IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication 2021

Digital Object Identifier Postprint

Invoice Factoring Registration Based on a Public Blockchain

NASIBEH MOHAMMADZADEH[1], SADEGH DORRI NOGOORANI[2], AND JOSÉ LUIS MUÑOZ-TAPIA.[3]

[1]Department of Network Engineering, Polytechnic University of Catalonia, Spain (e-mail: nasibeh.mohammadzadeh@upc.edu)
[2]Faculty of Electrical and Computer Engineering, Tarbiat Modares University, Tehran, Iran, P.O. Box: 14115-111 (e-mail: dorri@modares.ac.ir)
[3]Department of Network Engineering, Polytechnic University of Catalonia, Spain (e-mail: jose.luis.munoz@upc.edu)

Corresponding author: Nasibeh Mohammadzadeh (e-mail: nasibeh.mohammadzadeh@upc.edu).

ABSTRACT 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 article, we propose an architecture for invoice factoring registration based on a public blockchain. To solve the aforemeRUP SEGURETAT DE LA INFORMACIÓ (ISG) Arquitectura software para control de tasa en redes integradas satélite-terrestre Intelligent, Interoperable, Integrative and deployable open source MARKETplace with trusted and secure software tools for incentivising the industry data economy pRIvntioned 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.

INDEX TERMS Public blockchain, smart contract, double factoring, transparency, auditability, privacy, dispute resolution.

I. 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. 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 billions Euros in 2017 [1].

There are several issues and challenges in the traditional invoice factoring proces. For example, it often requires several manual steps and the information is dispersed among different systems and databases [2] [3]. 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 the 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 [4]. 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[1] in this work.

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 blockchain seems a natural tool to solve these issues, because it can keep the record of factoring agreements but also prevent double factoring [5]. A blockchain can make our record-keeping database distributed, highly available but logically unique and secure from manipulations [6]. This way, factoring agreements can be made faster with fewer errors, and still carry the authenticity and credibility of manual contracts.

In this article, we propose 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 *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* 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 [7]) that use distributed storage systems such as IPFS [8] 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 transaction to the blockchain, just 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 [9]. 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, evidences registered in the blockchain and public information are enough to solve disputes without further intervention from the *buyer*.

Finally, we would like to remark that while there are other proposals in the literature, which are discussed and compared in section V, none of them are tailored to our requirements or provide a solution for these requirements as optimal as ours.

The rest of the paper is organized as follows: in section II, we briefly provide the required background; in section III, we describe our proposal and its main assumptions; in section IV, we analyze our architecture from the security perspective; in section V, we present the related work and make a comparison with our proposal; and we finally conclude in section VI.

II. BACKGROUND

A. THE FACTORING PROCESS

A factoring relationship involves three parties [10]: (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) 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*.

[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.

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*.

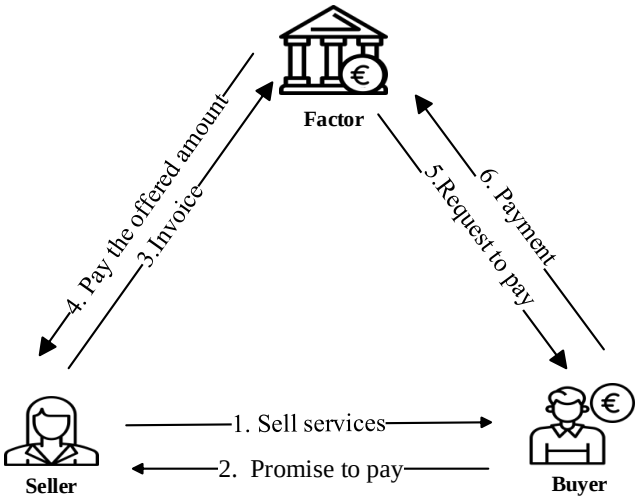

Figure1: Typical factoring.

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 (*double factoring* problem) [4].

B. 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 [11]. 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 [12].

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 [13].

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 [14].

Ethereum [15] is the most popular public blockchain capable of running smart contracts, and the platform of choice for many developers for implementing applications with blockchain [16]. 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 [17].

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 [18].

III. 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. 2):

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 invoice cryptographic digest 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 - 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.

A. 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*.

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 for a registration system, 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 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 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 service/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.

B. 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 1 contains the notation used throughout the paper.

1) Key Management

We use a Diffie-Hellman (DH) key exchange scheme [9] 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

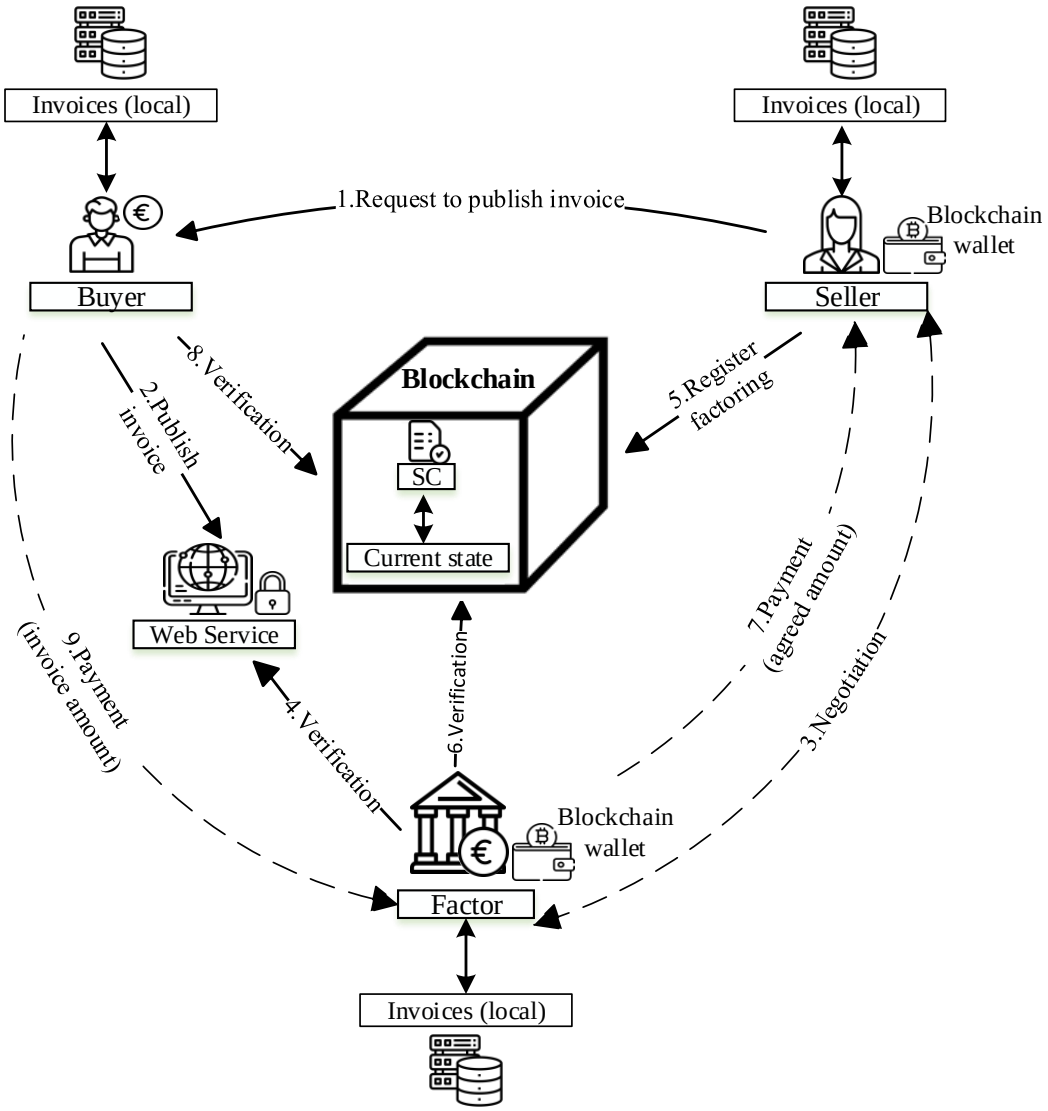

Figure2: Our architecture.

Table1: Notation

Notation	Meaning
KDF(m)	secure symmetric key derived from m (deterministic)
$h(m)$	cryptographic hash of m
ID_A	real identity of entity A
$@A$	blockchain address of entity A
Enc(K, m)	symmetric encryption of m using key K
MAC(K, m)	message authentication code of m using key K (e.g. *HMAC*)
σ_m^A	digital signature over message m using the private key of A
I	invoice number
S	*seller*
B	*buyer*
F	*factor*
C	*smart contract*
WS	*Web Service* (by the *buyer*)

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) [19] 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.

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:

$$K_{AB} = \mathrm{KDF}((g^{x_1})^{x_2})) \tag{1}$$

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

$$P_A = \mathrm{MAC}(K_{AB}, (ID_A, @A)) \tag{2}$$

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*:

$$B \rightarrow WS: \quad P_A, g^{x_2} \tag{3}$$

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):

$$C_A = (P_A, x_1, ID_A, @A) \tag{4}$$

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:

$$C_S^j = (P_S^j, s_1^j, ID_S, @S^j) \tag{5}$$

$$C_F^l = (P_F^l, f_1^l, ID_F, @F^l) \tag{6}$$

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*:

$$B \rightarrow WS: \quad P_S^j, g^{s_2^j}, P_F^l, g^{f_2^l} \quad \forall j, l \tag{7}$$

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) 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): (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.

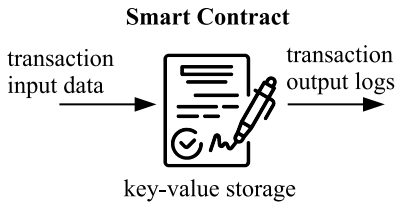

Figure3: Summary of smart contract storage possibilities.

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 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 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) [19], 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.

C. 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. 4 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:

$$m = (C_S, I, g^{i_1}) \tag{8}$$

$$S \rightarrow B : m, \sigma_m^{@S} \tag{9}$$

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:

$$K_{SB} = \text{KDF}((g^{i_1})^{i_2}) \tag{10}$$

$$\mathcal{P}_I = \text{MAC}(K_{SB}, (C_S, I, a_I, d_I, @C)) \tag{11}$$

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

[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).

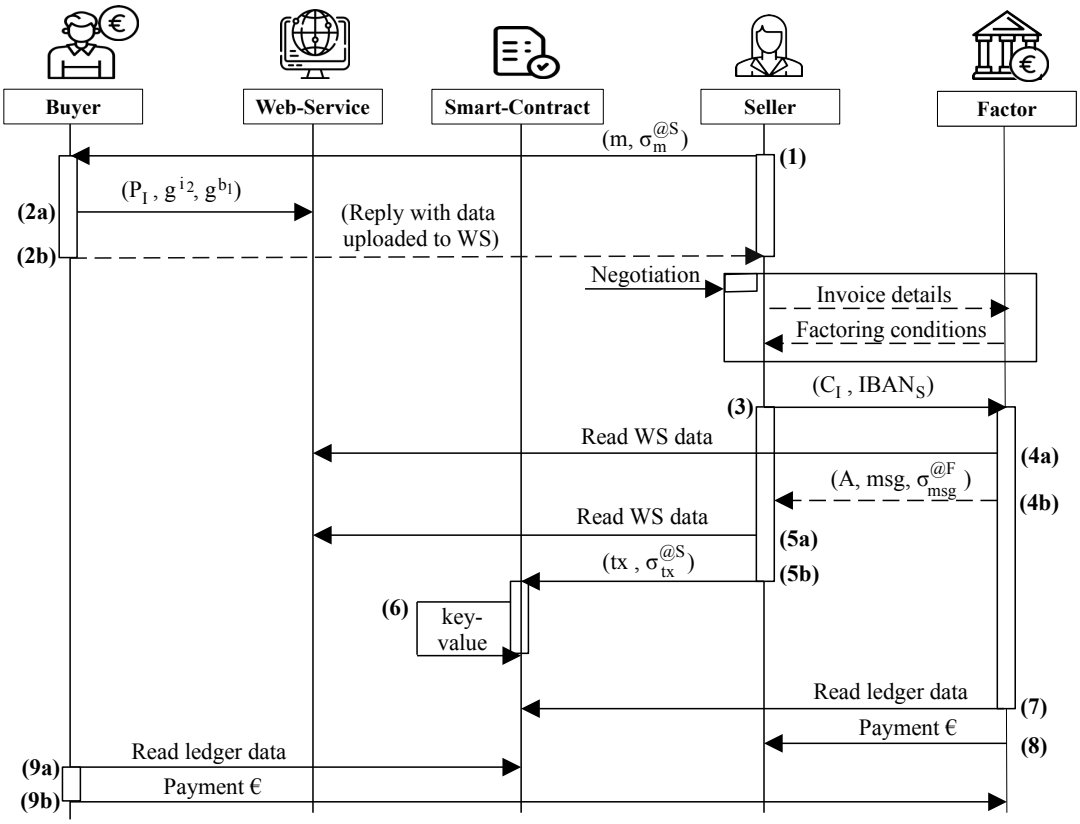

Figure4: Our protocol.

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*:

$$B \rightarrow WS: \quad \mathcal{P}_I, g^{i_2}, g^{b_1} \tag{12}$$

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:

$$\mathcal{C}_I = (\mathcal{P}_I, i_1, C_S, I, a_I, d_I, @C) \tag{13}$$

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* just having DH public values. This protects the privacy of the corresponding *sellers*, *factors* and *invoices*.

D. 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.

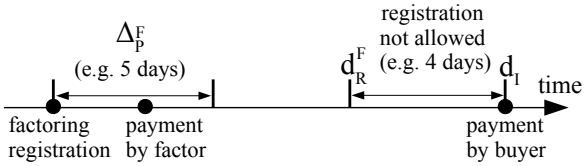

Figure5: Timestamps and periods for factoring registration and payment.

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. 5):

- 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:

$$d_R^F = d_I - 4 \text{ days} \tag{14}$$

  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 but 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*:

$$S \rightarrow F: \quad (\mathcal{C}_I, IBAN_S) \tag{15}$$

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*:

$$K_{FB} = \text{KDF}((g^{b_1})^{b_2}) \tag{16}$$

$$\text{Enc}(K_{FB}, IBAN_F) \tag{17}$$

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

$$\mathcal{A} = (\mathcal{C}_I, C_F, a_F, \Delta_P^F, IBAN_S) \tag{18}$$

The *factor* sends the following data to the *seller*:

$$F \rightarrow S: \quad (\mathcal{A}, msg, \sigma_{msg}^{@F}) \tag{19}$$

where *msg* is defined as follows:

$$msg = (\mathcal{P}_I, d_R^F, \text{Enc}(K_{FB}, IBAN_F), g^{b_2}, h(\mathcal{A})) \tag{20}$$

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*:

$$S \rightarrow C: \quad (tx, \sigma_{tx}^{@S}) \tag{21}$$

where:

$$tx = (msg, \sigma_{msg}^{@F}) \tag{22}$$

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:

$$\texttt{registration.timestamp} \leq d_R^F \tag{23}$$

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}.
- $\text{Enc}(K_{FB}, IBAN_F)$: 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 ($\mathcal{V}$):

$$\mathcal{V} = h(g^{b_2}, Enc(K_{FB}, IBAN_F), @F, h(\mathcal{A})) \tag{24}$$

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

$$\mathcal{H} \Rightarrow \mathcal{V} \tag{25}$$

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:

$$log(\mathcal{H}, g^{b_2}, Enc(K_{FB}, IBAN_F), @F, h(\mathcal{A})) \tag{26}$$

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

E. 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.

IV. 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).

A. 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.

B. 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.

C. 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.

D. 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.

E. 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. [7]).

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. (19) 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* ($\sigma^{@F}_{msg}$) 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. (16)) and $\mathcal{C}_I$ to a judge. Then, the following steps are sufficient to handle the case:

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. (16).) 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 $\text{Enc}(K_{FB}, IBAN_F)$.
8) Resolution: the case can be resolved and the culprit can be detected according to the bank account logs.

V. RELATED WORK

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 2 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 [4]. 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

Table2: Comparison with close related work.

Proposal	*Buyer* tasks	Currency	Responsible for the cost	Factoring negotiation	Availability	Immutability	Privacy
DecReg [4]	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. [7]	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

system using distributed ledger technology [7]. 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 [7] 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 [7], 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 [7] 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.

VI. CONCLUSIONS

In this article, we propose 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 *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.

VII. ACKNOWLEDGMENTS

This research has been supported by TCO-RISEBLOCK (PID2019-110224RB-I00), H2020-i3-MARKET, ARPASAT (TEC2015-70197-R) and 2014-SGR-1504. Also thank Francesc Cubel from the Economics Department of the Generalitat of Catalonia, Marta Bellés from Pompeu Fabra University and Héctor Masip and Rafael Genés from `hardapps.io`.

References

[1] E. Battaiola, F. Massacci, C. N. Ngo, and P. Sterlini, "Blockchain-based invoice factoring: from business requirements

to commitments," in *Proceedings of the Second Distributed Ledger Technology Workshop (DLT@ITASEC)*, ser. CEUR Workshop Proceedings, vol. 2334. CEUR-WS.org, February 2019, pp. 17–31. [Online]. Available: http://ceur-ws.org/Vol-2334/DLTpaper2.pdf
[2] Behalf Company, "5 most common invoice factoring problems," January 2017. [Online]. Available: https://www.behalf.com/merchants/factoring/invoice-factoring-problems/
[3] G. D. Keaton and S. Keaton, "Factoring system and method," Nov. 10 2009, US Patent 7,617,146.
[4] H. Lycklama à Nijeholt, J. Oudejans, and Z. Erkin, "Decreg: A framework for preventing double-financing using blockchain technology," in *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts (BCC)*, New York, NY, USA, 2017, pp. 29–34.
[5] N. Mohamed and J. Al-Jaroodi, "Applying blockchain in industry 4.0 applications," in *Proceedings of IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2019, pp. 0852–0858.
[6] B. K. Mohanta, S. S. Panda, and D. 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)*. IEEE, 2018, pp. 1–4.
[7] M. Guerar, A. Merlo, M. Migliardi, F. Palmieri, and L. Verderame, "A fraud-resilient blockchain-based solution for invoice financing," *IEEE Transactions on Engineering Management*, vol. 67, no. 4, pp. 1086–1098, 2020.
[8] J. Benet, "Ipfs - content addressed, versioned, p2p file system," 07 2014. [Online]. Available: https://arxiv.org/abs/1407.3561
[9] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.
[10] S. Goel, *Financial Services*. PHI Learning Pvt. Ltd., 2011.
[11] I. Bashir, *Mastering blockchain*. Packt Publishing Ltd, 2017.
[12] D. Yaga, P. Mell, N. Roby, and K. Scarfone, "Blockchain technology overview," *arXiv preprint arXiv:1906.11078*, 2019.
[13] A. M. Antonopoulos, *Mastering Bitcoin: unlocking digital cryptocurrencies*. O'Reilly Media, Inc., 2014.
[14] V. Y. Kemmoe, W. Stone, J. Kim, D. Kim, and J. Son, "Recent advances in smart contracts: A technical overview and state of the art," *IEEE Access*, vol. 8, pp. 117 782–117 801, 2020.
[15] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project White Paper*, vol. 151, no. 2014, pp. 1–32, 2014.
[16] H. Chen, M. Pendleton, L. Njilla, and S. Xu, "A survey on ethereum systems security: Vulnerabilities, attacks, and defenses," *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, pp. 1–43, 2020.
[17] S. Rouhani and R. Deters, "Security, performance, and applications of smart contracts: A systematic survey," *IEEE Access*, vol. 7, pp. 50 759–50 779, 2019.
[18] J. Liu and Z. Liu, "A survey on security verification of blockchain smart contracts," *IEEE Access*, vol. 7, pp. 77 894–77 904, 2019.
[19] S. SEC, "1: Elliptic curve cryptography, version 2.0," *Standards for Efficient Cryptography Group*, 2009.

NASIBEH MOHAMMADZADEH, is currently a Ph.D. student of the Information Security Group (ISG). She is doing research on invoice factoring through Blockchain Technology at Universitat Politècnica de Catalunya (UPC). She obtained a Master's degree in information technology from the University of Hyderabad, India, where she focused on network security in cellular networks, which was implemented at the Institute for Development and Research in Banking Technology (IDRBT) of India.

SADEGH DORRI NOGOORANI, is an assistant professor and the head of the Blockchain Laboratory at Electrical and Computer Engineering Faculty of Tarbiat Modares University. He holds an M.S. in Computer Networks, and a Ph.D. in Computer Engineering from Sharif University of Technology. His research focus is on security and privacy in distributed ledger technologies and applications, and he is also interested in security and privacy in other distributed software systems, trust management, and risk management in computer systems.

JOSÉ L. MUÑOZ-TAPIA, is a researcher of the Information Security Group (ISG) and an associate professor of the Department of Network Engineering of the Universitat Politècnica de Catalunya (UPC). He holds an M.S. in Telecommunications Engineering (1999) and a PhD in Security Engineering (2003). He has worked in applied cryptography, network security and game theory models applied to networks and simulators. His research focus has now turned to distributed ledgers technologies and he is the director of the master's program in Blockchain technologies at UPC School.