- Chapter 6 introduces a summary of the results of the thesis, the future work, and the thesis conclusion.

Chapter 2

Background

In this background chapter, we explained the background of the tools utilized in our dissertation on three protocols. At first, the Public Distributed Ledgers section 2.1 expresses the Distributed Ledgers, blockchain networks, Ethereum, and the advantage of using smart contracts. Then, a description Cryptographic Primitives section 2.2, including the security mechanism, a brief illustration of the digital signature, Diffie-Hellmann (DH) Key Exchange protocol, and hash function that we used on the first and second protocols 3,4. Third, in the Decentralized Identifiers section 2.3, we presented self-sovereign identities under complete control of their related entity and DID methods used in the second protocol 4.

Then, we briefly explain about asynchronous communication protocol in the DIDComm section 2.4. We briefly define how direct and indirect messaging work in DIDComm. We utilized DIDcomm technology for the second protocol. At last, the zero-knowledge proof section 2.5 describes zero-knowledge poof and the advantage of zk-SNARK. In the final protocol in chapter 5, we use the zero-knowledge proof.

2.1 Public Distributed Ledgers

The main technology to build a public ledger is a blockchain network. In a blockchain network, users can run a blockchain node to send their transactions or use some available node that allows them to do so. Then, in a distributed way,

the blockchain network can create a unique sequence of ordered transactions. In more detail, the network creates a chain of blocks using a consensus algorithm to order transactions [14]. A block contains several transactions, and an important property is that, once the consensus algorithm definitively accepts a block, this block will be known by all the nodes and it will be impossible to manipulate or delete it [15].

In a blockchain network, users can own one or more accounts. Accounts are identified via a public identifier (usually derived from a random public key using a hash function). New blockchain accounts can be created by simply generating a pair of asymmetric keys and deriving the account identifier from the public key. In general, account identifiers are not directly linked with any user data, so they can be considered pseudo-anonymous identifiers.

Transactions carry the source account identifier and a destination account identifier, and they are all digitally signed using the private key of the source account. All the nodes that form the blockchain network see the same state (also known as world state) that results from executing all the transactions in order [16].

In most current public ledgers, the main use of blockchain is to create a cryptocurrency. As a result, the ledger state represents the balance of each account, and transactions are used to transfer the balance from one account to another. However, blockchain networks can be used to build other generic applications, like we will do for registering the factoring process. For this purpose, many distributed ledgers also provide users with the ability to use smart contracts [17].

Ethereum [18] is the most popular public blockchain capable of running smart contracts, and the platform of choice for many developers for implementing applications with blockchain [19]. Taking Ethereum as a reference, we can define a smart contract as code that implements business logic to manage a portion of the ledger state. Smart contracts are deployed (installed) in the ledger through transactions. Deployed contracts, like user accounts, also have an identifier. Then, the portion of the ledger state which is controlled by the smart contract can be modified by sending a transaction to a function of that smart contract. In this case, the smart contract makes the corresponding state changes according to its explicit and immutable logic. Moreover, once a smart contract is deployed on the blockchain, it can be automatically executed through transactions. The

correct operation of smart contracts is guaranteed by thousands of nodes all over the world, so smart contracts cannot be censured or stopped [20].

The main advantages of implementing business logic using smart contracts are that, on the one hand, the logic is publicly available and auditable, and on the other hand, the logic is immutable and tamper-proof, which guarantees that the execution will always be as defined. These advantages can be used to enforce the terms of an agreement between parties without the need for intermediaries [21].

2.2 Cryptographic Primitives

A number of cryptographic primitives have been used to secure our designs, and we briefly introduce them next. The interested reader is referred to [22] for more detail.

Encryption is a security mechanism to make data confidential. In particular, the data is transformed to a sequence of random-looking bytes that can only be understood by intended parties that have access to a decryption key. There are two types of encryption: symmetric and asymmetric. In symmetric encryption, a secret key is shared between intended parties and is used for both encryption and decryption. In contrast, in asymmetric encryption, a pair of keys (public and private) are used. The public key is available to everyone and can be used to encrypt data, but only one entity owns the private key and can decrypt the encrypted data.

A digital signature is a security mechanism to provide assurance about the originality of signed data and confirm the signatory's informed consent. Digital signatures are a method of public-key (asymmetric) cryptography. More specifically, the private key is used to generate a fixed-size signature from the data, and everybody can validate the signature by the corresponding public key. If a fake private key is used or the data is manipulated, the signature does not match the data and the public key.

A hash function is a cryptographic primitive to derive a fixed-size digest of its input data. Secure hash functions (such as SHA-256) are irreversible in practice, and the original data cannot be guessed from their output value. However, brute-force guessing attacks are still possible if the length of the input is too

short. Therefore, special families of hash functions with configurable (sliding) time-complexity and memory-consumption are used in security protocols to reduce the vulnerability of online and offline brute-force attacks. These algorithms (such as scrypt [23]) are built around the idea of iteratively applying the input data and a random number (a.k.a. the salt) to a secure cryptographic hash function. The salt is used to increase the cost of pre-computation.

Diffie-Hellmann (DH) Key Exchange protocol [10] is the first key exchange protocol in a public-key (asymmetric) setting. It allows two parties to create a shared secret (key) without any prior secret sharing and secure it through an insecure communication channel. In a DH key exchange, participants agree on a finite cyclic group $\mathbb{G}$ of order n and a generator $g \in \mathbb{G}$. One party selects a random number $b_1 \in (1, n)$ and sends g^{b_1} to the other party. Then, the other party selects another random number $b_2 \in (1, n)$ and sends g^{b_2}. The agreed DH secret is $g^{b_1 b_2}$. In order to use DH key exchange securely, the two ends should authenticate the received values to prevent man-in-the-middle attacks and apply a key derivation function (KDF) to the agreed DH secret.

2.3 Decentralized Identifiers

Identifiers (IDs), as their name suggests, are used to identify and distinguish between individuals/entities in the digital world. There are two important properties that an ID shall have: uniqueness and verifiability. The former property guarantees that two different entities do not have the same ID, and the latter one requires that the link between an ID and the related entity must be provable. Both properties are often provided by relying on a central server or a third-party called identity provider. Decentralized identifiers (DIDs) [24] are a means to implement self-sovereign identities (SSIs)—IDs that are under full control of their related entity. DIDs are designed to allow for a verifiable and decentralized digital identity system for subjects and to decouple them from centralized registries, ID providers, and certificate authorities. A DID has a controller which has full power over the subject of the DID without requesting permission from any other entity. Other entities may only facilitate the discovery of information related to a DID.

A DID is a simple text in form of a URI, e.g., did:bc:1234, consisting of a URI scheme identifier (did), a DID method (bc), and a DID method-specific identifier (1234). This opaque string associates a DID subject with a DID document (DDO) to ensure secure and reliable interactions among subjects. When a user acknowledges a claim from an issuer, the corresponding DDO is generated. Each DDO can contain public cryptographic material (e.g., public keys and authentication mechanisms) or service endpoints in order to provide a set of mechanisms to reach the subject and communicate with it securely.

A DID method specification explains specific ways for creating, resolving/verifying, updating, and deleting DIDs, and these functionalities are implemented differently for each DID method. A list of registered DID methods (more than 80) and their specifications can be accessed from Reference [25]. Blockchains and distributed ledgers, in general, are suitable candidates for implementing the verifiable data registry required for implementing DIDs. Regardless of the type of blockchain (public, private, permissioned, or permission-less), specific methods are proposed. In particular, there are proposals based on Sovrin, Ethereum, Bitcoin, Tangle, Hyperledger, ICON, Corda, and other blockchains, and some of them are already operational.

For any identity management solution, privacy is a pivotal component; and blockchain-based DIDs must be carefully designed so that their immutable and transparent nature does not impair privacy. The following features of DIDs can implement privacy by design at the very lowest level of infrastructure and for building robust, modern, and privacy-preserving technologies:

- Pairwise-pseudonymous DIDs: In addition to being used as well-known public identifiers, DIDs can be used as private identifiers issued on a per-relationship basis. In this way, subjects can have multiple pairwise-unique DIDs that cannot be correlated without their permission and, therefore, do not compel a subject to have a single DID, like a national ID number.
- Off-chain private data: It is possible, and already implemented in some existing DID methods, that all private data are stored off-chain and shared only over encrypted, private, and peer-to-peer connections. Because there are two risks for storing personally identifiable information (PII) on a public blockchain, even encrypted or hashed: (1) When the information is shared with multiple parties, the encrypted or hashed data becomes a

global correlation point. (2) When the encryption is eventually broken, e.g., by a quantum computer, the data will be accessible forever on an immutable public ledger.

- Selective disclosure: DIDs can open the door for individuals to gain greater control over their personal data by using DIDs and the greater ecosystem of Verifiable Credentials [26] based on them: (1) by privately sharing encrypted digital credentials only with intended parties, or (2) by using zero-knowledge proofs (ZKP) to minimize data leakage. For example, a ZKP enables a user to disclose that he/she is over a certain age without disclosing his/her exact date of birth.

2.4 DIDComm

DIDComm is an asynchronous communication protocol for establishing secure and private channels between parties based on their DIDs [27]. DIDComm supports both centralized and decentralized communication models. Different parties do not need a highly available web server to be accessible for communications. Individuals on semi-connected mobile devices can also exchange messages in a decentralized fashion, and messages can pass through mixed networks, e.g., an email can connect A to B without a direct connection. DIDComm also supports HTTPS endpoints which can be used to communicate with standard HTTP servers over TLS.

All the information required for establishing a DIDComm channel exists in the DDOs of the involved parties. DIDComm uses public-key cryptography, and the privacy of communications are preserved in the sense that third parties do not learn about the content and the sender of a message. Each party utilizes a software agent to process requests and manage keys. All interactions actually take place between the two ends' software agents. An agent can be implemented in a special desktop/mobile application or a web-based application and be run inside a standard web browser.

Next, we briefly explain how direct and indirect messaging work in DIDComm. In the direct case, Alice directly sends a message to the endpoint specified in Bob's DDO [28]. In a decentralized and ad-hoc case, the endpoint is Bob's agent, who is accessible through the Internet [29]. The confidentiality and

integrity of the message are guaranteed by typical public-key cryptography and digital signature. To do so, their agents use the other party's public key, which is specified in his/her DDO.

In the indirect case, Alice and Bob cannot connect directly, and Alice uses an intermediary Relay [30]. She wraps her encrypted message in another message, encrypts the whole, and sends them to a Relay (direct messaging). The Relay decrypts and unwraps the message and forwards it to Bob (direct messaging). Finally, Bob decrypts and recovers the original message.

2.5 Zero-Knowledge Proofs

Goldwasser et al. created the concept of Zero-Knowledge Proofs (ZKPs) [31]. A ZKP ensures that a prover can persuade a verifier that a statement is authentic without disclosing any information beyond the statement itself. ZKPs were revolutionary because for the first time in cryptography, instead of looking for secure communications in which some of the parties are trusted, the goal was to establish trust between distrustful parties. The first generation of these protocols worked the following way: the verifier sent some challenges to the prover to check that, in fact, he knew the secret information, but the answers from the prover did not allow the verifier to reconstruct any part of the secret. Hence, ZKPs not only protect verifiers against malicious provers but also provers from malicious verifiers that want to obtain information from the prover. Since their appearance 30 years ago, there has been a lot of research into developing strong and efficient protocols.

Zero knowledge proofs fulfill the following characteristics:

- Completeness: If a statement is true and both users follow the protocol rules, then the verifier will be convinced about the validity of the proof.
- Soundness: If the statement is incorrect, the verifier will be unconvinced of the proof with an overwhelming probability.
- Zero-knowledge: In any case, the verifier does not obtain information beyond the correctness or incorrectness of the statement.

Among ZKP systems, non-interactive zero-knowledge (NIZK) protocols are the most interesting for our scenarios in which we use distributed ledger technology because the prover can generate all the proof himself without need to interact with the verifier. For this reason, NIZKs are very suitable for blockchain applications because the verifier can create the proof and send it as part of a transaction to a smart contract. Then, the smart contract can act as verifier and perform some action depending on whether the proof is valid or not. Since we are using a NIZK, the proof can be created in advance by the prover and the smart contract does not need to generate any challenge.

Among NIZK protocols, the most interesting ones for blockchain are those ones whose proof size is small and verification time is also short. This type of NIZK protocols that have succinct proof size and constant verification time exist and are called zk-SNARKs [32]. Circuit-based NIKZs are quite revolutionary because they provide a relatively simple interface (the arithmetic circuit) for developers that want to create privacy-enabled applications and abstract the complexity of the underlying proving mechanism.

Chapter 3

Invoice Factoring Registration Based on a Public Blockchain

3.1 Introduction

Invoice factoring is a very useful tool for developing businesses that face liquidity problems. The main property that a factoring system needs to fulfill is to prevent an invoice from being factored twice. In order to prevent double factoring, many factoring ecosystems use one or several centralized entities to register factoring agreements. However, this puts a lot of power in the hands of these centralized entities and makes it difficult for users to dispute situations in which factoring data is unavailable, wrongly recorded or manipulated by negligence or on purpose. In this chapter, we propose an architecture for invoice factoring registration based on a public blockchain. To solve the aforementioned drawbacks, we replace the trusted third parties for factoring registration with a smart contract. Using a smart contract, we record digital evidence of the terms and conditions of factoring agreements in explicit detail, allowing auditability and dispute resolution. Relevant information is highly available on the blockchain while its privacy is protected. The registration is optimal, since it needs only one blockchain transaction and one key-value storage per invoice factoring.

In this chapter, we first introduce the proposed architecture, then analyse its security. Afterwards, we review the related works and compare them with our proposal.

3.2 Proposed Architecture

In our architecture, we have the three classical entities of the factoring scenario—namely the buyer, the seller, and the factor—as well as a smart contract deployed on a public blockchain. At a high level, our protocol works as follows (see Fig. 3.1):

1. The seller submits a request to the buyer for publishing the invoice.
2. The buyer publishes a cryptographic digest of the invoice in a Web Service.
3. The seller negotiates with several factoring companies and chooses a desired factor.
4. The factor verifies the cryptographic digest of the invoice by accessing the buyer's Web Service.
5. The seller registers the appropriate factoring agreement in a smart contact that is available on a public blockchain.
6. The factor queries the smart contact to check that he/she has been selected.
7. Since the factoring decision registered in the smart contact is immutable, the factor pays the agreed amount (invoice amount minus some fee) to the seller.
8. When the invoice payment deadline is reached, the buyer checks the smart contract and notices that the invoice is factored.
9. Finally, the buyer pays the invoice amount to the factor.

Next, we present a detailed explanation of our proposal including our design goals, assumptions, setup and the detailed protocol.

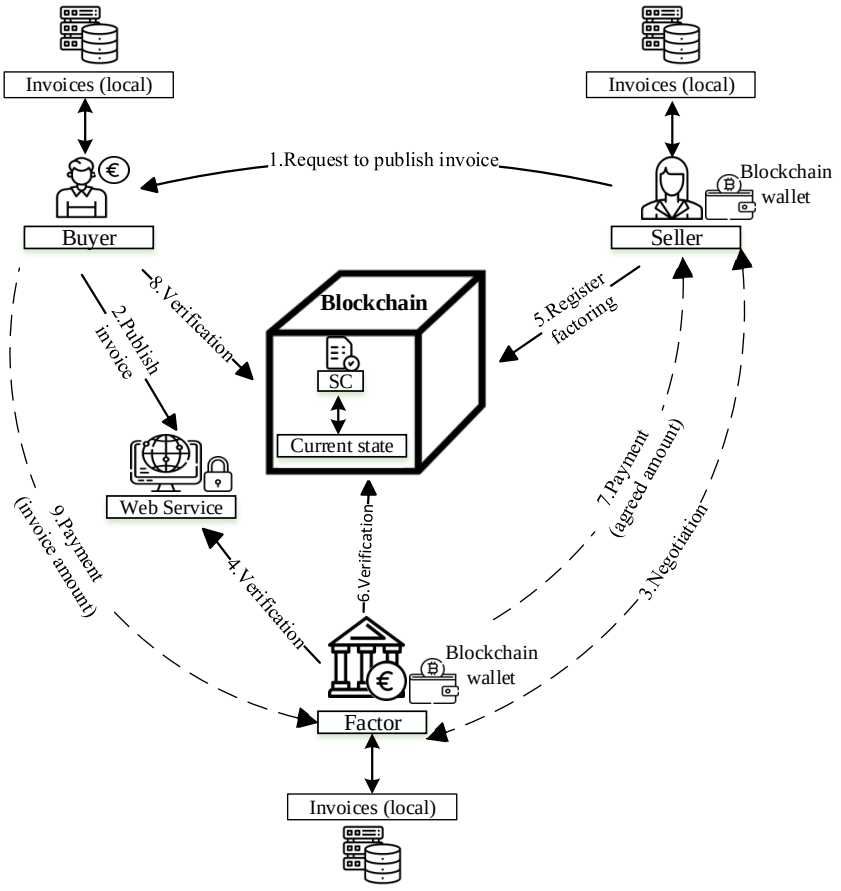

Figure 3.1: Our architecture.

3.2.1 Design Goals & Assumptions

In our architecture, we assume that the buyer is trustworthy for the factoring process. This is clearly true when the buyer is an administration or a government, which is our main use case[1]. In the case of other types of buyers, the factor would need to check the corresponding creditworthiness before accepting to factor invoices issued by a specific buyer.

We also assume a public blockchain for our architecture, but we must remark that we do not use blockchain cryptocurrencies for payments. Our architecture is a registration system, and the actual payments are made off-chain using fiat transfers between bank accounts.

All the interactions to complete a factoring registry will be managed by a smart contract. All parties can trust the correct execution of transactions managed by

[1]We would like to specially thank Francesc Cubel from the Economics Department of the Generalitat of Catalonia (Government of Catalonia in Spain) for collaborating with us in the definition of the requirements for this use case.

the smart contract because the blockchain platform guarantees this execution. If the invoice has been factored, the buyer has to pay the invoice to the bank account of the entity registered by the smart contract. Therefore, all involved parties have to review the smart contract code and ensure its correctness. The smart contract address is also part of the negotiation between the seller and the factor.

Since each transaction that changes the state of a public blockchain has a cost, one of our main design goals is to have the minimum possible number of transactions for completing a factoring registry. Actually, we only use one transaction per invoice factoring and much of the communications between the different parties are off-chain.

Since the buyer does not have incentives in the factoring process, we prevent him from sending transactions to the blockchain. As a general rule, in our design, the factoring process is as less complex and resource-consuming as possible for the buyer. In particular, in our architecture, the buyer will not need specific digital certificates for the factoring process and will not perform digital signatures related to this process. Instead, the buyer will provide a simple Web Service to give access to some minimal information about his invoices so that the factor can check the information provided by the seller.

Another issue to take into account is that, when using a public ledger we gain transparency, but at the same time, everybody has access to the stored data. In the factoring process, there is sensitive business information which shall be appropriately protected. For privacy protection, we do not store sensitive data directly on the blockchain. Instead, some part of the data is symmetrically encrypted before being stored on-chain; another part of the data is stored off-chain, and we use cryptographic commitments to provide proofs of existence. Once an invoice factoring has been registered, we guarantee that:

- There is no possibility of double factoring.
- The relevant parties have access to the relevant data and its proof of existence.
- There is no way to dispute the factoring once the smart contract has registered it.

In addition, to perform the registration process, all the parties will have real identities (e.g., tax identifiers) and the seller and the factor will also have blockchain accounts (which are pseudo-anonymous identifiers).

Finally, we assume that an invoice contains the following information: the seller and the buyer identities, invoice number, issuance date, due payment deadline, total amount (and currency code), and other details about the services/goods provided by the seller to the buyer. We assume that the identity of the seller and the invoice number are enough to uniquely identify the invoice, thus, the use of unique invoice numbers should be enforced. Besides, the identifier of the buyer, due payment deadline, and the total amount are necessary for factoring negotiations. Other information can be added to the invoice without affecting how our architecture works.

3.2.2 Setup

In this section, we describe our key management scheme for the on-chain data encryption. We also describe the concept of Blockchain Certificate, and we provide some preliminary discussions related to our smart contract. We would like to mention that Table ?? contains the notation used throughout this chapter.

3.2.2.1 Key Management

We use a Diffie-Hellman (DH) key exchange scheme [10] to set up our symmetric keys for confidentiality. To use the DH key exchange, participating parties just need to agree on a finite cyclic group $\mathbb{G}$ of order n and a generator $g \in \mathbb{G}$. DH is a two party computation protocol that consists in exchanging two messages. One party selects a random number $x_1 \in (1, n)$ and sends g^{x_1} to the other party. Then, the other party selects another random number $x_2 \in (1, n)$ and sends g^{x_2}. The agreed DH key is $g^{x_1 x_2}$.

We must stress that we do not depend on any DH-specific implementation including Elliptic Curve Diffie-Hellman (ECDH) [33] which is commonly used in the context of blockchain. Moreover, the basic DH key exchange is vulnerable to the man-in-the-middle attack, so in our protocol, all public DH values are either explicitly signed or transferred over authenticated channels. On the other hand,

in the regular use of the DH key exchange, the two parties involved are online, and they exchange two messages over the network to establish a confidential session. In our case, we use persistent storage to allow an asynchronous key exchange in which one party provides a message that will be accessed by the other party in the future. As we explain later, depending on the key being created, the persistent storage used is either the blockchain or the Web Service provided by the buyer. Finally, we use a Key Derivation Function (KDF) to derive secure and random symmetric keys based on the DH-agreed key.

3.2.2.2 Blockchain Certificates

Our architecture is framed in a financial context and hence, strict regulatory restrictions apply to it. In particular, following the Know-Your-Customer (KYC) regulation, the involved parties need to be well identified to each other, and their agreements have to be persisted for later audits and law enforcement.

In order to comply with the KYC regulation, the seller and the factor will register the correspondence between their real identity (ID_A) and their pseudo-anonymous identifier in the blockchain ($@A$). We use the term Blockchain Certificate to refer to these links between real identities and pseudo-identities. In our architecture, we rely on the buyer to create these links, because the buyer is supposed to pay to the factor, and therefore, we can assume that factors can trust buyers to certify sellers.

On the other hand, by design, our protocol avoids the buyer having to digitally sign the Blockchain Certificates or any other data. Our Blockchain Certificates are privately used and are only exchanged between a seller and a factor after they intend to make an agreement. In addition, an entity can have multiple Blockchain Certificates with different blockchain addresses to have additional protections from linking attacks.

Let's consider that the buyer is going to issue a Blockchain Certificate C_A for some entity A. The certificate will link the real identity of A (ID_A) with one of his/her identities in the blockchain ($@A$). To create the Blockchain Certificate, A chooses a random number $x_1 \in (1, n)$, and sends g^{x_1} to the buyer. The buyer chooses another random number $x_2 \in (1, n)$ and derives a symmetric key K_{AB}

using a deterministic KDF:

$$K_{AB} = \text{KDF}((g^{x_1})^{x_2})) \tag{3.1}$$

Next, the buyer calculates a pseudo-anonymous identifier for A (P_A) as follows:

$$P_A = \text{MAC}(K_{AB}, (ID_A, @A)) \tag{3.2}$$

Notice that without further information, the pseudo-anonymous identifier P_A does not reveal any information about ID_A or $@A$. Then, the buyer publishes P_A and g^{x_2} through his Web Service:

$$B \rightarrow WS: \quad P_A, g^{x_2} \tag{3.3}$$

The value of x_2 is not needed anymore and the buyer can discard it, if desired. Now, A can provide the tuple $(P_A, x_1, ID_A, @A)$ to anyone interested to verify his/her identity with the help of the buyer. We define this tuple as the Blockchain Certificate of A (C_A):

$$C_A = (P_A, x_1, ID_A, @A) \tag{3.4}$$

The verification of a Blockchain Certificate involves using the P_A in the certificate as the key in the buyer's Web Service to obtain g^{x_2}. To accept the identity of A, P_A is recalculated from $(g^{x_2}, x_1, ID_A, @A)$ which should match the P_A in the certificate.

Unlike regular X.509 certificates, our certificates do not require digital signatures. Instead, to check the validity of the certificate, some information has to be retrieved from the Web Service. Moreover, our certificates are private, meaning that they are only exchanged between intended parties.

On the other hand, for better anonymity and prevention of linking attacks, each entity can have multiple Blockchain Certificates (with different blockchain addresses). Regarding the role of Blockchain Certificates in our protocol, they are needed for the seller and the factor. As mentioned, sellers and factors can

obtain as many Blockchain Certificates as desired:

$$C_S^j = (P_S^j, s_1^j, ID_S, @S^j) \tag{3.5}$$

$$C_F^l = (P_F^l, f_1^l, ID_F, @F^l) \tag{3.6}$$

where j and l are the indexes of particular certificates. The buyer has to publish the related parameters of these certificates in the Web Service:

$$B \rightarrow WS: \quad P_S^j, g^{s_2^j}, P_F^l, g^{f_2^l} \qquad \forall j, l \tag{3.7}$$

A particular invoice will be factored with one particular pair of Blockchain Certificates of the seller and the factor. For the sake of simplicity, from now on we will simply denote this pair of certificates as (C_S, C_F). As we show later, we follow a similar scheme to publish invoices and factoring information while protecting privacy.

3.2.2.3 The Smart Contract

Our protocol is built around a smart contract, which is deployed on a public blockchain. The smart contract will hold registration data for a set of factored invoices. No one (including its deployer) will have special powers over the contract. In particular, no one will be able to interfere with the smart contract operation or alter any data of the set of factored invoices.

We would like to emphasize that our architecture is designed to operate on a public blockchain. Public blockchains have costs, so, our protocol needs to be cost-efficient. In general, there are three different places in which data is stored on the blockchain (see Fig. 3.2): (i) transaction input data, (ii) key-value storage, and (iii) transaction output logs. Each of these places has a different purpose and a different cost.

The transaction input data is the data in the transaction that provides the inputs to execute the smart contract logic for the current transaction. The transaction input data is very cheap compared to the key-value storage, which is by far the most expensive storage. The key-value storage provides storage to the smart contract that persists between transactions. The key-value storage is part of the current blockchain global state and as such, it can be used by

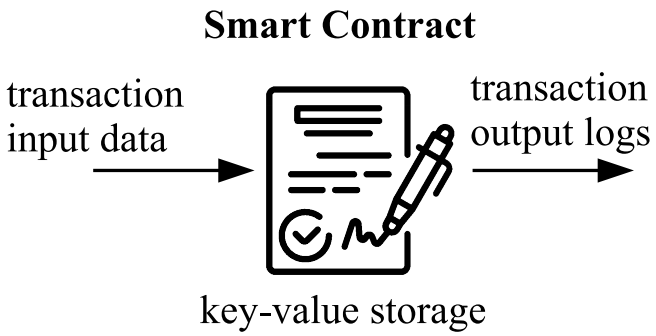

Figure 3.2: Summary of smart contract storage possibilities.

the smart contract logic for the execution of future transactions. Finally, the transaction output logs are data produced after a transaction is executed [2].

We would like to highlight that the transaction input data and the transaction output logs are part of the blockchain data, and as such, they are highly available and immutable. However, these data are not part of the blockchain's current global state which means that blockchain nodes do not need to keep these data in their current state once the transaction has been executed. This is the reason why these data are cheap to store and also the reason why the data of a log from a previous transaction is not available to the logic executing a posterior transaction.

In our protocol, we use a combination of the previous three storage places to provide an efficient implementation while preserving the architecture's privacy and security. In particular, we only use one persistent key-value slot to prevent double factoring. The factoring data is recorded using a transaction output log. We must mention that the data in transaction output logs typically have indexed fields that allow external entities to do quick searches based on these index fields. In our protocol, we use a pseudo-anonymous identifier for the invoice as an indexed log field to speed up the search of the associated factoring data.

The data registered by the smart contract will provide high availability for relevant data that the buyer needs to know, like the factor's bank account. Since a bank account is sensitive data, we encrypt this data before storing it on-chain. In addition, we store a proof of the summary of the factoring agreement in the form of a cryptographic commitment. Storing the summary of the factoring

[2]For example, the transaction logs in Ethereum smart contracts programmed with Solidity are implemented by emitting events (see https://docs.soliditylang.org/en/v0.6.7/contracts.html#events for further information).

agreement on-chain can be used to handle possible future disputes between the seller and the factor. This agreement summary has to be signed by both the seller and the factor.

Finally, we would like to mention that our smart contract stores the blockchain addresses of the seller and the factor on-chain as part of the factoring registration. To do so, we use the public key recovery mechanism available in signature schemes like the Elliptic Curve Digital Signature Algorithm (ECDSA) [33], which is used by many blockchains (e.g., Bitcoin-like blockchains and Ethereum). The public key recovery mechanism allows, given a message m and the signer's signature on that message σ_m^A, to recover the public key pk_A of the signer A. In the case of blockchain, from the public key we can also get the blockchain address (account) of the signer. In our protocol, as we will show, we use transactions that include signatures using blockchain identities of both the seller and the factor, and we will recover their blockchain addresses from these signatures.

In the following sections, we provide the details of the complete factoring process using our protocol.

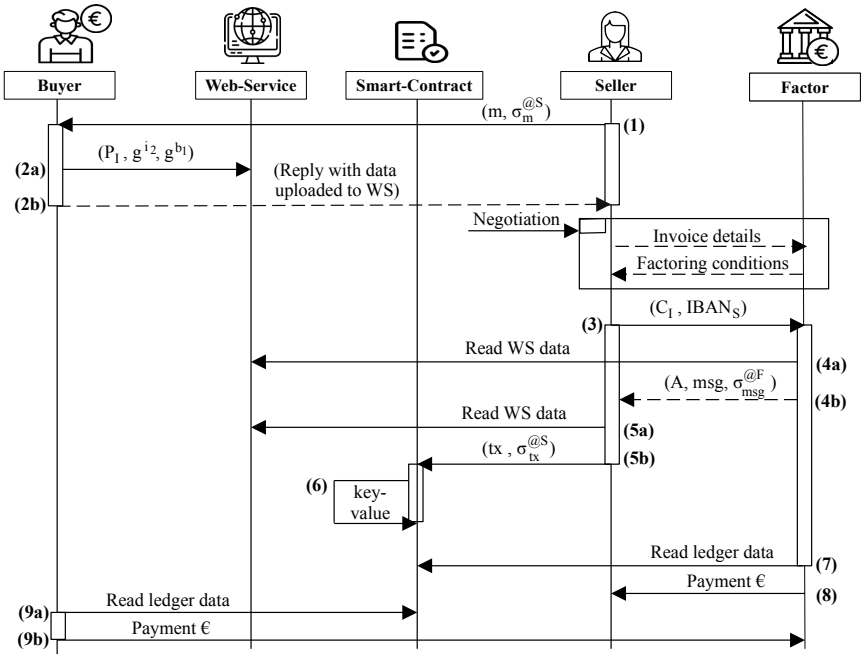

Figure 3.3: Our protocol.

3.2.3 Phase 1: Registration

The process of factoring a specific invoice starts with the registration phase and it is followed by factoring and payment phases. Each phase consists of several steps, which are depicted in Fig. 3.3 and explained subsequently.

At the beginning of the registration phase, the seller asks the buyer to publish invoice information through his Web Service. The publication is quite similar to the way Blockchain Certificates are published, except that additional information related to the factoring process is required. To start the process, the seller selects an invoice I and performs the following steps:

1. The seller chooses a random number $i_1 \in (1, n)$. Then, the seller sends the following signed request to the buyer through a secure channel:

$$m = (C_S, I, g^{i_1}) \tag{3.8}$$

$$S \rightarrow B : m, \sigma_m^{@S} \tag{3.9}$$

2. The buyer checks that the invoice has not been published before (according to ID_S and I). In such case, the buyer checks the signature and proceeds by selecting a random number $i_2 \in (1, n)$ and computing K_{SB} and $\mathcal{P}_I$ as follows:

$$K_{SB} = \text{KDF}((g^{i_1})^{i_2}) \tag{3.10}$$

$$\mathcal{P}_I = \text{MAC}(K_{SB}, (C_S, I, a_I, d_I, @C)) \tag{3.11}$$

   where a_I is the invoice amount, d_I is the invoice payment deadline, and $@C$ is the blockchain address of the smart contract. $\mathcal{P}_I$ is used as the pseudo-anonymous identifier of the invoice. Notice that without further information, $\mathcal{P}_I$ does not reveal any information about the invoice details.

2a. Then, the buyer selects another random number $b_1 \in (1, n)$ and publishes the following information through his Web Service:

$$B \rightarrow WS : \quad \mathcal{P}_I, g^{i_2}, g^{b_1} \tag{3.12}$$

   The value of i_2 will not be needed anymore and can be discarded. However, the buyer does need to keep b_1. This is because the factor will store

his/her International Bank Account Identifier (IBAN) for the payment in a symmetrically encrypted manner on-chain. In more detail, the selected factor will choose a random number $b_2 \in (1, n)$ and provide g^{b_2} on-chain to establish a symmetric key with the buyer (K_{FB}).

2b. Optionally, the buyer might reply to the seller with the same information published in the Web Service. This reply can be omitted and let the seller read this data directly from the Web Service.

As already mentioned, the seller may have multiple blockchain addresses for better anonymity. However, an invoice can be registered only with one of these addresses (notice that for this purpose the address used by the seller is included in the computation of $\mathcal{P}_I$). We define the Invoice Certificate ($\mathcal{C}_I$) as:

$$\mathcal{C}_I = (\mathcal{P}_I, i_1, C_S, I, a_I, d_I, @C) \tag{3.13}$$

Also note that:

- The buyer does not need to perform digital signatures, he only does small computations once per invoice.
- The seller authenticates the Web Service (buyer) before sending the request, and the connection is secured by HTTPS, thus preserving her privacy.
- Only a MAC and two public DH values are published for better privacy protection. Privacy is protected because an external party cannot obtain any identity agreed with the buyer by just having access to the DH public values. This protects the privacy of the corresponding sellers, factors and invoices.

3.2.4 Phase 2: Factoring

This phase starts with the seller contacting multiple factors over an out-of-band but private channel to negotiate and compare the different offers and conditions. The seller should naturally provide her invoice details, including invoice number

(I), the total amount of the invoice (a_I), and payment due deadline (d_I) to the possible factors. Then, according to the received offers, the seller selects the best factor to continue with.

The selected factor must provide his/her certificate C_F to the seller. Using the buyer's Web Service, both the seller and the factor must verify that the certificates from the other party are valid.

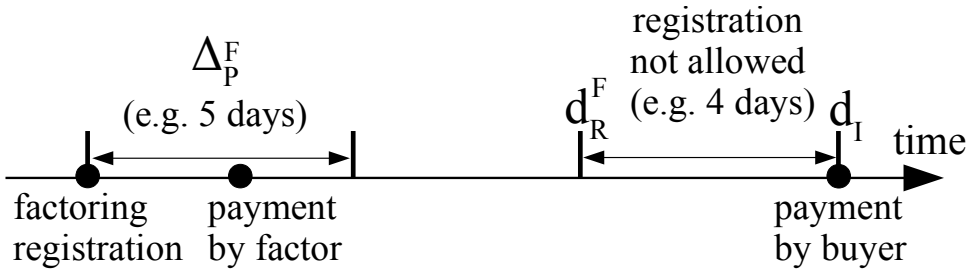

Figure 3.4: Timestamps and periods for factoring registration and payment.

The factors also specify their offered amount for the invoice ($a_F = a_I - fee$) and two time parameters Δ_P^F and d_R^F (see Fig. 3.4):

- Δ_P^F: maximum period of time that the factor can last in paying the seller, after the factoring is successfully registered by the smart contract.
- d_R^F: deadline for completing the factoring registration. This deadline is important and it is enforced by the smart contract. It should be long enough to let the seller register the invoice factoring. Since the seller is the most interested party in doing the factoring, she will probably take the decision and register the factoring as soon as possible. However, the deadline set by the factor should be far enough from the due deadline of the invoice to prevent the seller from performing a late registration and trying to get paid by both the buyer and the factor. E.g., if invoices are paid at 90 days, the deadline could be set by factors, for example, to 4 days before the invoice due deadline:

$$d_R^F = d_I - 4 \text{ days} \tag{3.14}$$

  In this case, $d_R^F = 4$ days is far enough from the invoice due deadline, so that when the buyer reads the data in the blockchain, the data is stable: the invoice is either factored or cannot be factored, and there is no possibility of double factoring. If d_R^F is too close to the invoice due deadline,

it could happen that the buyer pays to the seller, the seller registers the factor in the smart contract and the factor reads the blockchain and also pays the seller. In this case, we assume that an honest seller is not probably going to factor an invoice just 4 days before the date in which he can receive the total amount of the invoice. By the way, this is obviously a configurable parameter.

On the other hand, the messages exchanged between the seller and the selected factor for registering the factoring are the following:

3. Using a secure channel (e.g., HTTPS), the seller sends $\mathcal{C}_I$ (which includes her associated C_S) and her bank account number ($IBAN_S$) to the selected factor:

$$S \rightarrow F: \quad (\mathcal{C}_I, IBAN_S) \tag{3.15}$$

4a. Using the Web Service, the selected factor verifies $\mathcal{C}_I$, which includes the verification of the embedded C_S. If verifications are correct, the factor is sure about the invoice details, including the address of the factoring smart contract. Then, the factor checks the smart contract to make sure that the invoice has not been already factored (if the invoice has been factored the process is obviously canceled.)

4b. The factor chooses a random number $b_2 \in (1, n)$ and uses it to generate a symmetric key encryption K_{FB} that will be used to store his account number ($IBAN_F$) on-chain and in an encrypted manner to receive the invoice amount from the buyer:

$$K_{FB} = \text{KDF}((g^{b_1})^{b_2}) \tag{3.16}$$

$$\text{Enc}(K_{FB}, IBAN_F) \tag{3.17}$$

We define the agreement data ($\mathcal{A}$) between the factor and the seller as follows:

$$\mathcal{A} = (\mathcal{C}_I, C_F, a_F, \Delta_P^F, IBAN_S) \tag{3.18}$$

The factor sends the following data to the seller:

$$F \rightarrow S: \quad (\mathcal{A}, msg, \sigma_{msg}^{@F}) \tag{3.19}$$

where msg is defined as follows:

$$msg = (\mathcal{P}_I, d_R^F, \text{Enc}(K_{FB}, IBAN_F), g^{b_2}, h(\mathcal{A})) \tag{3.20}$$

Notice that the factor produces the signature over msg using his blockchain address $@F$. This is because the smart contract also checks this signature before registering the factoring.

5a. The seller checks that $\sigma_{msg}^{@F}$ is valid, that the agreement $\mathcal{A}$ is correct (including the verification of C_F), and that the hash of the agreement $h(\mathcal{A})$ is also correct. The seller records the signature for possible later use as digital evidence.

5b. Using her blockchain address $@S$, the seller sends a signed transaction tx to the smart contract:

$$S \rightarrow C: \quad (tx, \sigma_{tx}^{@S}) \tag{3.21}$$

where:

$$tx = (msg, \sigma_{msg}^{@F}) \tag{3.22}$$

Notice that the transaction contains the signature of the factor and also the signature of the seller and thus, it is explicitly approved by both parties.

6. The smart contract is designed to ensure the security of the system and to provide an efficient on-chain storage. To do so, the smart contract processes the transaction (tx) as follows:

   - From the transaction signature ($\sigma_{tx}^{@S}$), it recovers the blockchain address of the seller ($@S$).
   - From the signature embedded in the transaction ($\sigma_{msg}^{@F}$), it recovers the blockchain address of the factor ($@F$).
   - From the msg, it gets the deadline for registration set by the factor (d_R^F) and verifies that the current blockchain time is smaller than the registration deadline:

$$\texttt{registration.timestamp} \leq d_R^F \tag{3.23}$$

The registration.timestamp is obtained from the timestamp included in the block that contains the factoring transaction.

The smart contract needs to register the following data related to the factoring:

- $@S$: blockchain address of the seller, which is recovered by the smart contract from the transaction signature.
- $@F$: blockchain address of the factor, which is recovered by the smart contract from the signature of the message embedded in the transaction.
- $\mathcal{P}_I$: pseudo-anonymous identifier of the invoice.
- g^{b_2}: it will be retrieved by the buyer to compute K_{FB}.
- $\mathrm{Enc}(K_{FB}, IBAN_F)$: symmetrically encrypted account number of the factor that will be retrieved by the buyer to make the appropriate payment. Note that by storing this value on-chain we get the high availability and transparency of blockchain while preserving privacy.
- $h(\mathcal{A})$: fingerprint of the factoring agreement. This value can be used by the seller or the factor as a proof of existence in case of dispute. Note that before storing this value, the signature of both the seller and the factor are checked by the smart contract.

7. Using $h(@S, \mathcal{P}_I)$ as index for the log in the smart contract, the factor can get the associated registration data and verify whether the invoice has been assigned to himself or not.

We efficiently store the data by using only one key-value per invoice in the storage of the smart contract as follows. We use $\mathcal{H} = h(@S, \mathcal{P}_I)$ as the key of the registry. Note that $@S$ and $\mathcal{P}_I$ are known to the buyer, so the buyer can compute this hash and use it as key to find the factoring registration in the smart contract. Associated with the key, the smart contract stores the following value ($\mathcal{V}$):

$$\mathcal{V} = h(g^{b_2}, Enc(K_{FB}, IBAN_F), @F, h(\mathcal{A})) \tag{3.24}$$

So, the smart contract will contain the following key-value mapping:

$$\mathcal{H} \Rightarrow \mathcal{V} \tag{3.25}$$

Obviously, the key-value mapping of the smart contract storage can only be set if it was not previously set to a previous value, which prevents double-factoring.

Finally, we need the factoring registration data to be available for the parties: the buyer for paying the factor, and the seller and the factor to have evidence proofs in case of dispute. To do this efficiently, we store this data as a smart contract output transaction log after the successful invoice registration:

$$log(\mathcal{H}, g^{b_2}, Enc(K_{FB}, IBAN_F), @F, h(\mathcal{A})) \tag{3.26}$$

$\mathcal{H}$ is defined as an index field for quick search.

3.2.5 Phase 3: Payment

In the third phase, after checking the registered information by the smart contract, the factor pays a_F to the seller. Later, the buyer will pay the complete invoice amount a_I to the factor.

8. The factor proceeds to pay a_F to the account of the seller ($IBAN_S$). This has to happen before the agreed payment deadline, that is, not later than $registration.timestamp + \Delta_P^F$.

9a. When the deadline of an invoice (d_I) expires, the buyer has to query the smart contract to figure out whether the invoice has been factored or not. The buyer knows the address of the smart contract ($@S$) and the pseudo-anonymous identifier of the invoice ($\mathcal{P}_I$). Using these two values, the buyer computes the hash $\mathcal{H} = h(@S, \mathcal{P}_I)$ and queries a blockchain node to obtain the log with index field $\mathcal{H}$ of the smart contract in address $@S$. From the log, the buyer obtains $Enc(K_{FB}, IBAN_F)$ and g^{b_2}. Using g^{b_2} and the value b_1 that the buyer has stored, he computes K_{FB} and decrypts the bank account of the factor ($IBAN_F$).

9b. Finally, the buyer pays the factor using the $IBAN_F$ decrypted in 9a.

3.3 Security Analysis

Our analysis is divided into several subsections, specifically, the security of Blockchain and Invoice Certificates, communications security, data manipulation attacks, replay attacks, confidentiality and privacy, and fraud handling. In each subsection, we explain security requirement(s), possible attack(s), and our mitigation method(s).

3.3.1 Blockchain and Invoice Certificates

The contents of the Blockchain and Invoice Certificates are not published, and their owners may hand them to other parties at will. Every Certificate is protected by a MAC, which is generated using a fresh DH-generated symmetric key. For a verifier to ensure the authenticity of a Certificate, the MAC is queried over an authenticated HTTPS channel. Therefore, as long as the MAC and HTTPS are secure, the Certificates are secure as well.

3.3.2 Communications' Security

Communications between the Web Service of the buyer and the seller or the factor are conducted over HTTPS. Therefore, the integrity and confidentiality of the requests and responses are guaranteed. A traditional web certificate authenticates the server-side (Web Service), and the messages from the other side are protected by explicit signatures when necessary. These signatures assure the interested party that the other side cannot deny having generated a message.

3.3.3 Data Manipulation and Repudiation Attacks

All data stored on a blockchain are publicly readable; therefore, confidentiality and privacy are more of a concern in comparison to traditional systems. We do not store any information that can be used to identify or trace the seller or the factor on the blockchain. A pseudo-anonymous identifier is used for the seller and the invoice, and other information is encrypted. An asynchronous version

of the DH key exchange is used to generate the encryption keys, and a digital signature is provided by the factor for non-repudiation.

The blockchain offers a unique security feature that protects stored data from malicious manipulation. To be more precise, the data can only be updated or deleted by predefined smart contract methods. If an attacker aims to disrupt the network by taking down one or only a small portion of the network, it will not succeed. This feature makes blockchain technology suitable for transaction data, determining which data is valid or tampered with, and can create a network of untrusted participants. Moreover, once the smart contract registers an invoice, this fact can never be changed.

In addition to the smart contract, the buyer also publishes information. In this respect, sellers and factors in our architecture must trust that the buyer will not publish false information. The information is communicated through secure channels to different parties and cannot be manipulated at the connection level. For non-repudiation purposes, we require the factor and the seller to sign their messages digitally.

3.3.4 Confidentiality and Privacy

Concerns about confidentiality and privacy of financial data are significant in our use case. All private information is transferred over HTTPS or explicitly encrypted. Encryption keys are not shared between multiple parties, and only the intended party can decrypt the information. Pseudo-anonymous identifiers are used in communications and stored on the blockchain to better preserve the involved party's privacy. The seller and the factor can use a fresh blockchain address in each factoring contract to prevent linking attacks.

The buyer in our architecture is not involved in the negotiations between the seller and the factors. In particular, the buyer cannot predict if the seller will factor her invoice or not (before agreement about payment conditions and other invoice details). Moreover, our architecture protects the factors from each other as they do not have access to their competitors' conditions (before and after finalizing the factoring contract). Factors are not notified if the seller applies to multiple factors to obtain a better bid for the invoice.

3.3.5 Dispute Handling

We must remark that our registration protocol is not secure against malicious buyers because if the buyer publishes false information, this can not be disputed by the seller or the factor. As a result, the seller and the factor need to trust the buyer. A malicious buyer, for example, may not pay the seller or the factor. A malicious buyer may also scam factors by creating a fake seller and a high amount of non-existent invoices. Then, the fake seller receives the payments from the factors but the corresponding payments are not made by the malicious buyer. In case the buyer is not trustful, some mechanism to enforce good behavior must be used (like a reputation system as in Guerar et al.[8]).

On the other hand, a seller may be concerned about a malicious factor that may refrain from payment. In this case, the seller reveals the message sent by the factor in Eq. (5.2.5) to a judge. Then, the following steps are sufficient to handle the case:

1. Verification of Invoice Certificate ($\mathcal{C}_I$).
2. Verification of the agreement terms ($\mathcal{A}$).
3. Verification of the address of the seller (C_S).
4. Verification of the address of the factor (C_F).
5. Signature verification: the judge has all the required information about the factoring agreement and can verify the signature of the factor ($\sigma_{msg}^{@F}$) on this information.
6. Determining $h(@S, \mathcal{P}_I)$ (according to $\mathcal{C}_I$).
7. Retrieval of registration information from the smart contract and its logs. The address of the contract is certified by $\mathcal{C}_I$ and the judge has the signature of the factor on it.
8. Trial: The factor will be doomed according to non-repudiation of digital signatures, and tamper-proof evidence from the smart contract.

A factor may also be concerned about the case in which the buyer pays the amount to another bank account. In this case, the factor reveals b_2 (in Eq. (4.8)) and $\mathcal{C}_I$ to a judge. Then, the following steps are sufficient to handle the case:

Table 3.1: Comparison with close related work.

Proposal	Buyer tasks	Currency	Responsible for the cost	Factoring negotiation	Availability	Immutability	Privacy
DecReg [5]	operating a node of a private blockchain	fiat	shared	off-chain	private on-chain	private blockchain	private network
Battaiola et al. [1]	operating a node of a private blockchain	fiat	shared	off-chain	private on-chain	private blockchain	commitments
Guerar et al. [8]	sending 3 transactions per factoring to a public ledger	crypto	shared	on-chain	IPFS	public blockchain	commitments and encryption
Our Proposal	publishing data by a web service	fiat	seller	off-chain	public on-chain	public blockchain	commitments and encryption

1. Verification of Invoice Certificate ($\mathcal{C}_I$) from the Web Service of the suspected buyer.
2. Verification of the address of the factor (C_F).
3. Verification of the address of the seller (C_S).
4. Key confirmation: uses b_2 to verify correctness of g^{b_2} and K_{FB} (according to Eq. (4.8).) The judge can infer that the buyer could also calculate K_{FB} using b_1.
5. Determining $h(@S, \mathcal{P}_I)$ (according to $\mathcal{C}_I$).
6. Retrieval of registration information from the smart contract and its logs. The address of the contract is certified by $\mathcal{C}_I$.
7. Verification of factor's bank account: use of K_{FB} to decrypt $\mathsf{Enc}(K_{FB}, IBAN_F)$.
8. Resolution: the case can be resolved and the culprit can be detected according to the bank account logs.

3.4 Comparison with the Related Works

In this section, we describe the closest related works available in the literature that propose factoring solutions using distributed ledgers. Then, we compare these works with our proposal and provide a comparison in Table 4.1 according to the following parameters: buyer tasks, currency, who pays the cost of factoring registration, factoring negotiation, availability, immutability and privacy.

The first of these related systems is DecReg [5]. DecReg is a framework for preventing double factoring; that has been used by the Netherlands financial

industry and is implemented over a private blockchain. In DecReg, a Central Authority (CA) controls the access to the blockchain and prevents sensitive information from being accessed by uncertified parties. The main drawback of DecReg is that its CA is a centralization point and a single point of failure or corruption. This matter makes it vulnerable to double factoring attacks in case the CA is compromised. For example, if compromised, the CA can deny access of a factor to the network and subsequently, the factor cannot verify if an invoice is already factored or not. Besides, the CA prevents access to the confidential data solely from entities outside the private blockchain network. Data is not encrypted, so entities inside the network have access to these data, and as a result the privacy of the participants is not fully preserved.

In DecReg, unlike in our system, the buyer has to operate a node in the private blockchain, receive credentials from the CA, etc. So he is quite involved in the factoring process. Regarding dispute resolution, if an argument between a seller and a factor takes place, in DecReg, the signatures over transactions are the only proof that can be used. The main problem of this approach is that transactions are not publicly available and the system relies on the CA for managing access to the system. On the contrary, in our system, the agreement commitments and the payment data is publicly registered and accessible to the appropriate parties. In DecReg, availability is provided by the network of the private blockchain, which arguably, provides much fewer data replication than a public blockchain. Finally, it is worth mentioning that like our proposal, DecReg is a registry system in which actual payments are made in fiat.

Battaiola et al. also propose a system for registering factoring agreements while preventing double factoring and preserving the privacy of involved parties [1]. The architecture proposed in [1] uses a distributed ledger as the source of truth where all the parties send their private inputs in the form of commitments to protect the integrity and confidentiality of factoring data. While the idea of using commitments to protect privacy is similar to what we do in our protocol, their protocol is designed to work over a private blockchain network (in particular, Hyperledger Fabric). Authors claim that they can replace HyperLedger Fabric with any other ledger without affecting security. While it is possible, the problem of the protocol in [1] is that it is not cost-optimized. Unlike our protocol, their protocol is not optimized in terms of (i) the number of transactions required to complete a factoring registration (each party needs to send a

transaction to the ledger), and (ii) the amount of persistent storage which is needed to record factoring agreements. In addition, regarding the buyer, his involvement in the factoring process is quite high since he needs credentials in the Hyperledger Fabric network and not only queries the ledger state but also signs and sends transactions.

In [1], the data availability and immutability depends on the security provided by the private blockchain network. A more secure private network involves more nodes and more entities participating, which means a higher operation cost. In particular, in the paper, it is not defined who has to account for the cost of operating the Hyperledger Fabric network. In our protocol, the data availability and immutability is provided by a public network. We do not register only the commitments of factoring agreements, but also payment data on-chain with symmetric encryption using an asynchronously exchanged key between interested parties. This provides to our protocol the highest possible degree of availability and immutability for relevant payment data. On the other hand, in our protocol, the seller (the most interested party) is in-charge of paying the cost of factoring registration. Furthermore, this cost is optimally minimized to only one blockchain transaction that uses just one key-value of blockchain storage. Finally, it is worth mentioning that [1], similar to our proposal, is concerned about creating a practical registry system in which actual payments are made in fiat.

Recently, Guerar et al. have proposed a factoring system using distributed ledger technology [8]. Like our proposal, Guerar et al. use a public blockchain (Ethereum). However, they make quite different assumptions from the ones that we have. In first place, they do not consider buyers as trusted. Making this assumption leads them to build a system that needs to measure the reputation of the buyers based on their past behavior. To build the reputation system, they rely on their platform to assign a stable identifier to each buyer. However, this makes the platform defined in [8] a trusted party. The platform is trusted because it is in charge of creating the stable accounts for reputation, so, if the platform does not correctly certify real identities of buyers, the security of the system is jeopardized. In our protocol, we assume that the buyer, who is the final payer of the invoice, is trustworthy.

Another assumption that is different regarding our system is that Guerar et al. propose an open environment where factors are not only banks and financial

companies but any investor can register into their platform. Furthermore, factoring negotiation is done with an on-chain auction. While interesting, this is not suitable for our type of buyers, that, in particular, can be governments or administrations that need to comply with regulations and identify themselves the possible factors. On the other hand, their system seems to be defined more for products than for services, because authors mention that the invoice factoring negotiation phase starts when transported goods are received. On the contrary, our system is general in this regard and invoices can be created for either goods or services.

In [8], data availability is provided by IPFS. While IPFS can provide a decent level of availability, it is not as high as the one provided by on-chain data and this is important in case the buyer needs to access the payment data with a virtually zero down-time (as in our protocol). Finally, since they do the factoring negotiation on-chain, in [8] the number of transactions required to complete an invoice factoring is much higher than in our protocol. In particular, there is a minimum of seven transactions to complete a factoring. But, the main drawback is that the buyer, who in general does not have many incentives in the factoring process, has to perform three transactions in the public ledger per invoice factoring. In particular, the buyer has to perform one transaction to accept the invoice and pay the shipping, another transaction for confirming the delivery of the goods, and a final transaction is used for paying the entire amount of the invoice to the corresponding factor.

3.5 Conclusions

In this chapter, we proposed an architecture for factoring registration using a public blockchain. Our protocol is designed to minimize the buyer's involvement in the factoring process. The buyer is just supposed to publish a hash of invoice details for verification of factors, and the rest of the process is implemented by sellers and factors having on-chain and off-chain communications. We use a smart contract to register invoice factoring details on-chain in a very efficient manner and to prevent double factoring. At the same time, we use pseudo-anonymous identifiers, symmetric encryption for on-chain data, and cryptographic commitments to increase the sellers' and the factors' privacy protections. The registered information is later used by the buyer to pay the

corresponding factor, and it can also be used as digital evidence for dispute resolutions. The comparison with the related work demonstrated that, while there are other proposals in the literature, none of them are tailored to our requirements or provide a solution as optimal as ours.

In the following chapter, we introduce another party, a relayer, to facilitate interactions with the blockchain. In addition, we improve our identity management to better respect the Know-Your-Customer rule, in a decentralized and privacy-respecting manner.

Chapter 4

Decentralized Factoring for Self-Sovereign Identities

4.1 Introduction

Distributed ledger technology is suitable for implementing the platform to register invoice factoring agreements and prevent double-factoring. Several works have been proposed to use this technology for invoice factoring. However, current proposals lack in one or several aspects, such as decentralization and security against corruption, protecting business and personally identifiable information (PII), providing non-repudiation for handling disputes, Know-Your-Customer (KYC) compliance, easy user on-boarding, and being cost-efficient. In this chapter we build upon our previous protocol introduced in Chapter 3, and add a new party to our architecture, namely the relayer, to address the entry barrier that the users have due to the need of managing cryptocurrencies and interacting with the public ledger. Moreover, we leverage the concept of Verifiable Credentials (VCs) for KYC compliance, and allow the parties to implement their self-sovereign identities by using decentralized identifiers (DIDs). DIDs enable us to relay on the DIDComm protocol for asynchronous and secure off-chain communications.

The rest of this chapter is organized as follows: first we propose our enhanced architecture. Then, we analyze our improved protocol from several security aspects, compare it to the related work, and study a possible business use case.

We conclude this chapter by introducing and studying a proof-of-concept (PoC) implementation of our protocol.

4.2 Proposed Architecture

In this architecture, we have the three classical entities of the factoring scenario: the buyer, the seller, and the factor. Additionally, we have a smart contract deployed on a public distributed ledger and a relayer that facilitates sending the transactions to the ledger. At a high level, our protocol works as follows:

- The seller submits a request to the buyer for registering an invoice.
- The buyer issues a cryptographic digest for the invoice.
- The seller negotiates with several factoring companies and chooses a desired factor.
- The factor verifies the cryptographic digest of the invoice by querying the buyer, and then sends the signed factoring agreement to the seller.
- The seller uses a relayer to register the agreement in a smart contract that is available on a public ledger.
- The factoring company queries the smart contract to ensure that it is actually selected as the factor.
- Since the factoring decision registered in the smart contract is immutable, the factor pays the agreed amount (invoice amount – fee) to the seller.
- When the invoice payment deadline is reached, the buyer checks the smart contract and notices that the invoice is factored.
- Finally, the buyer pays the invoice amount to the factor.

Next, we present a detailed explanation of our proposal, including our design goals, assumptions, and the detailed protocol.

4.2.1 Design Goals & Assumptions

Our basic goals and assumptions are the same as the ones in Chapter 3. To be brief, we focus on their differences.

An important obstacle against the adoption of distributed applications is the need of having cryptocurrency to pay the transaction fees. Managing cryptocurrencies may be difficult for institutions and companies because of their high volatility, financial risk, and regulation issues. This may lead to a situation in which parties refuse to use cryptocurrency and, hence, cannot interact directly with smart contracts.

To overcome this problem, we use a relayer in this architecture, as shown in Figure 4.1. A relayer is a facilitator that sends the transactions on behalf of other users. The relayer will pay the fees, but it is not a trusted party. In more detail, this means that the seller is who authorizes the factoring registrations, and the relayer is an entity to merely forward and pay the transaction fee. The advantage of this architecture is that the seller can pay the relayer with classical payment methods (e.g., credit cards, bank transfer, etc.).

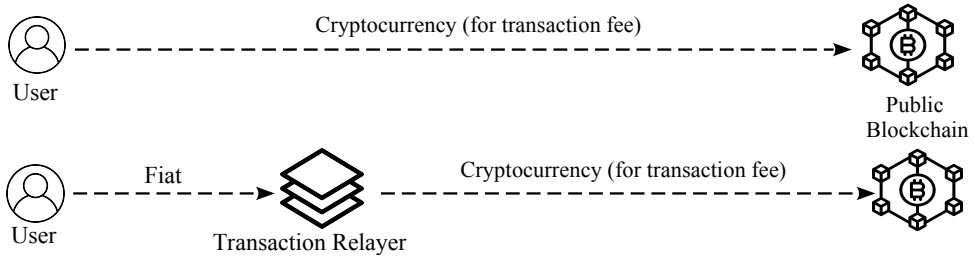

Figure 4.1: Transaction relayer.

Since the buyer does not have incentives in the factoring process, as a general rule, in our design, the factoring process is as less complex and resource-consuming as possible for the buyer. In particular, in our architecture, the buyer will not need specific digital certificates for the factoring process and will not perform digital signatures related to this process. We replace the buyer's Web Service (in Chapter 3) with a software agent which gives access to some minimal information about his invoices so that factors can check the information provided by sellers.

In addition, to perform the registration process, all parties are identified by DIDs, and some communications use DIDComm. Our proposal is independent of any specific DID-method, but we have the following assumptions:

- Sellers and factors may have multiple DIDs, but once a pair decides to enter a factoring agreement, they use a specific DID during the whole process. Their DIDs shall be bound to a pair of digital signature keys to sign requests for non-repudiation purposes.
- While our architecture supports multiple buyers, but we focus on invoices related to one buyer and assume that the respected sellers and factors already know the DID of the buyer.
- We have a relayer role in our architecture which is also identified by a DID. In addition to the DID, the relayer has a blockchain address for issuing transactions. However, there is no need for the DID and the address to be linked together.
- The other entities in our proposal are not required to have blockchain addresses.

4.2.2 The Protocol

Remember that our architecture is framed in a financial context; hence, strict regulatory restrictions apply to it—particularly, the Know-Your-Customer (KYC) regulation. In order to comply with this regulation in this architecture, the buyer issues Verifiable Credentials (VCs) to certify the real identity of the factors, the sellers, and also exact details of the invoices. The buyer is supposed to pay the factor; therefore, as mentioned, we assume that factors can trust buyers for this purpose. Our protocol avoids the buyer from having to digitally sign a VC or any other data. Instead, the authenticity of VCs are verified by securely querying the (agent of the) buyer.

The process of factoring an invoice starts with the registration phase and is followed by factoring and payment phases. Each phase consists of several steps, which are explained in the following sections. In Table ??, we show the notation that we utilize to describe our protocol.

4.2.2.1 Phase 1: Credential Registration

In this phase, VCs are registered by the buyer for identifying factors and for identifying invoices of sellers (seller-invoice VC). Both VCs are registered in essentially the same manner. We first explain the factors' VC registration, and then the seller-invoice VCs.

Factor's credential: The VC of a factor is registered as follows (see Figure 4.2):

1a The factor establishes a mutually authenticated DIDComm channel with the buyer and sends one of its DIDs (DID_F). This channel is re-used for other communications between these two entities.

2a.1 The buyer selects a random number s (salt) and generates an identifier for the credential as follows:

$$P_F = \mathsf{h}(s, (DID_F, PU_F, ID_F)). \tag{4.1}$$

To generate the identifier for the factor (P_F), we use its decentralized identifier (DID_F), the real identity of the factor (ID_F), and the public key of the factor (PU_F). The factor's public key is obtained from the DDO that resolves DID_F. Finally, the verifiable credential for the factor is the following tuple:

$$C_F = (P_F, DID_F, PU_F, ID_F). \tag{4.2}$$

2a.2 The buyer keeps the VC and salt for further reference and replies to the factor with P_F.

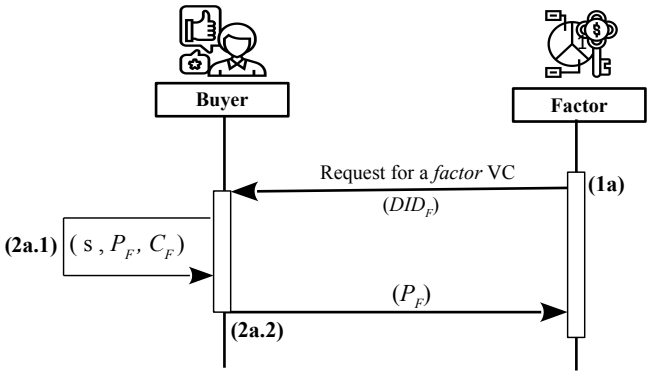

Figure 4.2: Registration of a factor's VC.

Note that having the value of P_F, the factor can compose his/her credential (C_F). After that, any seller with a copy of C_F can consult the buyer's agent, send P_F, and obtain s to check the integrity of the VC content. Note that P_F is cryptographically bound to the contents of C_F. Therefore, if any of the contents are changed, P_F does not match them anymore. Secure hash functions which are resistant against brute-force guessing attacks (such as scrypt [23]) should be used here. They prevent an attacker from discovering the actual content of C_F by trying different values, and matching P_F with the guessed content. The security of the communication with the buyer and pre-image resistance of the hash function assure the authenticity of the VC. The VC is kept private and only exchanged between intended parties. For better anonymity and prevention of linking attacks, a factor can have multiple DIDs but can use only one of them during the whole process of factoring a particular invoice.

Seller-invoice credential: When a seller decides to factor an invoice, she asks the buyer to register a seller-invoice VC. The registration is independent of factors' registration. In particular, this registration consists of the following steps (see Figure 4.3):

1b The seller establishes a mutually authenticated DIDComm channel with the buyer and sends one of its DIDs (DID_S) and the identifier (I) of the corresponding invoice. This channel is re-used for other communications between these two entities.

2b.1 The buyer checks that a credential has not been already registered for the invoice I. Then, they proceed by selecting a random salt s and generating the credential identifier.

A seller-invoice credential is similar to a factor's VC, but it contains not only the seller identifiers but also invoice information:

$$C_I = (P_I, DID_S, PU_S, ID_S, I, a_I, d_I, @C), \tag{4.3}$$

where DID_S and ID_S are the seller's identifiers, PU_S is the public key of the seller, I is the invoice number, a_I is the invoice amount, d_I is the invoice payment deadline, and $@C$ is the blockchain address of the factoring smart contract. Finally, P_I is the identifier of the seller-invoice

credential, which is generated in the same way as in Equation Eq. (4.1), but computing the hash over the contents of C_I:

$$P_I = \mathrm{h}(s, (DID_S, PU_S, ID_S, I, a_I, d_I, @C)). \tag{4.4}$$

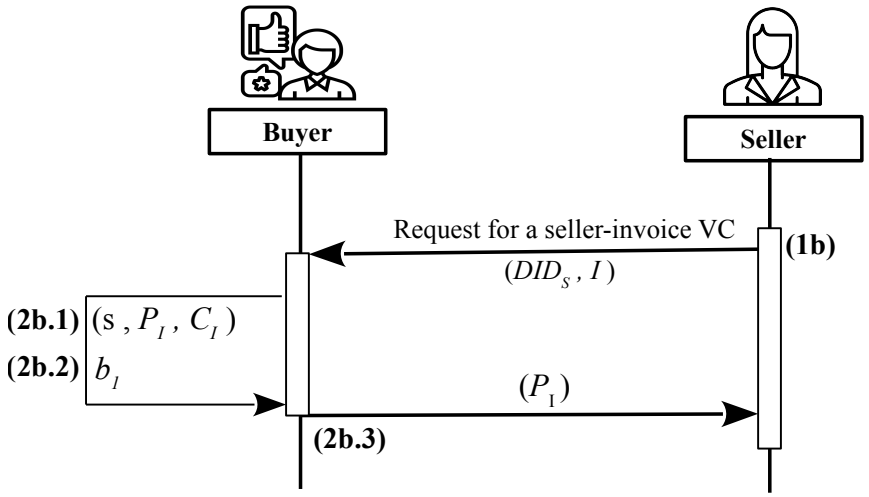

Figure 4.3: Registration of an invoice for a seller (seller-invoice VC).

The verification process of C_I is also the same as C_F. As factors, sellers can also have multiple DIDs for better anonymity and prevention of linking attacks. However, as with factors, a seller can only use one of its DIDs to receive the seller-invoice credential for a particular invoice. The buyer performs the following additional processing for seller-invoice registration:

2b.2 The buyer selects another random number $b_1 \in (1, n)$ and stores it for later use. In particular, b_1 will be used by the selected factor to derive an encryption key using a Diffie-Hellman (DH) key exchange scheme. As we explain in the next phase of the protocol, we use DH to establish a shared secret key between the buyer and the selected factor using the buyer's agent and on-chain information provided by the factor.

2b.3 The buyer replies to the seller with P_I. Having the value of P_I, the seller can compose the corresponding seller-invoice credential C_I (see Equation Eq. (4.3)).

4.2.2.2 Phase 2: Factoring

According to Figure 4.4, the steps followed in this phase are the following:

3.1 The factoring phase starts with the seller contacting multiple factors to negotiate and compare the different offers and conditions for the possible invoice factoring. The seller provides her invoice details, including the invoice number (I), the total amount of the invoice (a_I), and the payment due deadline (d_I), to the factor. The factor specifies his/her offered amount for the invoice ($a_F = a_I - fee$) and the deadline for completing the factoring registration (d_F). Then, according to the received offers, the seller selects the best factor to continue with. After this decision, the factor sends its credential to (C_F) to the seller, and the seller sends the seller-invoice credential (C_I) to the factor. Remember that the seller-invoice credential identifies both, the seller and the invoice that is going to be factored.

3.2 The factor extracts the seller-invoice credential identifier (P_I) from the credential received from the seller and sends it to the buyer agent using an end-to-end encrypted DIDComm channel. Then, the buyer answers with the associated salt s and the DH parameter g^{b_1}. Next, using Equation Eq. (4.4) with the salt and the DH parameter received, the factor can check whether the C_I provided by the seller is valid and accepted by the buyer or not. In the affirmative case, the protocol continues with the next step.

3.3 On the side of the seller, a similar operation as the previous one is carried out to check the credential of the factor (C_F). This step begins with the seller extracting the factor credential identifier (P_F) from the credential received from the factor and sending this identifier to the buyer agent using an end-to-end encrypted DIDComm channel. Then, the buyer answers with the associated salt s. Next, using Equation Eq. (4.1) with the salt received, the seller can check whether the C_F provided by the factor is valid and accepted by the buyer or not. In the affirmative case, the protocol continues with the next step.

4.1 When the verifiable credential presented by the selected factor is verified, the seller sends her bank account number ($IBAN_S$) to the factor using the associated end-to-end encrypted DIDComm channel.

4.2 The factor checks the smart contract at address $@C$ (as specified in C_I) to ensure that the invoice has not been already factored. In addition,

he/she subscribes to one or several nodes of the distributed ledger to be notified about any factoring agreement registered by the smart contract in the ledger.

4.3 The factor sends the agreement information ($\mathcal{A}$) and settlement information ($\mathcal{S}$) to the seller. All these data are digitally signed using the public key PU_F, which is resolved from DID_F (the signature is noted as $\sigma_{\mathcal{S}}^{DID_F}$):

$$F \rightarrow S: \quad \mathcal{A}, \mathcal{S}, \sigma_{\mathcal{S}}^{DID_F}, \tag{4.5}$$

$$\mathcal{A} = (C_I, C_F, a_F, IBAN_S), \tag{4.6}$$

$$\mathcal{S} = (P_I, d_F, \mathrm{Enc}(K_{FB}, C_F, IBAN_F), g^{b_2}, r, \mathrm{h}(r, \mathcal{A})), \tag{4.7}$$

where r is a random number (salt), $\mathcal{A}$ is actually a confirmation for the seller, and neither is given to the buyer nor stored on-chain. However, the salted hash of $\mathcal{A}$ is included in the signature ($\sigma_{\mathcal{S}}^{DID_F}$) as a commitment and for non-repudiation purposes. In contrast, $\mathcal{S}$ will be registered on-chain and a part of it is encrypted and hidden from the seller. Notice that the encrypted part is essentially the account number of the factor where the buyer has to pay in case the invoice has been factored. Clearly, this information is not necessary to be known by the seller. To create this symmetric encryption, the value g^{b_2} is provided on-chain by the factor to allow the buyer to reconstruct the shared key K_{FB} and decrypt that part. To do so, the factor selects a random number $b_2 \in (1, n)$ and uses the DH key-exchange formula to generate the symmetric encryption key K_{FB}:

$$K_{FB} = \mathrm{KDF}((g^{b_1})^{b_2}). \tag{4.8}$$

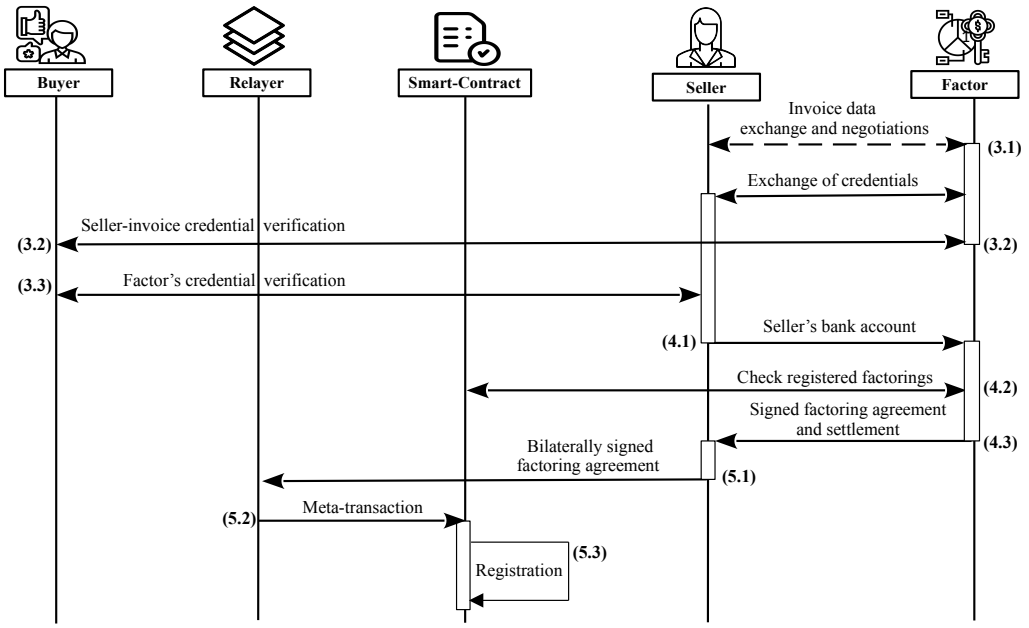

Figure 4.4: The factoring phase.

After an agreement is reached, the relevant factoring and payment details have to be registered in the distributed ledger. In general, each interaction that modifies the state of a public ledger requires a fee to be paid. In our protocol, one of our main design goals is to have the minimum possible number of transactions for completing a factoring registration in order to avoid paying excessive fees. Actually, we only need one transaction per invoice factoring, and the majority of the communications between the different parties are off-chain. In particular, our protocol is designed so that only the seller has to pay for invoice registration, while interactions with the distributed ledger by the factor and the buyer are view-only, as well as are free of charge.

Moreover, as mentioned in Section 5.2.1, managing cryptocurrencies to pay the fees may be difficult for institutions and companies. To overcome this problem, we use a relayer. In order to explain how the relayer works, we have to understand the purpose of the signature in a regular transaction. In this respect, the signature in a regular transaction has two different purposes. In the first place, it determines who pays the fee, and, in the second place, it is used to authenticate and authorize a user in front of a smart contract. Introducing a relayer in our protocol allows us to decouple the signature required for paying the transaction fee, which will be paid by the relayer, from the signatures required for authentication/authorization, which will be performed bilaterally by both, the seller and the factor.

An advantage of using a relayer is that it enables a clear and easy-to-implement business model for our protocol. The relayer can charge sellers per usage (e.g., per transaction request), and the sellers may have to buy some credit to use the relayer's API with any classical off-chain payment method, such as a credit card, bank transfer, etc.

5.1 The seller checks that the factor's signature over the settlement data ($\sigma_{\mathcal{S}}^{DID_F}$) is valid (Equation Eq. (5.2.5)), that the agreement data ($\mathcal{A}$) is what it has been negotiated with the factor, and that the hash value included in the settlement information ($\mathcal{S}$) is correct. The seller records the factor's signature for possible later use as digital evidence. Then, the seller creates a message for the bilateral settlement $\mathcal{M}$:

$$\mathcal{M} = (\mathcal{S}, \sigma_{\mathcal{S}}^{DID_F}). \tag{4.9}$$

Then, the seller signs $\mathcal{M}$ and sends the bilateral settlement message and its signature to the relayer:

$$S \rightarrow R: \quad (\mathcal{M}, \sigma_{\mathcal{M}}^{DID_S}). \tag{4.10}$$

Notice that we do not need to trust the relayer, and the seller does not share any confidential/private data with it. The signature $\sigma_{\mathcal{M}}^{DID_S}$ authenticates the seller and prevents the relayer from changing any detail of the bilateral agreement. Therefore, the relayer only plays the role of a facilitator and nothing more. As mentioned, the factor and the buyer do not need to interact with the relayer because they only need to query the smart contract. These queries are performed directly from blockchain nodes and are free-of-charge.

5.2 The relayer deducts the transaction fee plus probably some extra commission from the seller's credit, bundles the received information into a meta-transaction, and sends it to the smart contract:

$$R \rightarrow C: \quad (mtx, \sigma_{mtx}^{@R}), \tag{4.11}$$

$$mtx = (\mathcal{M}, \sigma_{\mathcal{M}}^{DID_S}). \tag{4.12}$$

Notice that, in fact, the settlement data ($\mathcal{S}$) is triply signed at this step: (i) by the relayer to pay its transaction fee, (ii) by the seller for registering the factoring agreement, and (iii) by the factor for promising to pay the invoice. The first signature is automatically verified by the blockchain platform, and the transaction fee is reduced from the balance of the relayer.

5.3 In the final step of this phase, the smart contract processes the meta-transaction (mtx) as follows:

- Determines the public keys used for signing $\mathcal{M}$ and $\mathcal{S}$, that is PU_S and PU_F, respectively. With these keys, it checks the signatures in the meta-transaction.
- Verifies that the current blockchain time is smaller than the registration deadline (d_F). This deadline is extracted from $\mathcal{S}$.
- If the invoice is already registered by the seller, the meta-transaction is rejected.
- If all the previous steps are correctly passed, the smart contract registers the factoring agreement by setting a flag in its key-value storage and storing the public keys and settlement information in a log.

In most distributed ledgers, logs are on-chain data produced by the transaction execution, but the log's contents are not recorded in the ledger's global state. This makes logs much cheaper than storing data in the smart contract storage and also makes them convenient if their content is not needed by successive transactions (the case in our registration protocol). More details about the registration by the smart contract are given next.

We efficiently store the data by using only one boolean flag per invoice in the key-value storage of the smart contract:

$$P_I \Rightarrow \texttt{true}. \tag{4.13}$$

Obviously, the mapping can only be set if it was not already set to another value and is sufficient for the correct functionality of the smart contract and prevention of double-factoring. The complete settlement information ($\mathcal{S}$) is not

required anymore to be accessible by the smart contract code, so we can store it in a log to get the immutability of blockchain:

$$\mathtt{log}(P_I, \mathcal{S}, PU_S, PU_F). \tag{4.14}$$

P_I is defined as an index field for quick search. We use P_I again as the key of the log, which is known by all involved parties. The buyer can use it as the key to finding out whether the invoice is factored or not, and get access to the logged information. The contract cannot link the identities of the seller and the factor with the transaction. Therefore, the public keys are recorded for later verification by the buyer. The salted hash (fingerprint) of the factoring agreement $\mathsf{h}(r, \mathcal{A})$ is also included in $\mathcal{S}$, which can be used as a proof-of-existence by the seller or the factor in case of dispute. Finally, the seller and the factor can be subscribed to the smart contract, get automatically informed about the registration, and verify the contract log to ensure everything is recorded in line with their agreement.

4.2.2.3 Phase 3: Payment

In the third and last phase, the factor pays the seller after checking the information registered by the smart contract. Later, the buyer will pay the complete invoice amount to the factor also, after checking the information registered by the smart contract. The steps followed in this phase are described next (see Figure 4.5):

6. The factor proceeds to pay $a_F = a_I - fee$ to the account of the seller. This has to happen before the agreed payment deadline.

7.1 When the deadline of an invoice (d_I) expires, the buyer queries the smart contract to figure out whether the invoice has been factored or not. The buyer knows the address of the smart contract ($@C$) and the pseudo-anonymous identifier of the invoice (P_I). Using these two values, the buyer queries a node of the distributed ledger to obtain the log with the index field P_I from the smart contract. From this on-chain log, the buyer obtains $\mathsf{Enc}(K_{FB}, C_F, IBAN_F)$ and g^{b_2}.

7.2 The buyer computes K_{FB} and decrypts the encrypted part of the logged information using the obtained g^{b_2} and the value b_1 that it stored in step (2b.2) for this invoice. Then, the buyer:

- Verifies that the value PU_F which is recorded by the smart contract, belongs to C_F. Otherwise, the factor has cheated because the seller has verified everything other than this encrypted part. In this case, the buyer cannot pay the factor because of the KYC regulation.
- Obtains the seller-invoice credential related to P_I from its database and matches its public key with the recorded PU_S. If the public key does not match, obviously both the seller and the factor are cheating because the invoice does not belong to this seller. In this case, the buyer will pay the invoice amount to the authentic seller.

If the verifications in the last step are passed, the buyer knows the correct selected factor and his/her associated *IBAN*and pays the invoice amount (a_I) to that *IBAN*.

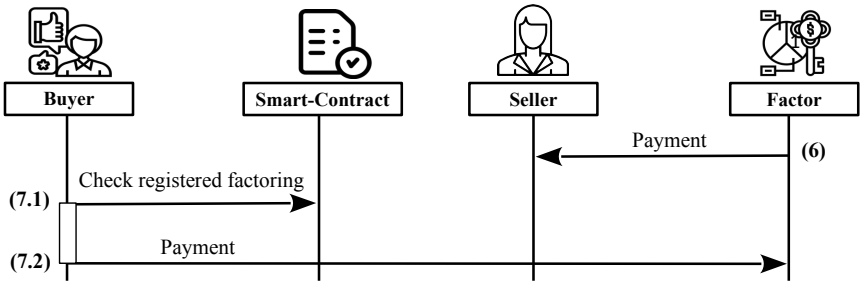

Figure 4.5: Payment phase.

4.2.3 Use Case

This section describes a typical scenario for better understanding the process in which the system in this chapter is used to register a factoring agreement related to monetizing geolocation data. In our scenario, the buyer is the municipality of a city buying anonymized geolocation data from an online navigation platform (the seller). The municipality uses the data for better provisioning of city growth and land development in areas, such as public transport system, highway routing, housing and zoning, placement of new public services and utilities, and other land use plannings. In order to simplify things and avoid physical

authentication complexities, we assume all parties are registered legal entities, and the government has issued them verifiable credentials. The credentials specify their tax identification number which can be used to identify them (real identity) in our protocols.

The municipality buys a lot of products and services from private companies, and many of them make factoring agreements with financial institutions. In order to reduce errors and prevent double-factoring, the municipality has accredited an implementation of our architecture, and the sellers have to register their factoring agreements in this system. The implementation provides a special agent to the buyer for performing the required operations. In addition, the municipality owns a government-issued VC and DID which is in the control of the agent.

The municipality invites different providers of online navigation systems to a tender, and after receiving their offers, selects the best candidate. During post-tender negotiations, the seller company agrees to receive its money 90 days after supplying the data. However, the company lacks enough working capital to continue its service properly, and decides to factor the invoice. The seller connects to the buyer's agent over DIDComm, authenticates itself using the government-issued VC, and proceeds until it receives a seller-invoice VC for the aforementioned invoice.

Then, the seller contacts multiple factors, negotiates with them, compares their offers and conditions, and, finally, selects one of them. The factor connects to the municipality's agent over DIDComm, and registers a factor VC. After that, the seller and the factor establish a DIDComm channel and exchange and verify their seller-invoice and factor VCs. Note that their tax identification numbers are also evident from their VCs, and they register tax IDs for their work. The seller provides the factor with its bank account number, and the factor digitally signs the factoring agreement and sends the signed settlement information to the seller. The seller connects to a relayer and gives the bilaterally signed agreement to it, which subsequently hands all this information to the system's smart contract. The smart contract verifies the request and registers the agreement in the public distributed ledger (blockchain). In this stage, the factor pays the seller a sub-total of the invoice amount according to their agreement. After the 90-day period is passed, the municipality's agent checks the smart contract and finds out that the invoice has been factored. It decrypts the encrypted

part of the logged information, verifies that everything is correct, and pays the complete invoice amount to the bank account of the factor.

4.3 Evaluation

In this section, we provide a security analysis of the proposal in this chapter and a comparison with the related works.

4.3.1 Security Analysis

In the following security analysis, we analyze the security of verifiable credentials, data security and privacy, availability, and dispute handling. For each of these aspects, we explain security requirement(s), possible attack(s), and our mitigation method(s).

4.3.1.1 Verifiable Credentials

The contents of the seller-invoice and factor credentials are not published or disclosed by the buyer; their owners may hand them to other parties at will. Their identifiers are generated by a cryptographic hash function (such as scrypt), which is:

- One-way: the content of a credential cannot be recovered from its identifier.
- Pre-image resistant: a fake credential cannot be matched to a valid identifier.
- Integrity guarantee: if the credential is manipulated, the identifier does not match (avalanche effect).
- Resistant to guessing attacks: guessing attacks are infeasible because a fresh salt is used (blocks offline attacks), and its computation is resource-intensive (blocks online attacks).

For a verifier to verify a credential, the identifier is queried over an authenticated DIDComm channel. Therefore, as long as the hash and DIDComm are secure, the credentials are secure, as well.

4.3.1.2 Data Security and Privacy

We use a smart contract to process and store critical factoring agreement details on the blockchain. The blockchain is designed to guarantee the precise execution of the contract and protect stored data from manipulations. All this data is publicly readable; therefore, confidentiality and privacy are more of a concern in comparison to traditional systems.

We do not store any information that can be used to identify or trace the seller or the factor on the blockchain. No personally identifiable information (PII), even the invoice number, is transmitted over the network in plain text or publicly stored on the blockchain. The signatures of the seller and the factor are verified by the smart contract and their public keys are recorded to prevent fraud. However, these keys do not contain any information that can be traced back to the real identity of their owner. In addition, sellers and factors are free to use new public keys for each invoice factoring, which protects them against curious observers who try to link transactions together and trace individuals.

Only the seller and the factor store the agreement information (the amount that the factor pays to the seller, and the bank account number of the seller), and this information is hidden from the buyer. In addition, we privately hand over the identity and bank account number of the factor to the buyer. To be more precise, the factor encrypts this information by a symmetric key which only the buyer can know. Therefore, no one (including the seller) can have access to this information.

The buyer in our architecture is not involved in the negotiations between the seller and the factors. In particular, the buyer cannot predict if the seller will factor her invoice or not (before the agreement about payment conditions and other invoice details). Moreover, our architecture protects the factors from each other as they do not have access to their competitors' conditions (before and after finalizing the factoring contract). Factors are not notified if the seller applies to multiple factors to obtain a better bid for the invoice.

4.3.1.3 Availability

Public distributed ledgers are maintained by many geographically distributed nodes over the Internet. For example, at the time of this writing, the Ethereum network is operated by more than three and half a thousand nodes [34]. Therefore, our on-chain registered data is highly available for both the factors to protect them from double-factoring, and the buyer to discover the correct payment information.

One may be concerned about the availability of the relayer. The first and the most important thing is that the safety and correctness of our protocol is not endangered when the relayer becomes unavailable. However, the protocol may not proceed, and the seller may lose her time to conclude the factoring agreement. We used DIDComm for our communications which is more resistant to temporary network disconnections. Nonetheless, if the relayer is unavailable, the seller may switch to a distributed network of third-party relayers, e.g. the Gas Station Network (GSN) [35], to fulfill its duties. As we do not trust the relayer, this will not pose any threats to the security of our protocol.

4.3.1.4 Dispute Handling

We must remark that our registration protocol is not secure against malicious buyers because, if the buyer registers false information, this cannot be disputed by the seller nor the factor. As a result, the seller and the factor need to trust the buyer. A malicious buyer, for example, may not pay the seller or the factor. A malicious buyer may also scam factors by creating a fake seller and a high amount of non-existent invoices. Then, the fake seller receives the payments from the factors, but the corresponding payments are not made by the malicious buyer. In case the buyer is not trustful, some mechanism to enforce good behavior must be used (like a reputation system as in Guerar et al. [8]).

On the one hand, a seller may be concerned about a malicious factor that may refrain from payment. In this case, the seller reveals the agreement information to a judge, and the judge can doom the factor according to the tamper-proof evidence stored by the smart contract on the blockchain. On the other hand, a factor may also be concerned about the case in which the buyer pays the

amount to another bank account. In this case, the factor reveals the agreement information, as well as its private DH value (b_2), to a judge. The judge can also detect the culprit in this case by referring to the bank account logs and the blockchain evidence. Note that a fraudulent seller/factor cannot conclude an invoice factoring agreement because, before paying anything to the factor, the buyer verifies the stored information on the blockchain against his registered information.

4.3.2 Comparison to Related Work

Below, we compare the protocol intoduced in this chapter with the related works. The related works are reviewed in the previous chapter, and we refere the reader to Chapter 3 for more complete description of the works. However, as we aim at additional design goals, we focus on the related aspects of each of the works, and give a brief review of the works with respect to the specific aspect. A summary of the comparison with the related works is shown in Table 4.1.

Table 4.1: Comparison with the related works.

	[5]	[1]	[8]	Our Chapter 3	This Chapter
Type of ledger	private	private	public	public	public
# Transactions	1	5	7	1	1
Users do not need cryptocurrency	true	true	false	false	true
Enables SSI	false	false	false	false	true

The first aspect to take into account is the type of the ledger used to register the invoice factoring. DecReg [5] and Battaiola et al. [1] propose to use a private ledger, while Guerar et al. [8], our Chapter 3, this chapter's protocol use a public ledger. The type of ledger is one of the most relevant decisions to take when designing an invoice factoring solution since it directly impacts in aspects, such as privacy, immutability, transparency, availability, the need to optimize the number of transactions, and the need that users manage cryptocurrency to do the registrations.

Private or permissioned ledgers are blockchain networks that restrict their access to registered users. This allows implementing some degree of privacy by grating visibility to the set of transactions only to designated participants. This is the approach followed by DecReg [5], in which transactions are not publicly released, and the system relies on a Central Authority (CA) to control the access to the system. Nevertheless, since the CA acts as a point of centralization,

it can become a single point of failure, making it vulnerable to double factoring attacks. In fact, if the CA is compromised, it can deny a factor from accessing the network, and make it impossible for the factor to determine if an invoice has already been factored. In this framework, data is not encrypted, and the CA limits access to confidential data to individuals outside the private blockchain network, which does not fully protect the privacy of the network's participants. In addition, the buyer is required to operate a node on the private blockchain and obtain credentials from the CA, which heavily involves the buyer in the factoring process. On the other hand, if a dispute arises between a seller and a factor, the only proof that can be used under DecReg is the signatures on transactions. The main problem is that the system relies on the CA for managing access to the system, and the transactions are not publicly available.

Battaiola et al. [1] also propose to use a private ledger for registering invoice factoring. However, instead of sending transaction data in clear, they use commitments to hide data and provide privacy. Although the authors claim that any other ledger can be used in place of the private ledger without compromising protection, this replacement is not cost-effective due to the number of transactions (five) required to complete a factoring registration. In particular, the proposal needs one transaction from the seller to do the invoice registration, another from the buyer to approve this registration, another from the seller to register the factoring proposal, another from the factor to accept the factoring proposal, and finally, another for registering the payment (or the factoring expiration). In addition, the availability and immutability of data are dependent on the security provided by the private ledger. A more secure private network includes more nodes and organizations, resulting in a higher operating cost. In particular, in [1], it is not specified who is responsible for the cost of operating the private ledger.

As shown in Table 4.1, Guerar et al. [8], our Chapter 3, and the proposed protocol in this chapter use a public ledger. Public ledgers provide the best immutability, transparency and availability. However, when designing a solution over a public ledger, special care must be taken with privacy, cost, and cryptocurrency management. Regarding privacy, these three proposals make use of commitments to guarantee that sensitive data is not disclosed in the public network.

Regarding cost, the solution proposed in [8] conducts the invoice factoring negotiation on-chain which makes it rather expensive, requiring seven transactions to complete an invoice factoring. Concretely, the buyer, who is generally not that much motivated for the factoring process, must perform three transactions in the public ledger per invoice factoring: one transaction to accept the invoice and pay the shipping, another transaction to confirm the delivery of the products, and a final transaction to pay the entire amount of the invoice to the corresponding factor. Another critical issue of the proposal by Guerar et al. is that, since the platform is in charge of creating stable accounts for reputation, if the platform does not correctly certify the real identities of buyers, the system's security is jeopardized. Finally, it is worth it to mention that the proposal uses IPFS. Although this peer to peer distributed file system can provide a reasonable level of availability, it is not as high as that provided by on-chain data, which is critical if the buyer needs to access payment data with no downtime.

Our protocol in Chapter 3 is based on a public ledger that is cost-efficient, just requiring one transaction in the ledger to register an invoice factoring. Appropriately, the single transaction for the invoice factoring registration is performed by the seller, who is the most interested party. However, in that protocol, the seller is forced to use cryptocurrency to execute this transaction, which might be an obstacle against the adoption of the protocol.

In the protocol proposed in this chapter, we include a relayer in the architecture to overcome the problem of users having to manage cryptocurrency. The relayer allows us to free users from having to use cryptocurrency while keeping the properties of decentralized applications. Furthermore, as shown in Table 4.1, this protocol is not only optimal in terms of cost, and built over the most reliable, transparent, and secure type of ledger; it also enables new functionality not available in any other related protocol. Specifically, the protocol allows parties to implement their self-sovereign identities making use of their self-managed identifiers.

The protocol also leverages the concept of Verifiable Credentials (VCs), which are credentials issued to self-sovereign identities and grant permission to the parties to participate in our invoice factoring architecture. Another advantage of using DIDs is that we can relay on the new communications models that are being developed in this ecosystem, like DIDComm. DIDComm allows us to

implement asynchronous and secure off-chain communications between participants, which means that a party does not need to be present at the moment that another party sends a message. The response can be received, processed, and approved asynchronously.

4.4 Implementation

In this section we describe a possible proof-of-concept implementation for our architecture. For this proof-of-concept, we created two components: a front-end and a registration smart contract. Next, we mainly focus on the description of the registration smart contract, which is the core component of our proposal.

4.4.1 Tools and Setup

The registration smart contract is written in Solidity, which is the Ethereum's most popular programming language for writing smart contracts. We also used the Truffle[1] Development Framework, which helped us to create, compile, develop and test smart contracts. The Truffle framework can used to develop distributed applications, in particular, using a personal blockchain (Ganache) and a front-end DApp development kit (Drizzle).

Apart from Truffle, we also used Infura. Infura[2] provides us with a gateway to different Ethereum networks, so that we do not need to run an Ethereum node by ourselves. Recall that every change we make in Ethereum has to be done with a transaction; whether it is transferring Ether, deploying a smart contract, or calling a function of an already deployed smart contract. A private key must be used to sign every transaction, so our app needs the ability to sign transactions to be able to make state changes in Ethereum. To send these signed transactions to the Infura nodes, we used the Truffle HD wallet provider.

Finally, to make more real tests of our implementation, we deployed our smart contract to Rinkeby. Rinkeby is a public Ethereum test network where we can deploy our code in an environment similar to the main Ethereum network. For this, we created a test account and used one of the Rinkeby's faucets to receive

[1] https://trufflesuite.com
[2] https://infura.io

test Ether. This Ether allows us to pay the cost of each transaction. We also configured Truffle to use the Infura's nodes for interacting with the Rinkeby test network. All the secrets where stored in a configuration file called .env that is not shared or versioned.

4.4.2 The Smart Contract

The complete code of the registration smart contract is available at appendix A. The main function of the smart contract is the invoiceRegistration function. The function examines the following steps to verify its inputs and complete the registration transaction:

- First, we check the timestamp to ensure that the contract execution time is less than the due time.
- Then we verify the given signatures. In order to be able to check a signature, it is necessary to have its corresponding message and the signer's public key. The point is that instead of giving the public key to the contract manually, it is recovered from the signature itself. In particular, ECDSA signatures have a property that the public key of the person who signed the message is recognizable. We have a recoverSigner function to extract the signer's public key from his/her signature. To recap this step, we compose the signed messages S and M from the provided input to the contract, then calculate the public keys of the seller and the factor (PU_S and PU_F) from the signatures (σ^{DID_F}, σ^{DID_S}).
- Then, the identifier of the invoice (P_I) is checked so that it is not registered before. If P_I is being re-registered, the execution of the function invoiceRegistration will fail, reverting the transaction.
- The registration is finalized by setting to true the corresponding identifier in the alreadyRegistered mapping, and emitting an event NewInvoiceRegistered.

All the above steps are implemented in the invoiceRegistration function which is called by the relayer (Listing 4.1.)

```
1  function invoiceRegistration(bytes memory p_I,uint256 due_time,string memory
     enc_factor_bank,string memory public_bank,uint256 r,uint256 h,bytes memory
     sign_DID_F,bytes memory sign_DID_S) public {
2
3      address p_u_s;
4
5      address p_u_f;
6
7      require(block.timestamp < due_time, "Meta transaction is expired");
8
9      bytes32 M = keccak256( abi.encodePacked(
10     p_I,due_time,enc_factor_bank,public_bank,r,h,sign_DID_F));
11
12     bytes32 S = keccak256(
13     abi.encodePacked(p_I, due_time, enc_factor_bank,public_bank,r, h)
14     );
15
16     p_u_s = recoverSigner(M, sign_DID_S);
17     p_u_f = recoverSigner(S, sign_DID_F);
18
19     require(!isInvoiceRegistered(p_I), "The invoice is already registered.");
20
21     alreadyRegistered[p_I] = true;
22
23     emit NewInvoiceRegistered(
24     p_I,due_time,enc_factor_bank,public_bank,r,h,p_u_s,p_u_f
25     );
26     }
27
```

Listing 4.1: invoiceRegistration function to factor an invoice

The isInvoiceRegistered function simply returns true if the identifier of an invoice (P_I) is registered (Listing 4.2).

```
1  function isInvoiceRegistered(bytes memory p_I) public view returns (bool) {
2    return AlreadyRegistered[p_I];
3  }
```

Listing 4.2: isInvoiceRegistered function to check if an invoice is already registered

4.4.3 Functions for Testing

In the above steps, the digital signatures on M and S have to be checked. In order to test our code, we have written utility test functions in our smart contract to calculate the digital signatures (σ^{DID_F} and σ^{DID_S}).

The function get_S calculates the value of S from the corresponding input data (Listing 4.3.)

```
1  function get_S(
2      bytes memory p_I,
3      uint256 due_time,
4      string memory enc_factor_bank,
5      string memory public_bank,
6      uint256 r,
7      uint256 h)
8      public pure returns (bytes32)
9    {
10     bytes32 S = keccak256(abi.encodePacked(p_I, due_time, enc_factor_bank,
       public_bank, r, h));
11     return (S);
12   }
```

Listing 4.3: Calculate S (on-chain)

With S, we can compute σ^{DID_F} off-chain. For example, we can do so using the following Javascript code (Listing 4.4).

```
1  const EthUtil = require('ethereumjs-util');
2  const web3 = require('web3');
3
4  const messageToSign =
       "0x607766330da20b09790047553de552dc475b0760fd9e2ba19af598a9a56ee019";
5  const privateKey =
       "dcb0075613ce655885815afc168dbb0b188addb5bba9fb25e3a744db266c5b43";
6
7    var msgHash = EthUtil.hashPersonalMessage(Buffer.from(messageToSign));
8    var signature = EthUtil.ecsign(msgHash, Buffer.from(privateKey, 'hex'));
9    var signatureRPC = EthUtil.toRpcSig(signature.v, signature.r, signature.s)
10   var arr = new Uint8Array(msgHash);
11   var hash = web3.utils.bytesToHex(arr);
12   console.log("Hash: ", hash.toString(), " sign: ", signatureRPC);
```

Listing 4.4: Calculate σ^{DID_F} (off-chain)

Note that messageToSign and privateKey are sample test values. Finally, we can provide σ^{DID_F} to the function (get_M) of the smart contract, which returns the M value (Listing 4.5).

```
function get_M(
    bytes memory p_I,
    uint256 due_time,
    string memory enc_factor_bank,
    string memory public_bank,
    uint256 r,
    uint256 h,
    bytes memory sign_DID_F)
    public view returns (bytes32 M)
  {
    require(
    block.timestamp < due_time,
    "META_TX: Meta transaction is expired"
    );

    return
    keccak256(
    abi.encodePacked(p_I,due_time,enc_factor_bank,public_bank,r,h,sign_DID_F)
    );
  }
```

Listing 4.5: Calculate M (on-chain)

4.5 Conclusions

In this chapter, we presented an improved protocol with respect to the one we proposed in Chapter 3 that uses a public distributed ledger to register invoice factorings. We added several enhancements to the protocol and simplified it to increase efficiency and flexibility, as well as to facilitate user on-boarding. We used Decentralized IDentifiers (DIDs) and let the involved parties use their self-sovereign identities (SSIs). One advantage of using DIDs is that we can relay on the new protocols being developed in this ecosystem, like DIDComm, which allows us to implement asynchronous secure communications between participants. When using DIDs, we can also leverage on the concept of Verifiable

Credentials (VCs) to grant permission to parties allowed to participate in our invoice factoring ecosystem.

In this ecosystem, the buyer, who is considered to be the trusted party for the factoring process, issued Verifiable Credentials to DIDs of sellers and factors. The proposed protocol maintained its efficiency from the base protocol, and used only one meta-transaction per factoring registry. The seller used to pay the cost of executing the meta-transaction in the public distributed ledger since it was the party with the highest interest in the service. To provide sellers with an easy on-boarding, a relayer was introduced in our invoice factoring ecosystem. The advantage of using the relayer is that sellers can have the high security and availability levels provided by public distributed ledgers without having to deal with cryptocurrency. This is because sellers can pay to the relayer with off-chain methods, like credit card or bank transfer, but the relayer cannot alter any aspect of the invoice factoring agreement being recorded. As a result, the proposed architecture provided an efficient and friendly protocol for registering factored invoices.

In the next chapter, we present our final architecture and protocol which adds zero knowledge proofs (ZKPs) to the protocol. By adding the ZKPs, the smart contract can better verify the identity of the involved parties in one hand, and the autenticity of the provided data on the other hand. In this case, most of the checks that we required the buyer to perform, are handled by the smart contract and a special denial-of-service attack is prevented.

Chapter 5

Enhanced Privacy with Zero-Knowledge Proofs

5.1 Introduction

The protocol presented in this chapter is an enhancement regarding our previous protocols [12] and [13]. In particular, we enhance the privacy design, by only allowing the intended parties to see their relevant data. We also fix a possible denial of service (DoS) attack in which the identifier of the invoice was set in the smart contract with dummy data. This could create a DoS attack because the corresponding invoice cannot be registered. Notice that the invoice identifier is not available until a registration transaction is sent to the blockchain network, but the attack, in principle, can succeed by doing a front running (sending a transaction with more fee). In this version, we fix this issue with a zk-SNARK proof. The zk-SNARK proof guaranties that the registration comes from the authentic parties but without affecting privacy, that is to say, without disclosing any identity or any other private data.

The rest of the chapter is organized as follows: section proposed architecture 5.2 introduces the third protocol. Afterwards, we review the related works 5.3 and compare them with our proposal,and chapter is concluded in the section conclusion 5.4.

5.2 Proposed Architecture

In our architecture, we have the three classical entities of the factoring scenario: the buyer, the seller, and the factor. Additionally, we have a smart contract deployed on a public distributed ledger. Our protocol is related to a factoring process on a pre-arranged agreement between a seller and a buyer. Once the agreement is final, the buyer is required to issue a special invoice certificate about the details of the agreement. At this time, the seller may decide to factor-out the invoice using our protocol. In the following sections, we will first state our goals and assumptions. Then, we will explain the format of the invoice certificate, and subsequently detail our protocol.

At a high level, our protocol works as follows:

- The seller negotiates with several factoring companies and chooses a desired factor.
- The factor sends a signed factoring agreement to the seller. In this system, before entering the factoring phase, the factor is required to register with an accreditation agency and receive a compliance certificate ($\mathcal{C}_F$). The compliance certificate is a certificate that adds information to the digital certificate of a factor, in particular, it states that the identified entity can execute financial transactions.
- The seller adds other information to the agreement, signs the everything, and sends it to the smart contract. The seller includes a non-interactive zero-knowledge proof in his/her transaction to the smart contract to prove the possession of the invoice certificate.
- The smart contract verifies the credibility of the involved parties as well as the public invoice details which, among other things, confirm the identity of the seller. If everything is okay, and the invoice has not been factored before, the smart contract registers the agreement to prevent double-factoring.
- The factor queries the smart contract to ensure that the agreement is final, and pays the agreed amount to the seller.
- When the invoice payment deadline is reached, the buyer checks the smart contract and notices that the invoice is factored.

- Finally, the buyer pays the invoice amount to the factor.

The main advantage of the protocol in comparison to the existing protocols are the following:

- Supports multiple buyers.
- The level of detail checked by the smart contract greatly reduces the possibility that the buyer refuses to pay in the final step.
- The zk-SNARK proof prevents a fraudulent seller from registering a fake invoice.

All the above improvements are achieved without compromising privacy.

5.2.1 Design Goals & Assumptions

In our design, a smart contract manages all interactions required to finish a factoring registry. Payments are made off-chain via fiat transfers between bank accounts. We assume that all parties trust the correct execution of transactions controlled by the smart contract. Generally, the smart contract must be reviewed by all parties concerned and these parties must agree to use it as the reference for their factoring purposes. The buyer must pay the invoice to the bank account of the smart contract's registered entity if the invoice has been factored.

In our previous works, we relied on hash functions and symmetric encryption to make the factoring process is as less complex and resource-consuming as possible for the buyer. The buyer was unique so it was possible to do the design with these simple cryptographic primivies. However, how we can have multiple buyers, so we need a mechanism to identify buyers. This is the reason why we use public key cryptography and digital certificates.

In the factoring process, sensitive business information must be adequately protected, especially considering that we use a public ledger in which everybody has access to data sent in transactions. For this reason, any sensitive data that must go on-chain is symmetrically encrypted. This encryption uses different keys depending on whether the data should be accessible by the seller and the

factor or by the buyer and the seller. We use zero knowledge to guarantee that registrations are correctly done by an authorized seller. With this protocol, our invoice factoring process:

- Prevents double factoring without threatening privacy.
- There is no way to challenge the smart contract once it has registered an invoice factoring.
- Allows the appropriate parties to have access to their relevant data as well as provides evidences of the existence of these data.

In general, an invoice has the following information: the seller and the buyer identities, invoice number, issuance date, due payment deadline, total amount (and currency code), and other details about the service/goods provided by the seller to the buyer. We consider that the seller's identity and the invoice number are sufficient to identify the invoice uniquely; consequently, using unique invoice numbers should be mandated. In addition, the buyer's identification, due payment deadline, and total amount are required for factoring agreements. Other details can be added to the invoice without impacting how our system functions.

5.2.2 The Protocol

Our protocol is an improved extension to our previous protocols [12] with non-interactive zero-knowledge proofs. The extensions enable the smart contract to throughly verify factoring agreements before registering them, without compromizing the privacy of the involved parties. In addition, we have stripped some details from the protocol, and the factoring process to better focus on our contributions. The protocol is composed of four phases and the notation used to describe it can be found in Table ??.

5.2.3 The Protocol Setup

All the involved parties (buyers, sellers, factors) in our system shall have long-term asymmetric encryption and digital signature key-pairs. We assume they

have registered their public keys (denoted by PU_X) with a trusted registration authority (e.g., a government authority) and have a corresponding digital certificate (denoted by ID_X). The certificate links a real entity to a long-term public key. Given of a public key PU_X, any entity in possession of the corresponding private key is assumed to be the entity X. From now on, without loss of generality, we solely rely on these public keys to identify the parties.

As we use zk-SNARKs in our protocol, a trusted setup procedure is also followed in this phase to select the public parameters, and distribute the ZK proving and verification keys. In addition, a smart contract is deployed on a public blockchain to assist in the verification of zero-knowledge proofs. A registration smart contract is in the core of our protocol and is also deployed on the blockchain in this phase.

5.2.4 The Registration Phase

In this phase, two types of certificates should be issued which are used in the subsequent phase (factoring): (1) the buyer issues an invoice certificate (denoted by $\mathcal{C}_I$) to the seller, and (2) a trusted authority issues a compliance certificate (denoted by $\mathcal{C}_F$) to the factor. The certificates are explained in the following sections.

5.2.4.1 The Invoice Certificate

We assume that the seller always receives an electronic invoice certificate ($\mathcal{C}_I$) from the buyer, even if she is not going to factor it. We also assume that the buyer will not issue multiple invoice certificates for the same invoice. The invoice certificate is a digital certificate that includes all the details of an invoice, and that it is signed by the corresponding buyer. For privacy purposes, not all involved parties in a factoring scenario should have access to all details of the invoice/certificate. In particular, the buyer and the seller have complete access to the details, while the factor should only know the financial details, and the smart contract only needs to know the due deadline of the invoice and the identity of the buyer (we will elaborate on the details needed by the smart contract when we explain the factoring process.)

One possible solution is to split the invoice details into three fragments corresponding to the mentioned three levels of access, and ask the buyer to sign each fragment, separately. However, this solution is not efficient, as we need to (1) include redundant information into the fragments to link them together, and (2) one should verify multiple signatures in case he/she has access to multiple fragments.

Instead, we build a two-level Merkle hash tree in which the buyer only signs the root of the tree (top-level hash). In particular, the invoice certificate is composed of three fragments, as well as a signature over the root:

$$\mathcal{C}_I = [\alpha_I, \beta_I, \gamma_I, \sigma^B_{\theta_I}],$$

where

$$\begin{aligned}
\alpha_I &= [P_I, PU_B, deadline_I], \\
\beta_I &= [N_I, PU_S, date_I, amount_I], \\
\gamma_I &= [\text{other business-related invoice details}], \\
\theta_I &= [h(\alpha_I), h(\beta_I), h(\gamma_I)].
\end{aligned}$$

The first fragment is α_I, which is publicly accessible. This fragment is composed of the pseudo-identifier of the invoice, the public key of its buyer, and its payment deadline, respectively. The pseudo-identifier (P_I) uniquely identifies the invoice and is checked to prevent double-factoring.

The second fragment of the invoice certificate is β_I which is composed of the invoice number, the public key of its seller, and its issue date and amount, respectively. The β_I fragment is accessible to the buyer and the seller as well as the factor.

Finally, the most private invoice details are included in the third fragment, γ_I, which are the business-related information such as the list of the provided goods or services, and their fees. The γ_I fragment is only known by the buyer and the seller.

The hashes of the individual fragments are concatenated and signed by the buyer collectively ($\sigma^B_{\theta_I}$). One can verify the signature with the corresponding Merkle proof, that is to say, by considering the hashes of his/her inaccessible

fragments and calculating the hashes of the accessible fragment(s). Therefore, the certificate is verifiable even if some of its fragments are not disclosed to the verifier.

5.2.4.2 The Compliance Certificate

Because of know-your-customer (KYC) and other legal compliance issues, the involved parties shall authenticate and know each other diligently. We can assume that the seller and the buyer in one hand, and the seller and the factor on the other hand know each other before interacting with our system. However, the buyer and the factor, in general, do not have to know each other before the payment deadline of the invoice. Therefore, we would like to comply with the KYC regulation in this relation, and require the factor to register with an accreditation agency, and receive a compliance certificate ($\mathcal{C}_F$) before entering the next (factoring) phase.

The compliance certificate is a special attribute certificate which adds information to the digital certificate of a factor. In particular, it asserts that the factor identified by a public key is a legal entity that can perform financial transactions. For simplicity and in order to eliminate the necessity of a revocation mechanism, the certificate is short-lived and limited to the duration of a factoring agreement. The compliance certificate has the following details:

$$\mathcal{C}_F = [\alpha_F, \beta_F, \sigma^A_{\theta_F}],$$

where

$$\begin{aligned}
\alpha_F &= [PU_A, date_F, expiry_F],\\
\beta_F &= [SN, PU_F],\\
\theta_F &= [h(\alpha_F), h(\beta_F)].
\end{aligned}$$

in which the content of the certificate is its serial number, the long-term public key of the factor, the public key of the accreditation agency, and the issuance and expiry dates, respectively. All the content are signed by the agency ($\sigma^A_{\theta_F}$).

5.2.5 The Factoring Phase

In this phase, the seller and the factor register their agreement around the details of factoring an invoice, which is denoted by I. The registration is performed in the following steps:

1. The seller sends the following message to the factor over a secure channel:

$$S \rightarrow F : [\alpha_I, \beta_I, h(\gamma_I), \sigma^B_{\theta_I}, IBAN_S].$$

   The message begins with disclosing relevant fragments of the invoice certificate (α_I and β_I) the hash of the other fragment (γ_I), and the signature of the buyer on the certificate as well as the bank account number of the seller ($IBAN_S$).

2. The factor confirms the factoring agreement, and provides other information to the seller for sending to the registration smart contract. In particular, the factor:

   2.1. Uses the public key of the buyer PU_B, obtained from α_I, to verify the signature $\sigma^B_{\theta_I}$ over the certificate $\mathcal{C}_I$.

   2.2. Looks up the pseudo-identifier of the invoice (P_I) from the smart contract to ensure that the invoice has not been already factored.

   2.3. Generates a compliance proof (denoted by π_F) which is a zk-SNARK which (1) proves that he is in possession of a compliance certificate, and (2) links his long-term public key with one of his blockchain addresses.

   2.4. Composes the agreement and settlement information:

$$\begin{aligned}
&\mathcal{A} = [\alpha_I, \beta_I, h(\gamma_I), \sigma^B_{\theta_I}, \mathcal{C}_F, ID_F, a_F, IBAN_S],\\
&com_F = h(r_2, \beta_F),\\
&\mathcal{M} = [P_I, d_F, \alpha_F, com_F,\\
&\qquad \mathrm{Enc}(K_{FB}, \mathcal{C}_F, ID_F, r_2, IBAN_F),\\
&\qquad \mathrm{Enc}(PU_B, K_{FB}), h(r_1, \mathcal{A})].
\end{aligned}$$

   The agreement information ($\mathcal{A}$) is the factor's commitment to the mutual agreement which is neither given to the buyer nor stored

on-chain. We included α_I and β_I in $\mathcal{A}$ and only a hash of γ_I for privacy reasons. By the way, the signature ($\sigma^B_{\theta_I}$) is verifiable by this information. com_F is another commitment which privately links PU_F to π_F. The amount mentioned in $\mathcal{A}$ will be payed by the factor to the seller after reducing some fee from the amount of the invoice ($a_F = a_I - fee$). The d_F is the deadline before which the seller has to register the factoring agreement. The settlement information ($\mathcal{M}$) contains the details which will be stored on-chain, and includes the salted hash of $\mathcal{A}$ as a commitment and for non-repudiation purposes. The value r_1 is a random number (salt) to prevent brute-force guessing attacks on the hash of $\mathcal{A}$ in $\mathcal{M}$. The $\mathcal{M}$ includes the compliance certificate of the factor as well as his account number. The public fragment of the certificate (α_F) is directly included. However, the private fragment which is hidden from the seller and is included in the encrypted part of $\mathcal{M}$. We used a symmetric key (K_{FB}) to encrypt the mentioned information which is wrapped by the public key of the buyer ($\mathsf{Enc}(PU_B, K_{FB})$).

2.5. Sends the agreement and signed settlement information as well as the compliance proof to the seller over a secure channel:

$$F \rightarrow S : [\mathcal{A}, \mathcal{M}, r_1, \sigma^{@F}_{\mathcal{M}}, r_2, \pi_F]$$

The signature is generated with the factor's blockchain public key ($@F$). The compliance proof links the long-term public key of the factor with his blockchain address, and the commitment com_F. The secrets in the commitment are revealed in the encrypted part of $\mathcal{M}$ to give an evidence to the buyer about the identity of the factor who generated π_F.

3. The seller checks the details in $\mathcal{A}$ and $\mathcal{M}$, verifies the salted hash of $\mathcal{A}$ is correctly included in $\mathcal{M}$, verifies the commitment com_F matches r_2 and $\mathcal{C}_F$, and if agrees with them, proceeds by sending the registration request to the smart contract. In addition, she records the factor's signature for possible later use as digital evidence. The registration information is minimal and without any personally identifiable information. The request also includes an authenticity proof (denoted by π_S) which is a zk-SNARK proof which (1) proves that she is the authentic seller mentioned in the invoice

certificate, and (2) links her long-term public key with her blockchain address. The request is signed by one of the blockchain addresses of the seller, and sent to the smart contract in a blockchain transaction.

$$S \rightarrow C : [tx, \sigma_{tx}^{@S}],$$

where

$$tx = [\mathcal{M}, \sigma_{\mathcal{M}}^{@F}, \pi_F, \alpha_I, \pi_S].$$

The value of α_I is the public information of the invoice certificate which is required to verify π_S.

4. The smart contract:

   4.1. Verifies that the invoice (P_I) has not been already factored.

   4.2. Verifies the dates. In particular, verifies that the current blockchain time is: (1) prior the registration deadline (d_F in $\mathcal{M}$) and (2) after the issue date of the factor's compliance certificate ($date_F$). In addition, (3) the expiry date of the certificate is after the invoice deadline (d_I in α_I).

   4.3. Recovers the addresses of the seller ($@S$) and the factor ($@F$) from the transaction. Then, verifies π_S and π_F to ensure the that: (1) the invoice is valid, (2) the authentic seller has sent the transaction, (3) a complaint factor is involved, (4) the factor has confirmed the factoring, (5) the commitment com_F matches the certificate of the factor.

   4.4. If all the previous steps are correctly passed, it registers the factoring agreement by setting a flag in its key-value storage:

$$\langle P_I, PU_B \rangle \Rightarrow \texttt{true}$$

   4.5. Logs the details of the agreement to be used by the buyer for payment:

$$\log(P_I, PU_B, \mathcal{M}, @S, @F, \pi_F)$$

   P_I and PU_B are defined as both an index and the key of the log for quick search.

5. The factor proceeds to pay a_F to the account of the seller. This has to happen before the agreed payment deadline.

5.2.6 Phase 3: Payment

When the deadline of an invoice (d_I) expires, the buyer proceeds with the following steps to pay the factor. The buyer:

1. Queries the smart contract to figure out whether the invoice has been factored or not. Then, queries a node of the blockchain to obtain the log of the smart contract with the index field P_I and his/her public key PU_B.

2. From this on-chain log, the buyer obtains $\mathsf{Enc}(PU_B, K_{FB})$, uses his/her private key to decrypt it and recover K_{FB}. Then, decrypts the other encrypted part of $\mathcal{M}$ using K_{FB} and recovers the public key certificate of the factor, his compliance certificate, and his bank account number. The buyer also recovers the salt r_2 and with $\mathcal{C}_F$, he/she verifies that they match the commitment com_F in $\mathcal{M}$. The commitment is also verifies in π_F and ensures the buyer that $@F$ actually belongs to the factor who has generated π_F. If we did not have this commitment, π_F would prove that $@F$ belongs to a compliant factor but that factor might be different from the one that is mentioned in the encrypted part. In that case, we had two other options: (1) π_F should decrypt the encrypted parts of $\mathcal{M}$ and prove that the same certificate for the factor is included in them. This is a very heavy zero-knowledge proof, and we avoid that. (2) The factor should include a signature in $\mathcal{M}$ such as $\sigma^F_{@F}$ to link $@F$ to PU_F. This would also reveal the link to the public, and we should avoid that.

3. If the checks in the previous step succeed, the buyer knows the factor and his/her associated $IBAN_F$ and pays the invoice amount (a_I) to that $IBAN_F$.

5.2.7 Denial of Service Attack

The included public key (P_I) is a pseudo-identifier for the certificate, and only the correct seller has the corresponding private key. The seller uses a zk-SNARK

proof to prove possession of the private key to the smart contract. This prevents a fraudulent factor from depriving a seller from factoring his/her invoice. In this attack, once the seller negotiates with the factor, the factor may use the provided details about the invoice to impersonate the seller and fraudulently register the invoice instead of the original seller. If the attack succeeds, the smart contract will not allow the seller to register his/her factoring. In this case, even if anyone knows all the invoice details, including its pseudo-identifier (P_I), they cannot register it on the smart contract becase they cannot generate the zk-SNARK proof. The mentioned attack was possible in previous versions of our protocol. Nonetheless, the buyer did not pay the factor in the end, because the identity of the seller did not match the invoice. However, this was not possible to detect by the smart contract and would disrupt the factoring process.

5.3 Related Work

This section describes works in the literature that propose blockchain-based solutions using digital certificates. These works are the ones that are more related to our protocol. At the end of the section we provide a comparison of these works with our proposal in Table 4.1.

Yakubov et al.[36] presented a blockchain-based Public-Key Infrastructure (PKI) for issuing, validating, and revoking X.509 certificates. Blockchain smart contracts and PKI procedures as the underlying technology can guarantee authentication. For each Certificate Authorities(CA), a smart contract is established in the blockchain. The smart contract includes the CA certificate, hashes of the CA's issued certificates, and revocation status. They integrate metadata related to blockchain using extensions in standard X.509 certificates. When validating a TLS certificate, an extension is provided with the issuing CA's smart contract address and utilized to build a trust chain even though the certificate is not a root CA certificate. In reality, in the TLS client system, the essential certificates are validated by invoking a smart contract or utilizing a web service that checks the chain's authenticity. They do not require external auditing to assess their cryptographic correctness and behavior because new blocks are confirmed and appended to the log only when the underlying blockchain network has reached a consensus. Furthermore, it can be efficiently and quickly

monitored. To validate the TLS certificates in their system, clients must ask a web server or a bookkeeper, which violates their privacy.

CertLedger[37], the work presented by Kubilay et al., provides a blockchain system to replace the certificate validation and revocation logs as well as the certificate revocation list (CLR), eliminating CRL spoofing and retaining the full secure channels since the commencement of communication. CertLedger makes use of the blockchain to enable certificate transparency and provides an efficient means of certificate verification. It is a novel PKI proposal based on blockchain technology that provides certificate transparency in revocation and management of trustworthy CA activities. The basic idea behind this proposal is to use smart contracts to keep all transactions secure from Web server certificates. All of this is to prevent a data breach or hijack, given that the traditional CAs that produce and maintain certificates can go rogue, or the blockchain could be subject to a 51 percent attack. As a result, the actual security of the system is based on the architecture's cryptographic primitives.

Yang and Li [38] modified the current claim identification model in the blockchain using smart contracts and zero-knowledge proof (ZKP) methods to achieve identity unlinkability, thus preventing attribute ownership from being exposed. Fragmented identities, single points of failure, internal attacks, and data leakage have been issues with traditional centralized digital identity management systems (DIMS). DIMSs may be used in conjunction with blockchain technology, which considerably reduces the risks posed by a centralized third party. Nonetheless, the inherent transparency and lack of privacy constitute a significant barrier to DIMSs. They talked about the existing claim identity model as well as their suggested work on an improved identity claim model. Incorporating the attribute identifier and user's public key in the hash preimage avoids exposing the ownership of the privacy attribute in the public and transparent distributed ledger in the revised claim identification model. Furthermore, they created the BZDIMS system prototype, which features a challenge-response protocol that allows users to selectively disclose their ownership of characteristics to service providers in order to safeguard users' behavior privacy. Their approach achieved effective attribute privacy protection and a broader application scope, according to performance evaluation and security analysis. Yang and Li used zk-SNARKs for data minimization. The verifiable credential is issued as a committed form with its zk-SNARKs evidence in its design. The

user additionally develops a verified presentation that expresses the accuracy of the credential and its one-time use. Furthermore, the zk-SNARKs are used in their work to prove the ownership of a given attribute issued in advance. Furthermore, because they only use zk-SNARKs to improve verification efficiency, their work restricts the presentation format to revealing only the possession of an attribute, despite the expression power of the zk-SNARKs.

The table 5.1 summarizes the properties of state-of-the-art blockchain-based Zero-knowledge proof, digital certificate, and our idea.

Table 5.1: Comparison with close related work.

Proposal	Blockchain Permission type	Revocation	Blockchain type	Storage type	Privacy
Yakubov et al. [36]	Permissionless	Yes	Ethereum-based	on-chain:None off-chain: Public	No
CertLedger [37]	Permissionless	Yes	Ethereum-based	on-chain:Hash off-chain: Private	No
Yang and Li [38]	Permissionless	Yes	Ethereum-based	on-chain: Hash in a zk-SNARK off-chain: Public	Yes
Our Proposal	Permissionless	Yes	Ethereum-based	on-chain:Hash and Digital Signature in a zk-SNARK off-chain: Public	Yes

A short discussion of the general observations property by property follows:

Blockchain Permission Type: Since the permissionless blockchain is the same as the public blockchain, all the related works under consideration, including our proposed protocol, are permissionless. Permissionless implementations of Internet-scale applications are used in the chain consensus process to validate transactions and data and are available to everyone.

Revocation : Revocation is one of the security features of digital certificates in Blockchain and Zero-Knowledge proof. Because using blockchain, all approaches implement this feature. In the traditional method, reliable third parties were required for revocation, but in these systems, this revocation may not be done correctly due to the possibility of a MITM attack.

Blockchain type: Here, all implementations are based on Ethereum. Considering the reliability, non-manipulation, and availability of public blockchain is valid.

Storage type: This section examines storage in two forms, off-chain and on-chain. On-chain data storage in Yakubov et al. [36] implementation is stored certificate completely outside the chain. In CertLedger [37] the storage is only hash. In Yang and Li [38] work and our proposed protocol, if our certificates are to be stored, they are stored as a hash in zk-snark technology. In off-chain mode, storage for the kubilay project is private, and the rest of the related works are public.

Privacy: Yakubov et al. [36] andCertLedger [37] do not explicitly consider privacy in their implementation. While Yang and Li [38] and our proposed protocol take into the privacy of each entity. We do not store sensitive data directly on the blockchain for privacy reasons. Instead, before being put on a chain, the data portion is symmetrically encrypted. The extensions provide the smart contract to check factoring agreements before they are registered without jeopardizing the privacy of the persons concerned. Not all parties engaged in a factoring scenario should have access to all information of the invoice/certificate for privacy reasons.

5.4 Conclusions

In this chapter we introduced a protocol for an invoice factoring system with enhanced privacy. We utilized non-interactive zero-knowledge proofs for enhancing privacy. In particular, we use two ZKPs: one proof is created by the seller and it confirms that the buyer approves the invoice. This proof includes the blockchain address of the owner of the invoice. The other proof is created by the factor and it proves that an accreditation authority approves the factor. As we detailed in the explanations in the chapter, these zero-knowledge proofs also prevent possible denial of service attacks. Future work will concentrate on the detailed implementation of these proofs and in getting experimental results.

Chapter 6

Conclusions and Future Work

Although invoice factoring is a substantial part in the worldwide financial sector, the absence of a single source of truth makes it ripe for the double-factoring fraud. In this thesis we propose a blockchain-based architecture for invoice factoring registration. The architecture is gradually developed and enhanced in Chapters 3, 4 and 5. In the first protocol, we proposed our base architecture for factoring registration using a public blockchain. The protocol is intended to limit the buyer's participation in the factoring process. The buyer was only required to publish a hash of invoice details for verification by the factor, and the rest of the process was carried out by the sellers and the factors using on-chain and off-chain communications. The on-chain details were registered by a smart contract which also protected our system from double-factoring attacks. Simultaneously, pseudo-anonymous identifiers, symmetric encryption, and cryptographic commitments were used for on-chain data, to improve the privacy of the participants. The registered information was used by the buyer to pay the corresponding factor. In addition, it could be used as digital evidence for dispute resolution. A comparison with the literature revealed that our proposal were superior to the related works with respect to efficiency, privacy, and buyer's involvement.

The second proposed protocol was an evolution of our first protocol, with enhancements in flexibility and user on-boarding. We employed Decentralized IDentifiers (DIDs) and allowed the participants to use their self-sovereign identities (SSIs). One benefit of adopting DIDs was that we could rely on new protocols created in that ecosystem, such as DIDComm, to perform secure

asynchronous communications between participants. We were also enabled to use DIDs for granting permission to parties who were allowed to participate in our invoice factoring ecosystem by leveraging the concept of Verifiable Credentials (VCs). A relayer was also added to our ecosystem for facilitating the onboarding of sellers on the system. In particular, the sellers could send their transactions to the relayer to free themselves from dealing with cryptocurrency, and at the same time, benefit from the high security and availability of a public distributed ledger. This was achieved because the relayer sent the sellers' transactions to the blockchain (as a meta-transaction), and payed all respective blockchain fees on behalf of the sellers. The sellers could pay the relayers via off-chain methods (such as credit card or bank transfer). Therefore, the second proposal was easier to be used for registering invoice factorings. The improvements did not negatively affect our efficiency in comparison to our first protocol as only one meta-transaction were required per factoring registry.

In the third and last evolution of the protocol, we proposed to use zero-knowledge proofs (ZKPs) to allow the smart contract to do a more complete validation before a registration is actually realized and this was achieved without hindering privacy. In particular, we proposed to use non-interactive zero-knowledge proofs as validity evidences for protecting most invoice details, and the identity of involved parties from prying eyes. One of the proofs were created by the factor to prove approval from an authority and another proof was provided by the seller to confirm that the he/she was authorized to factor the invoice. We plan to further enhance our system in the future by conducting more comprehensive experimental assessments and analyses. Finally, we would like to remark that our ideas can be applied to other use cases and ecosystems in which efficient and privacy-preserving registrations are needed while having a high degree of privacy and where easy user on-boarding is important.

Appendix A

AppendixA

This appendix shows the main smart contract proposed for implementing the registration of invoice factoring. The smart contract is written in Solidity and it registers information from the seller and factor. The smart contract allows the buyer to determine whether an invoice identified by P_I has been previously registered or not.

```
1  // contracts/FactoringProcess.sol
2
3  // SPDX-License-Identifier: MIT
4
5  pragma solidity ^0.8.0;
6
7  contract FactoringProcess {
8
9  mapping(bytes => bool) private alreadyRegistered;
10
11   event NewInvoiceRegistered(
12     bytes indexed p_I,
13     uint256 due_time,
14     string enc_factor_bank,
15     string public_bank,
16     uint256 r,
17     uint256 h,
18     address p_u_s,
19     address p_u_f
20   );
21
22
```

```
23
24 function invoiceRegistration(
25     bytes memory p_I,
26     uint256 due_time,
27     string memory enc_factor_bank,
28     string memory public_bank,
29     uint256 r,
30     uint256 h,
31     bytes memory sign_DID_F,
32     bytes memory sign_DID_S
33   ) public {
34     address p_u_s;
35     address p_u_f;
36
37     require(
38     block.timestamp < due_time,
39     "META_TX: Meta transaction is expired"
40     );
41
42     bytes32 M = keccak256(abi.encodePacked(
43     p_I,due_time,enc_factor_bank,public_bank,r,h,sign_DID_F));
44     bytes32 S = keccak256(
45     abi.encodePacked(p_I, due_time, enc_factor_bank, public_bank, r, h)
46     );
47     p_u_s = recoverSigner(M, sign_DID_S);
48
49     p_u_f = recoverSigner(S, sign_DID_F);
50
51     require(!InvoiceRegister(p_I), "Already registered.");
52
53     AlreadyRegistered[p_I] = true;
54
55     emit newInvoiceRegistered(
56     p_I,due_time,enc_factor_bank,public_bank,r,h,p_u_s,p_u_f
57     );
58
59   }
60
61 function isInvoiceRegistered(bytes memory p_I) public view returns (bool) {
62     return AlreadyRegistered[p_I];
63   }
64
65
66
```

```
67
68  function getFirstCall(string memory yourname)
69      public
70      pure
71      returns (string memory)
72    {
73      return string(abi.encodePacked("Hello ", yourname));
74    }
75  function get_S(
76      bytes memory p_I,
77      uint256 due_time,
78      string memory enc_factor_bank,
79      string memory public_bank,
80      uint256 r,
81      uint256 h
82    ) public pure returns (bytes32) {
83
84      bytes32 S = keccak256(
85      abi.encodePacked(p_I, due_time, enc_factor_bank, public_bank, r, h)
86      );
87      return (S);
88
89    }
90  function get_M(
91      bytes memory p_I,
92      uint256 due_time,
93      string memory enc_factor_bank,
94      string memory public_bank,
95      uint256 r,
96      uint256 h,
97      bytes memory sign_DID_F
98    ) public view returns (bytes32 M) {
99      require(
100     block.timestamp < due_time,
101     "META_TX: Meta transaction is expired");
102
103     return keccak256(abi.encodePacked(
104     p_I,due_time,enc_factor_bank,public_bank,r,h,sign_DID_F));
105   }
106
107
108
109
110
```

```
111
112
113 function prefixed(bytes32 hash) internal pure returns (bytes32) {
114     return
115     keccak256(
116     abi.encodePacked("\x19Ethereum Signed Message:\n32", hash)
117   );
118   }
119
120 function splitSignature(bytes memory sig)
121   internal pure returns (
122     uint8 v,
123     bytes32 r,
124     bytes32 s
125   )
126   {
127     require(sig.length == 65);
128     assembly {
129     r := mload(add(sig, 32))
130     s := mload(add(sig, 64))
131     v := byte(0, mload(add(sig, 96)))
132   }
133   return (v, r, s);
134 }
135
136 function recoverSigner(bytes32 message, bytes memory sig)
137   internal pure returns (address)
138     {
139     (uint8 v, bytes32 r, bytes32 s) = splitSignature(sig);
140     return ecrecover(message, v, r, s);
141     }
142   }
```

Bibliography

[1] Ettore Battaiola, Fabio Massacci, Chan Nam Ngo, and Pierantonia Sterlini. Blockchain-based invoice factoring: from business requirements to commitments. In Proceedings of the Second Distributed Ledger Technology Workshop (DLT@ITASEC), volume 2334 of CEUR Workshop Proceedings, pages 17–31. CEUR-WS.org, February 2019. URL http://ceur-ws.org/Vol-2334/DLTpaper2.pdf.

[2] Sandeep Goel. Financial Services. PHI Learning Pvt. Ltd., 2011.

[3] Behalf Company. 5 most common invoice factoring problems, January 2017. URL https://www.behalf.com/merchants/factoring/invoice-factoring-problems/.

[4] G David Keaton and Samara Keaton. Factoring system and method, November 2009. US Patent 7,617,146.

[5] Hidde Lycklama à Nijeholt, Joris Oudejans, and Zekeriya Erkin. Decreg: A framework for preventing double-financing using blockchain technology. In Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts (BCC), pages 29–34, New York, NY, USA, 2017. ISBN 9781450349741.

[6] Nader Mohamed and Jameela Al-Jaroodi. Applying blockchain in industry 4.0 applications. In Proceedings of IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), pages 0852–0858. IEEE, 2019.

[7] Bhabendu Kumar Mohanta, Soumyashree S Panda, and Debasish Jena. An overview of smart contract and use cases in blockchain technology. In Proceedings of the 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), pages 1–4. IEEE, 2018.

[8] Meriem Guerar, Alessio Merlo, Mauro Migliardi, Francesco Palmieri, and Luca Verderame. A fraud-resilient blockchain-based solution for invoice financing. IEEE Transactions on Engineering Management, 67(4):1086–1098, 2020.

[9] Juan Benet. IPFS - content addressed, versioned, P2P file system. https://arxiv.org/abs/1407.3561, July 2014. [Accessed 10-May-2021].

[10] Whitfield Diffie and Martin Hellman. New directions in cryptography. IEEE transactions on Information Theory, 22(6):644–654, 1976.

[11] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. Management Information Systems Quarterly, 28(1):75–105, 03 2004.

[12] Nasibeh Mohammadzadeh, Sadegh Dorri Nogoorani, and José Luis Muñoz-Tapia. Invoice factoring registration based on a public blockchain. IEEE Access, 9:24221–24233, 2021.

[13] Nasibeh Mohammadzadeh, Sadegh Dorri Nogoorani, and José Luis Muñoz-Tapia. Decentralized factoring for self-sovereign identities. Electronics, 10 (12):1467, 2021.

[14] Imran Bashir. Mastering blockchain. Packt Publishing Ltd, 2017.

[15] Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. Blockchain technology overview. https://arxiv.org/abs/1906.11078, 2019. [Accessed 10-May-2021].

[16] Andreas M Antonopoulos. Mastering Bitcoin: unlocking digital cryptocurrencies. O'Reilly Media, Inc., 2014.

[17] Victor Youdom Kemmoe, William Stone, Jeehyeong Kim, Daeyoung Kim, and Junggab Son. Recent advances in smart contracts: A technical overview and state of the art. IEEE Access, 8:117782–117801, 2020.

[18] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. White Paper EIP-150, The Ethereum Foundation, 2014. [Accessed 10-May-2021].

[19] Huashan Chen, Marcus Pendleton, Laurent Njilla, and Shouhuai Xu. A survey on ethereum systems security: Vulnerabilities, attacks, and defenses. ACM Computing Surveys (CSUR), 53(3):1–43, 2020.

[20] Sara Rouhani and Ralph Deters. Security, performance, and applications of smart contracts: A systematic survey. IEEE Access, 7:50759–50779, 2019.

[21] Jing Liu and Zhentian Liu. A survey on security verification of blockchain smart contracts. IEEE Access, 7:77894–77904, 2019.

[22] William Stallings. Cryptography and Network Security: Principles and Practice. Pearson, 2017.

[23] C. Percival and S. Josefsson. The scrypt password-based key derivation function, internet engineering task force (ietf) request for comments 7914. https://www.rfc-editor.org/rfc/rfc7914.html, August 2016.

[24] Drummond Reed, Manu Sporny, Dave Longley, Christopher Allen, Ryan Grant, Markus Sabadello, and Jonathan Holt. Decentralized identifiers (DIDs) v1. 0: Core architecture, data model, and representations. https://www.w3.org/TR/did-core/, 2020. [Accessed 10-May-2021].

[25] Orie Steele and Manu Sporny. Did specification registries. https://www.w3.org/TR/did-spec-registries/, 2021. [Accessed 10-May-2021].

[26] Manu Sporny, Dave Longley, and Chadwick David. Verifiable credentials data model 1.0: Expressing verifiable information on the web. https://www.w3.org/TR/2019/REC-vc-data-model-20191119/, 2019. [Accessed 10-May-2021].

[27] D. Hardman. Didcomm messaging. https://identity.foundation/didcomm-messaging/spec/, . [Accessed 10-May-2021].

[28] Will Abramson, Adam James Hall, Pavlos Papadopoulos, Nikolaos Pitropakis, and William J Buchanan. A distributed trust framework for privacy-preserving machine learning. In International Conference on Trust and Privacy in Digital Business, pages 205–220. Springer, 2020.

[29] Daniel Hardman. Peer dids: a secure and scalable method for dids that's entirely off-ledger. https://ssimeetup.org/

peer-dids-secure-scalable-method-dids-off-ledger-daniel-hardman-webinar-42/, November 2019. [Accessed 10-May-2021].

[30] Daniel Hardman. Didcomm implementers guide. https://identity.foundation/didcomm-messaging/guide/, . [Accessed 10-May-2021].

[31] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. SIAM Journal on computing, 18(1): 186–208, 1989.

[32] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In 2013 IEEE Symposium on Security and Privacy, pages 238–252. IEEE, 2013.

[33] SECG SEC. 1: Elliptic curve cryptography, version 2.0. Standards for Efficient Cryptography Group, 2009.

[34] Etherscan node tracker. https://etherscan.io/nodetracker. [Accessed 10-May-2021].

[35] Gas station network (gsn). https://docs.opengsn.org/. [Accessed: 10-May-2021].

[36] Alexander Yakubov, Wazen Shbair, Anders Wallbom, David Sanda, et al. A blockchain-based pki management framework. In The First IEEE/IFIP International Workshop on Managing and Managed by Blockchain (Man2Block) colocated with IEEE/IFIP NOMS 2018, Tapei, Tawain 23-27 April 2018, 2018.

[37] Murat Yasin Kubilay, Mehmet Sabir Kiraz, and Hacı Ali Mantar. Certledger: A new pki model with certificate transparency based on blockchain. Computers & Security, 85:333–352, 2019.

[38] Xiaohui Yang and Wenjie Li. A zero-knowledge-proof-based digital identity management scheme in blockchain. Computers & Security, 99:102050, 2020.