electronics

MDPI

Article

Decentralized Factoring for Self-Sovereign Identities

Nasibeh Mohammadzadeh [1], Sadegh Dorri Nogoorani [2,*] and José Luis Muñoz-Tapia [1]

[1] Department of Network Engineering, Campus Nord, Universitat Politècnica de Catalunya (UPC), 08034 Barcelona, Spain; nasibeh.mohammadzadeh@upc.edu (N.M.); jose.luis.munoz@upc.edu (J.L.M.-T.)
[2] Faculty of Electrical and Computer Engineering, Tarbiat Modares University, Tehran P.O. Box 14115-111, Iran
* Correspondence: dorri@modares.ac.ir; Tel.: +98-21-82883905

Abstract: Invoice factoring is a handy 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. 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 article, a factoring registration protocol is proposed for invoice factoring registration based on a public distributed ledger which adheres to the aforementioned requirements. We include a relayer in our architecture to address the entry barrier that the users have due to the need of managing cryptocurrencies for interacting with the public ledger. Moreover, we leverage the concept of Verifiable Credentials (VCs) for KYC compliance, and allow 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. We analyze our protocol from several security aspects, compare it to the related work, and study a possible business use case. Our evaluations demonstrate that our proposal is secure and efficient, as well as covers requirements not addressed by existing related work.

Keywords: invoice factoring; public distributed ledger; blockchain; smart contract; decentralized identifiers; self-sovereign identities; DIDComm; relayer; dispute resolution

Citation: Mohammadzadeh, N.; Dorri Nogoorani, S.; Munoz-Tapia, J. L. Decentralized Factoring for Self-Sovereign Identities. *Electronics* **2021**, *10*, 1467. https://doi.org/10.3390/electronics10121467

Academic Editors: Miguel Soriano and Javier Lopez

Received: 15 May 2021
Accepted: 15 June 2021
Published: 18 June 2021

1. Introduction

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 [1]. There are several issues and challenges in the traditional invoice factoring process [2]. For example, it often requires several manual steps, and 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 have paid the amount of invoice by the due deadline, and the *buyer* has to comply with the factoring contract between the *seller* and the *factor*. Moreover, a malicious *seller* may try to cash an invoice at multiple *factors* to fraudulently double the amount of received money. This issue is known as *double factoring*, and it is a main problem that a factoring system needs to prevent. In more detail, *double factoring* is possible because there are no insights between *factors*, whether an invoice has already been financed or not [5]. 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. 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 involved 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 distributed ledger seems a natural tool to solve these issues because not only can it keep the record of factoring agreements but it can 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. 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 [2,5,7,8]. However, as we discuss in detail in Section 6.2, 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-efficient, and provide an easy user on-boarding.

In this article, we propose a protocol that fulfills all the previous requirements, 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. To overcome this problem, we include a *relayer* in our architecture. Additionally, the proposed protocol also enables new functionality that is 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 (DIDs). 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.

Our contributions are designing the system architecture, procedures, and communication protocols for an efficient system which is decentralized and highly available. We use different cryptographic primitives to make our architecture secure against attacks and preserve the privacy of involved parties. Our registration system relies on a public distributed ledger to prevent double-factoring and protect digital evidence from manipulations. The involved parties identify each other in a secure and privacy-preserving manner to comply with the KYC regulation. While we rely on a public distributed ledger, the parties are not required to use cryptocurrencies. Moreover, the *buyer* is not required to invest too many resources, nor be heavily involved in the factoring process.

The rest of this article is organized as follows: in Section 2, we present the followed methodology; in Section 3, we provide the background; in Section 4, we present the related

work; in Section 5, we introduce our protocol; in Section 6, we evaluate the proposal; and we conclude in Section 7.

2. Methodology

For the design of our ledger-based invoice factoring solution, we followed the design-science research methodology [9]. 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 addition, 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-efficient, and provide an easy user on-boarding.

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 [2,5,7,8]. However, as we discuss in detail in Section 6.2, none of them was completely able to fulfill the requirements.

In the third step, we designed a system architecture, and developed its procedures and communication protocols based on the distributed ledger technology for an efficient invoice factoring system. We made use of proven and solid cryptographic primitives to make our design secure while being functional and efficient. In Section 3, we briefly introduce the cryptographic primitives and the technology we use in our design.

The fourth step was to evaluate our solution and compare it with the related work. Our solution suites different cases for business use. Nonetheless, as an example, we demonstrate its applicability in a digital data marketplace in Section 5.3. Besides that, we analyzed the security of our proposal and compared it with the related work. According to the methodology, the remainder of the article presents our research, design, and evaluations.

3. Background

3.1. Cryptographic Primitives

A number of cryptographic primitives have been used to secure our design, and we briefly introduce them next. The interested reader is referred to reference [10] 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` [11]) 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 [12] 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.

3.2. 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 [13]. A block contains several transactions, and an important property is that, once the consensus algorithm definitively accepts a block, all the nodes will know this block, and it will be impossible to manipulate or delete it [14].

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 [15].

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 [16] and develop *decentralized applications* (dapps).

Ethereum [17] is the most popular public blockchain capable of running smart contracts, and the platform of choice for many developers for implementing dapps [18]. 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 [19].

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 [20].

3.3. 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) [21] 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 [22]. 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 permissionless), 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 [23] 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.

3.4. DIDComm

DIDComm is an asynchronous communication protocol for establishing secure and private channels between parties based on their DIDs [24]. DIDComm supports both centralized and decentralized communication models. Different parties do not need a highly available webserver 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 [25]. In a decentralized and ad-hoc case, the endpoint is Bob's agent, who is accessible through the Internet [26]. 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 [27]. 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.

4. Related Work

This section describes solutions that have been published in the literature, and that propose similar approaches to us to solve invoice factoring. That is to say, approaches that propose solutions based on distributed ledger technologies.

The first work that is worth mentioning is DecReg [5], which has been used by the Netherlands financial industry. DecReg can be used to track fiat payments and invoices that have been factored. The DecReg framework prevents double factoring by design and is implemented over a private blockchain. The operation of the framework requires that *buyers* operate a node in the private blockchain. To do so, *buyers* receive credentials from a Central Authority (CA). The CA not only provides credentials but also monitors the access to the private blockchain and prevents uncertified parties from accessing confidential information. Regarding dispute resolution, if an argument between a *seller* and a *factor* takes place, in DecReg, the signatures of transactions (of the private ledger) are used to resolve the dispute.

Battaiola et al. [7] proposed a framework for registering factoring agreements that can prevent double factoring and preserve the privacy of involved parties. The proposed architecture employs a distributed ledger as the source of truth. All parties submit their private inputs in the form of commitments to ensure the integrity and confidentiality of factoring data. The protocol operates over a private blockchain network. In particular, authors suggest the use of Hyperledger Fabric [28]. Involved parties have to operate the infrastructure of the private ledger. In addition, the registration of the factoring of an invoice requires that each involved party sends a transaction to the private ledger.

Guerar et al. [8] propose a factoring scheme based on a public distributed ledger. In particular, authors suggest the use of the Ethereum public blockchain network [17]. In the proposed protocol, *buyers* are not considered trustworthy. Following this assumption, the authors develop a framework to assess the credibility of *buyers* based on reputation. That

is to say, a reputation record is created for each *buyer* based on his previous behavior. To implement this reputation system, the platform is in charge of creating stable identifiers for linking reputation records to each *buyer*. On the other hand, in the proposed framework, factoring is negotiated using an on-chain auction in which any investor, not just banks and financial institutions, can register as a *factor*. Additionally, the protocol is specified for products rather than for services, as the authors mention that the invoice factoring negotiation process begins when transported goods are received. Finally, to provide data availability, authors suggest the use of a peer-to-peer distributed file system. In particular, authors base their solution in the Inter-Planetary File System (IPFS) [29].

In reference [2], the authors introduce a framework for factoring registration based on the use of a public distributed ledger. The proposed protocol is designed to reduce the *buyer*'s involvement in the factoring process. In particular, the *buyer* is only required to publish a hash of invoice details for *factor* verification, and the rest of the process was carried out by *sellers* and *factors* using on-chain and off-chain communications. A smart contract is used to register invoice factoring details on-chain efficiently and to avoid double factoring. In particular, only the *seller* needs to interact with the smart contract and only one transaction is sent to the public distributed ledger to complete the registration of the factoring process. In addition, authors use pseudo-anonymous identifiers, symmetric encryption for on-chain data and cryptographic commitments to improve the privacy of *sellers* and *factors*. In this protocol, the *buyer* uses the registered information in the public distributed ledger to pay to the corresponding *factor*. The payment is done off-chain via a bank transfer using fiat money. Finally, after the registration process takes place, the data stored on-chain can be used as digital evidence for the resolution of possible disputes between involved parties.

5. 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 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.

5.1. 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. 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*.

Our architecture is for a registration system but the actual payments are made off-chain using fiat transfers between bank accounts. 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. 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*.

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*, as shown in Figure 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.).

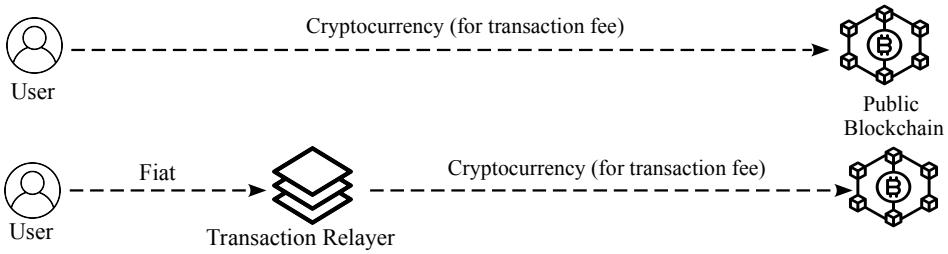

Figure 1. Transaction relayer.

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. Instead, the *buyer*'s software agent gives access to some minimal information about his invoices so that *factors* can check the information provided by *sellers*.

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 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.

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 service/goods provided by the *seller* to the *buyer*. We assume that the identity of the *seller* and the invoice number are enough to identify the invoice uniquely; 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.

5.2. The Protocol

Our 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 order to comply with the KYC regulation, in our 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 1, we show the notation that we utilize to describe our protocol.

Table 1. Notation.

Notation	Meaning
$\mathrm{KDF}(m)$	secure symmetric key derived from m (deterministic)
$\mathrm{h}(s, m)$	a salted hash function using salt s and string m.
ID_A	real identity of entity A
$@A$	blockchain address of entity A
DID_A	the public DID of entity A
PU_A	the public component of digital signature key-pair of A
$\mathrm{Enc}(K, m)$	symmetric encryption of m using key K
σ_m^X	digital signature over message m; X can be the signer's address or DID
I	invoice number
S	*seller*
B	*buyer*
F	*factor*
C	*smart contract*
R	*relayer*

5.2.1. 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 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:

$$P_F = \mathrm{h}(s, (DID_F, PU_F, ID_F)). \quad (1)$$

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:

$$C_F = (P_F, DID_F, PU_F, ID_F). \quad (2)$$

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

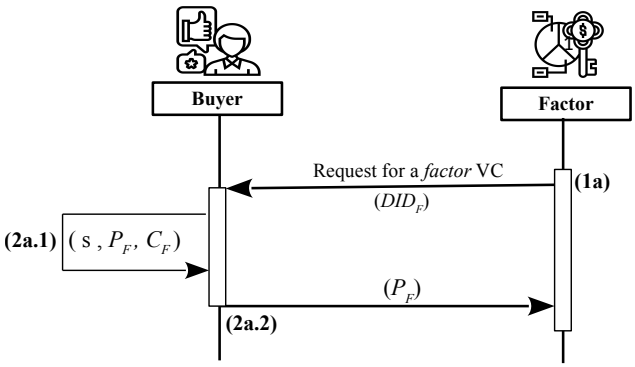

Figure 2. Registration of a *factor*'s VC.

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` [11]) 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 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:

$$C_I = (P_I, DID_S, PU_S, ID_S, I, a_I, d_I, @C), \quad (3)$$

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 (1), but computing the hash over the contents of C_I:

$$P_I = \mathrm{h}(s, (DID_S, PU_S, ID_S, I, a_I, d_I, @C)). \quad (4)$$

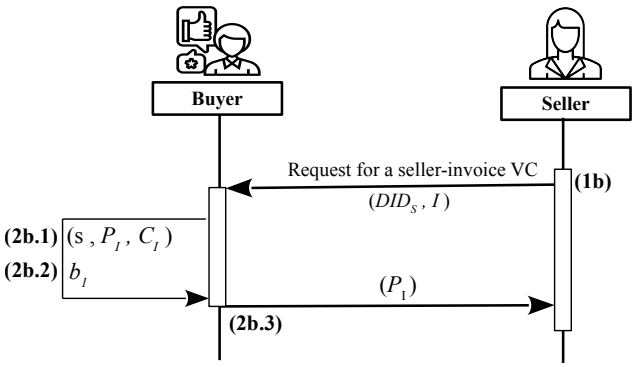

Figure 3. Registration of an invoice for a *seller* (seller-invoice VC).

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 (3)).

5.2.2. Phase 2: Factoring

According to Figure 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 (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 (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 $\sigma_{\mathcal{S}}^{DID_F}$):

$$F \rightarrow S: \quad \mathcal{A}, \mathcal{S}, \sigma_{\mathcal{S}}^{DID_F}, \tag{5}$$

$$\mathcal{A} = (C_I, C_F, a_F, IBAN_S), \tag{6}$$

$$\mathcal{S} = (P_I, d_F, \mathrm{Enc}(K_{FB}, C_F, IBAN_F), g^{b_2}, r, \mathrm{h}(r, \mathcal{A})), \tag{7}$$

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}:

$$K_{FB} = \mathrm{KDF}((g^{b_1})^{b_2}). \tag{8}$$

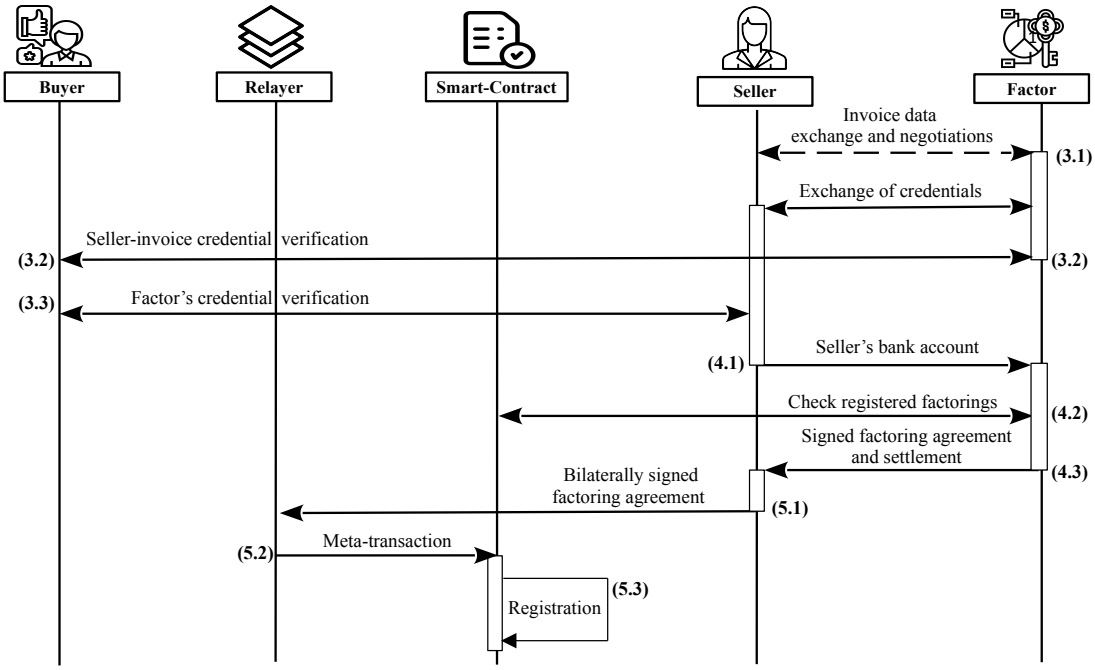

Figure 4. The factoring phase.

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 5.1, 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 (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 $\mathcal{M}$:

$$\mathcal{M} = (\mathcal{S}, \sigma_{\mathcal{S}}^{DID_F}). \tag{9}$$

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

$$S \rightarrow R: \quad (\mathcal{M}, \sigma_{\mathcal{M}}^{DID_S}). \tag{10}$$

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* deduces 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*:

$$R \rightarrow C: \quad (mtx, \sigma_{mtx}^{@R}), \tag{11}$$

$$mtx = (\mathcal{M}, \sigma_{\mathcal{M}}^{DID_S}). \tag{12}$$

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*:

$$P_I \Rightarrow \texttt{true}. \tag{13}$$

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:

$$\texttt{log}(P_I, \mathcal{S}, PU_S, PU_F). \tag{14}$$

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 $\mathrm{h}(r, \mathcal{A})$ 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.

5.2.3. 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 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 $\mathrm{Enc}(K_{FB}, C_F, IBAN_F)$ 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*.

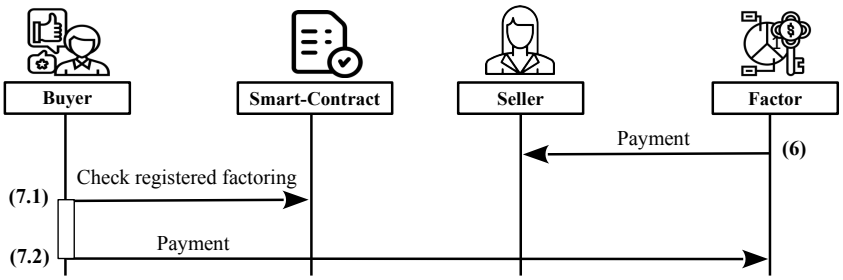

Figure 5. Payment phase.

5.3. Use Case

This section describes a typical scenario for better understanding the process in which our system 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* 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 paper 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*.

6. Evaluation

In this section, we provide a security analysis of our proposal and a comparison with related work.

6.1. 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).

6.1.1. 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.

6.1.2. 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.

6.1.3. 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 [30]. 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) [31], to fulfill its duties. As we do not trust the *relayer*, this will not pose any threats to the security of our protocol.

6.1.4. 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.

6.2. Comparison to Related Work

Below, we compare our protocol with the related work described in Section 4. A summary of this comparison is shown in Table 2.

Table 2. Comparison with related work.

	[5]	[7]	[8]	[2]	**This Proposal**
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. [7] propose to use a private ledger, while

Guerar et al. [8], Mohammadzadeh et al. [2], and the protocol proposed in this article 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. [7] 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 reference [7], it is not specified who is responsible for the cost of operating the private ledger.

As shown in Table 2, Guerar et al. [8], Mohammadzadeh et al. [2], and the proposed protocol 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 reference [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.

Mohammadzadeh et al. [2] proposed a protocol 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 the protocol by Mohammadzadeh et al., 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 article, 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 2, our 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 (DIDs). 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.

7. Conclusions

In this article, we presented a protocol that uses a public distributed ledger to register invoice factorings. The protocol presented is based on some preliminary work [2], but we added several enhancements and simplified the protocol to increase its 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 was very efficient, using 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.

Author Contributions: The authors wrote the manuscript, outlined the figures, contributed to discussion, and finalized the manuscript. Individual contributions are: Conceptualization, N.M., S.D.N. and J.L.M.-T.; Formal analysis, S.D.N. and J.L.M.-T.; Funding acquisition, J.L.M.-T.; Project administration, J.L.M.-T.; Resources, J.L.M.-T.; Writing—original draft, N.M., S.D.N. and J.L.M.-T.; Writing—review & editing, N.M., S.D.N. and J.L.M.-T. All authors have read and agreed to the published version of the manuscript.

Funding: This research has been funded by i3Market (H2020-ICT-2019-2 grant number 871754). This work is also supported by the TCO-RISEBLOCK (PID2019-110224RB-I00), ARPASAT (TEC2015-70197-R), and by the Generalitat de Catalunya grant 2014-SGR-1504.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Goel, S. *Financial Services*; PHI Learning Pvt. Ltd.: Delhi, India, 2011.
2. Mohammadzadeh, N.; Dorri Nogoorani, S.; Muñoz-Tapia, J.L. Invoice Factoring Registration Based on a Public Blockchain. *IEEE Access* **2021**, *9*, 24221–24233. [CrossRef]
3. Behalf Company. 5 Most Common Invoice Factoring Problems. 2017. Available online: https://www.behalf.com/merchants/factoring/invoice-factoring-problems/ (accessed on 10 May 2021).
4. Keaton, G.D.; Keaton, S. Factoring System and Method. U.S. Patent 7,617,146 B2, 10 November 2009.
5. Lycklama à Nijeholt, H.; Oudejans, J.; Erkin, Z. DecReg: A Framework for Preventing Double-Financing Using Blockchain Technology. In Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts (BCC), Abu Dhabi, United Arab Emirates, 2 April 2017; pp. 29–34.
6. Mohamed, N.; Al-Jaroodi, J. Applying blockchain in industry 4.0 applications. In Proceedings of the IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 7–9 January 2019; pp. 852–858.
7. Battaiola, E.; Massacci, F.; Ngo, C.N.; Sterlini, P. Blockchain-based Invoice Factoring: From business requirements to commitments. In *Second Distributed Ledger Technology Workshop*; CEUR Workshop Proceedings; DLT@ITASEC: Ancona, Italy, 2019; Volume 2334, pp. 17–31.
8. Guerar, M.; Merlo, A.; Migliardi, M.; Palmieri, F.; Verderame, L. A Fraud-Resilient Blockchain-Based Solution for Invoice Financing. *IEEE Trans. Eng. Manag.* **2020**, *67*, 1086–1098. [CrossRef]
9. Hevner, A.R.; March, S.T.; Park, J.; Ram, S. Design Science in Information Systems Research. *Manag. Inf. Syst. Q.* **2004**, *28*, 75–105. [CrossRef]
10. Stallings, W. *Cryptography and Network Security: Principles and Practice*; Pearson: London, UK, 2017.
11. Percival, C.; Josefsson, S. The Scrypt Password-Based Key Derivation Function, Internet Engineering Task Force (IETF) Request for Comments 7914. 2016. Available online: https://www.rfc-editor.org/rfc/rfc7914.html (accessed on 10 May 2021).
12. Diffie, W.; Hellman, M. New directions in cryptography. *IEEE Trans. Inform. Theory* **1976**, *22*, 644–654. [CrossRef]
13. Bashir, I. *Mastering Blockchain*; Packt Publishing Ltd.: Birmingham, UK, 2017.
14. Yaga, D.; Mell, P.; Roby, N.; Scarfone, K. Blockchain technology Overview. *arXiv* **2019**, arxiv:1906.11078.
15. Antonopoulos, A.M. *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2014.
16. Kemmoe, V.Y.; Stone, W.; Kim, J.; Kim, D.; Son, J. Recent advances in smart contracts: A technical overview and state of the art. *IEEE Access* **2020**, *8*, 117782–117801. [CrossRef]
17. Wood, G. Ethereum: A Secure Decentralised Generalised Transaction Ledger. 2014. Available online: https://gavwood.com/paper.pdf (accessed on 10 May 2021).
18. Chen, H.; Pendleton, M.; Njilla, L.; Xu, S. A survey on Ethereum systems security: Vulnerabilities, attacks, and defenses. *ACM Comput. Surv.* **2020**, *53*, 1–43. [CrossRef]
19. Rouhani, S.; Deters, R. Security, performance, and applications of smart contracts: A systematic survey. *IEEE Access* **2019**, *7*, 50759–50779. [CrossRef]
20. Liu, J.; Liu, Z. A survey on security verification of blockchain smart contracts. *IEEE Access* **2019**, *7*, 77894–77904. [CrossRef]
21. Reed, D.; Sporny, M.; Longley, D.; Allen, C.; Grant, R.; Sabadello, M.; Holt, J. Decentralized Identifiers (DIDs) V1. 0: Core Architecture, Data Model, and Representations. 2020. Available online: https://www.w3.org/TR/did-core/ (accessed on 10 May 2021).
22. Steele, O.; Sporny, M. DID Specification Registries. 2021. Available online: https://www.w3.org/TR/did-spec-registries/ (accessed on 10 May 2021).
23. Sporny, M.; Longley, D.; David, C. Verifiable Credentials Data Model 1.0: Expressing Verifiable Information on the Web. 2019. Available online: https://www.w3.org/TR/2019/REC-vc-data-model-20191119/ (accessed on 10 May 2021).
24. Hardman, D. DIDComm Messaging. Available online: https://identity.foundation/didcomm-messaging/spec/ (accessed on 10 May 2021).
25. Abramson, W.; Hall, A.J.; Papadopoulos, P.; Pitropakis, N.; Buchanan, W.J. A Distributed Trust Framework for Privacy-Preserving Machine Learning. In Proceedings of the International Conference on Trust and Privacy in Digital Business, Bratislava, Slovakia, 14–17 September 2020; Springer: Berlin/Heidelberg, Germany, 2020; pp. 205–220.
26. Hardman, D. Peer DIDs: A Secure and Scalable Method for DIDs That's Entirely Off-Ledger. 2019. Available online: https://ssimeetup.org/peer-dids-secure-scalable-method-dids-off-ledger-daniel-hardman-webinar-42/ (accessed on 10 May 2021).
27. Hardman, D. DIDComm Implementers Guide. Available online: https://identity.foundation/didcomm-messaging/guide/ (accessed on 10 May 2021).
28. Hyperledger Fabric. 2021. Available online: https://www.hyperledger.org/use/fabric (accessed on 7 June 2021).
29. Benet, J. IPFS—Content Addressed, Versioned, P2P File System. *arXiv* **2014**, arxiv:1407.3561.
30. Etherscan Node Tracker. Available online: https://etherscan.io/nodetracker (accessed on 10 May 2021).
31. Gas Station Network (GSN). Available online: https://docs.opengsn.org/ (accessed on 10 May 2021).