DecReg: A Framework for Preventing Double-Financing using Blockchain Technology

Hidde Lycklama à Nijeholt, Joris Oudejans and Zekeriya Erkin*
Cyber Security Group, Department of Intelligent Systems, Delft University of Technology
Mekelweg 4, 2628 CD, Delft, The Netherlands
mail.hidde@gmail.com, joris.oudejans@gmail.com, z.erkin@tudelft.nl

ABSTRACT

Factoring is an important financial instrument for SMEs to solve liquidity problems, where the invoice is cashed to avoid late buyer payments. Unfortunately, this business model is risky as it relies on human interaction and involved actors (factors in particular) suffer from information asymmetry. One of the risks involved is 'double-financing': the event that an SME extracts funds from multiple factors. To reduce this asymmetry and increase the scalability of this important instrument, we propose a framework, DecReg, based on blockchain technology. We provide the protocols designed for this framework and present performance analysis. This framework will be deployed in practice as of February 2017 in the Netherlands.

Keywords

Double Financing, distributed trust, blockchain.

1. INTRODUCTION

To conquer liquidity issues in small to medium enterprises (SMEs), a model known as factoring[22] is used. The time between sending an invoice and getting paid for that invoice can be long and unpredictable. For example, in the Netherlands in 2015, the average contractual payment time for business-to-business transactions was 23 days while the actual payment time was 29 days[17]. This uncertainty poses a real threat for companies when they need the owed money for e.g. making investments or compensating setbacks. In factoring, the stream of money does not go directly from the debtor, hereafter buyer, to the supplier, hereafter seller. Instead, the factoring service provider, FSP hereafter, buys the invoice from the seller for a portion of the original price, e.g. 95%. This way, the FSP takes over the risk from the seller, who has no uncertainty and immediately materializes the invoice's value.

Unfortunately, factoring gives rise to a serious problem: A seller can cash an invoice at multiple FSPs to double the amount of money received since there is no insight between factors in whether an invoice is already financed. In a similar scenario, a seller has a long-term agreement with a bank to sell all his invoices and in return get a loan. If the seller is in need of liquidity and factors an invoice, a dispute can arise between the bank and the other FSP. Because a buyer will most likely not be willing to pay the same invoice twice, at least one of the FSPs will be disadvantaged. We define this problem as Double-Financing, as briefly mentioned in [8], because that resembles the name of its counterpart in digital cash schemes as Double-Spending in Bitcoin[23].

A possible approach to solve the Double-Financing problem would be to create a central database with all agreements between FSPs and sellers, where each FSP could check whether a proposed invoice is still owned by the seller or not. However, having a central database requires a fully trusted, secure, scalable, easily accessible server with no down-time. In practice, maintaining such a system is costly. Moreover, since there are many FSPs with trust issues, keeping a central database is challenging in practice.

Our proposal to solve Double-Financing is to use an instrument that meets all the fore-mentioned requirements: A blockchain[23]. Although the technology is most widely known for Bitcoin and therefore financial transactions, blockchain provides a basis for solving distributed trust related problems. In short, a blockchain is an immutable, decentralized, public ledger that relies on a consensus algorithm to decide which data is appended. The immutability means that it is possible to store data on a blockchain such that it is unfeasible to ever change the data or remove it, relying on cryptography. And, equally important, every authorized person can query this ledger from anywhere in the world, resembling a decentralized database. Therefore, instead of storing every agreement in a central database, our proposal is to write them onto a blockchain. Then, in any future payment request by a seller, the FSP can check the blockchain for a previous agreement which involves the invoice. If such an agreement exists, the FSP knows the invoice was already financed and denies the request.

Although financial institutions all over the world[3] are highly interested in blockchain technology due to its potential, as far as we know, no significant applications have been put to the test in the world of invoice-financing. Our proposal for double financing, namely DecReg, shows us a

*H. Lycklama à Nijeholt and J. Oudejans conducted this research as a part of their Bachelor Honour Programme and partly supported by Innopay BV during their internship.

BCC'17, April 02 2017, Abu Dhabi, United Arab Emirates

ACM ISBN 978-1-4503-4974-1/17/04…$15.00

DOI: http://dx.doi.org/10.1145/3055518.3055529

blockchain application that leverages the inherent advantages of the technology and that can actually be implemented. Furthermore, although in China[24] a central registry is being used for invoices, a decentralized solution to the problem of Double-Financing has not yet been brought into practice in the factoring business. Therefore, our solution is the first of its kind with a significant impact in the invoice-financing domain.

In this paper, we describe the DecReg protocol that prevents Double-Financing of invoices in factoring. Our proposal relies on blockchain technology and prescribes accepted mutations on its data, thus does not need any trusted entity to handle this data. Since all involved parties have a copy of all the transactions, the verification procedure is fast and reliable. Furthermore, we provide an implementation of DecReg on a private blockchain run by FSPs. The consensus algorithm relies on the Tendermint[18] BFT protocol[19], which has demonstrated to have the sufficient performance needed for the factoring environment in the Netherlands. Furthermore, our protocol is planned to be deployed by industry in the Netherlands starting as of February 2017.

2. BACKGROUND

2.1 Blockchain

A blockchain is a distributed database where new data can only be appended to previous data. It consists of blocks where each block contains a list of transactions and a hash pointer to the previous block, forming a chain. Hence, it is only possible to add a block to the end of the chain. The content of a new block that will be appended is decided via a consensus protocol. The combination of the blockchain data structure and a networking protocol as a means to reach consensus in a distributed network was first introduced in Bitcoin by Nakamoto[23]. In Bitcoin, a Proof-of-Work[6] algorithm is used to reach consensus, relying on the assumption that computational power for hashing is a scarce resource. Nodes in the network communicate via a peer-to-peer protocol, mainly by transferring transactions and blocks between each other.

A blockchain can be defined in a more abstract way. A blockchain is a distributed application with a current state, like a state machine. The state is defined by the transactions that exist. The state is propagated to the nodes in the network via a consensus protocol. We will now go into both components.

2.2 Transactions

Transactions are used to mutate the state of a blockchain. The kinds of transactions that are valid in a given blockchain, along with their rules, define how the state can be changed and what meaning can be derived from the state. Therefore, the transactions can be seen as the application business logic of a blockchain application. Furthermore, transactions can be designed in such a way that considerably complex applications can be designed for a blockchain. Such complex applications are often referred to as smart contracts. In the Ethereum blockchain, smart contracts are defined in a Turing-complete virtual machine language, providing the possibility to create complex, distributed applications[15]. As data can only be appended, these transactions are the only way to change this state.

2.3 Consensus algorithm

A consensus algorithm is a protocol to replicate the state of a blockchain over multiple nodes in a network. A key property of any blockchain's consensus algorithm is that it must be Byzantine fault tolerant[21]. That is, an application remains operable even when some subset of the network consists of faulty, or Byzantine, nodes. Therefore, it must not rely on other nodes, such as on their uptime or functioning. Due to the data structure of a blockchain and its corresponding consensus algorithm, a blockchain can be Byzantine fault tolerant. The first and most prevalent consensus algorithm in current blockchains is a Proof-of-Work-based algorithm, first introduced by Back[6]. For instance in Bitcoin, the Proof-of-Work algorithm is used to accept the next block in the chain. In the Bitcoin network, miner nodes search for an acceptable block hash. The block is mined when the resulting hash is below a certain threshold, i.e. it has a certain number of leading zeros. As the probability of finding a nonce grows linearly with the amount of computing power, the state of the blockchain is decided not by the majority of nodes, but by the majority of computational capacity. The Proof-of-work algorithm is a reliable consensus algorithm, as it is easier for an adversary to get the majority of nodes, but not the majority of computing power. However, the algorithm is often criticized, as it requires significant resources. For instance, the Bitcoin network takes up 350 Mega Watt as of March 2016[11].

2.4 Double-Spending

An inherent property of blockchain is that it prevents the double spending of entities in an individual chain. For a cryptocurrency without centralized clearing of transactions, this is an important property. However, we shall later see that this property is useful for other applications, such as registering real-world entities. In Bitcoin, transactions indicate the transfer of BTC between Bitcoin addresses, derivations between respective public keys[2]. The new transaction references a previous transaction to the address, proving the possession of the amount of cryptocurrency. Due to the consensus protocol of Bitcoin, the transaction is picked up by a miner and eventually stored in a block. The state of the Bitcoin network, the amount of BTC every Bitcoin address holds, is mutated after the transaction is processed. Because miners validate the incoming transactions, it is statistically unlikely that a block containing an invalid transaction is appended, because the miners do not include an invalid transaction in a block. If a miner would include an invalid transaction and broadcasts the block with the solved hash, other nodes would not accept this block into the new state and the miner would thus not receive its reward.

2.5 Tendermint

Tendermint[9] is a consensus algorithm based on the algorithm proposed by Dwork et al. [13], solving the Byzantine Generals Problem[21]. The Tendermint consensus algorithm relies on chosen validator nodes voting on valid blocks[19]. Depending on the configuration, validator nodes can be expected to have put in some form of collateral for their position, to incentivize non-Byzantine behavior. This stake can be some form of cryptocurrency in the network, but can also be a physical asset. Blockchain implementations like Bitcoin[23] and Ethereum[15] have monolithic designs, where the state machine and consensus layers are contained in a

single application. However, Tendermint decouples the two, and only implements the consensus protocol. Therefore, the state transition machine and the content of the transactions can be arbitrary.

The consensus protocol makes use of one or more rounds that get repeated when an invalid action happens or when a timeout occurs. Each round has three steps, *Propose, Prevote* and *Precommit*, along with two special steps *NewHeight* and *Commit*. In an ideal situation, the consensus process starts at the *NewHeight* step and ends up in the *Commit* step after undergoing one round. Sometimes more rounds are required in order to successfully commit a block. This can be due to several reasons, such as the unavailability of certain nodes that the protocol relies on in that round. For example, the protocol requires a designated *proposer* node to propose a block in that round. Some problems are resolved by moving to the next round and proposer node, while others are resolved by increasing certain protocol timeouts. A more detailed explanation of protocol can be found at the Tendermint Wiki[19]. Tendermint nodes communicate with each other using an authenticated encryption scheme[20], preventing the eavesdropping of other parties. We shall see that this property, combined with the protocols public key infrastructure, prevents a man-in-the-middle attack[16].

The protocol ensures that at least 2/3 of the set of validator nodes decides on the validity of a block to be appended to the chain. The protocol is Byzantine fault tolerant in that it is able to do reliable state replication when up to 1/3 of the validator nodes is byzantine.

3. DecReg: A DOUBLE-FINANCE PREVENTION FRAMEWORK

DecReg, short for Decentralized Registry, is a framework composed of a dedicated blockchain, a certified network of nodes and three protocols to be executed on the blockchain. The consensus engine is Tendermint[18]. On top of that the protocols are executed on the specified transaction structure.

When an FSP receives an invoice finance request in the form of an invoice, it has to verify the absence of a previous agreement. To accomplish this, it hashes the specified data fields of the invoice and queries the blockchain. If the hash is present, a double finance attempt is detected and the FSP aborts the deal. If not, the FSP notarizes his agreement so other FSPs can detect future double finance attempts.

3.1 Transactions

The DecReg blockchain contains transactions that represent agreements between finance service providers and sellers. Moreover, the blockchain contains transactions that are used to manage the network, such as transactions to indicate the addition or removal of participants in the network. Agreement transactions can be of two types, namely (1) a single-invoice agreement or (2) a long-term agreement. Distinguishing between these two is important in order to understand the described protocols. Furthermore, both of these transaction types can have a Revoke-flag, indicating cancellation of an agreement or a single-invoice exemption from a long-term agreement. Naturally, a revoke transaction can only be done by the same node that did the initial notarize transaction. This implies an ownership of the notarized agreement by the registering node.

Table 1: Protocols and their arguments

	Single-invoice	**Long-term**
Verify	$(id_s, id_{inv}, ts_{inv})$	(id_{inv}, ts_{start})
Notarize	(id_s, id_{inv})	$(id_s, ts_{start}, ts_{end})$
Revoke	(id_s, id_{inv})	(id_s)

3.2 Network

The blockchain is maintained by a network consisting of certified nodes with each node representing a separate finance service providing party. The network is governed by a Central Authority (CA), which certifies nodes to participate in the network. Only nodes that are certified by the CA are able to communicate with other nodes. Therefore, the network is only available to the participating parties. The CA role can be placed with an existing institution or an institution can be specifically created for this purpose. Only with a certificate issued by the CA will the transactions broadcasted by a node be accepted by the other nodes. The certification of nodes also has the advantage of greatly reducing the chance of a successful Sybil attack[12], on which we will elaborate in Section 3.4.

3.3 Protocols

DecReg is composed of three protocols: Verifying, notarizing and revoking. The protocol for verifying is executed by nodes upon receiving a finance request by a seller to detect a possible Double-Finance attempt. Notarizing is the phase in which nodes notify all other nodes of a new agreement with a seller. Finally, a node notifies all other nodes by revoking after aborting an agreement or giving a single invoice exemption from a long-term agreement. We define the FSP as f with secret key sk_f, the seller as s with id id_s, the invoice as inv with id id_{inv} and timestamp ts_{inv}, the timestamp of the start of a long-term agreement as ts_{start} and the end of it as ts_{end} as summarized in Table 1.

Verify On receiving a finance request from a client seller an FSP queries the blockchain to verify the absence of a previous agreement. As there are two different types of agreements, we provide two different queries, one for each type. The verification algorithm for long-term agreements is given in Algorithm 1 and the one for single-invoice agreements is given in Algorithm 2.

Let us first consider the algorithm for verifying the absence of long-term agreements as it is referenced in the algorithm for verifying single-invoice agreements. The algorithm queries the blockchain for a long-term agreement transaction matching id_{inv} and where ts_{end} is greater than the specified start date. If such a transaction is found, the algorithm checks if the Revoke flag is set to true. If it is revoked, a long-term agreement has apparently been canceled and the algorithm accepts. If not, a possible fraud is detected and the algorithm rejects. If no transaction is found, there have not been overlapping agreements concerning the same seller and the algorithm accepts.

Algorithm 2 is similar to Algorithm 1. First, the algorithm queries the blockchain for a previous agreement matching id_s and id_{inv}. If there is such a transaction, the algorithm checks whether the Revoke flag is set to true. If it is, the found agreement is either a revoked single invoice or an exemption from a long-term agreement. In both cases, the

Algorithm 1 Verify long-term

```
procedure VERIFY(id_s, ts_begin)
    r ← QUERY(id_s, ts_begin)        ▷ Query blockchain for a previous agreement a where a.ts_end > ts_begin
    if r ≠ undefined then
        if r.revoked = true then
            OUTPUT('Revoked long-term agreement')
            accept
        else
            OUTPUT('Long-term agreement already exists')
            reject
        end if
    else
        OUTPUT('No previous long-term agreement')
        accept
    end if
end procedure
```

Algorithm 2 Verify single-invoice

```
procedure VERIFY(id_s, id_inv, ts_invoice)
    r ← QUERY(id_s, id_inv)
    if r ≠ undefined then
        if r.revoked = true then
            OUTPUT('Revoked or exempted invoice agreement')
            accept
        else
            OUTPUT('Single-invoice agreement already exists')
            reject
        end if
    else
        if VERIFY(id_s, ts_invoice) then
            OUTPUT('No previous agreements')
            accept
        else
            reject
        end if
    end if
end procedure
```

algorithm accepts. If it is not revoked, the invoice took part in a previously made agreement and thus cannot be part of a new one, so the algorithm rejects. If no transaction is found, the invoice can still be part of a long-term agreement, so Algorithm 2 calls Algorithm 1 with id_s and $ts_{invoice}$. It accepts if there is no long-term agreement overlapping with the invoice date, otherwise it rejects.

Notarize After verifying the absence of any previous agreement, an FSP notarizes his agreement with a seller on the blockchain. It does so by creating a transaction using either (id_s, id_{inv}) for a single invoice or $(id_s, ts_{start}, ts_{end})$ for a long-term agreement. It signs the transaction with sk_f and broadcasts it to the network.

Revoke For a blockchain can only be appended to, canceling an agreement must be done by making a new transaction. In the simple cases of revoking previously made single-invoice- or long-term-agreements, one creates a transactions using respectively (id_s, id_{inv}) and (id_s) with the Revoke flag set. This will make previous agreements notarized by the same FSP invalid. There is also the special case where an FSP has a long-term agreement with a seller but wants to make an exemption for an invoice that does not meet the contractual requirements of the long-term agreement. In that case, the FSP broadcasts a transactions using (id_s, id_{inv}) and with the Revoke flag set to true, where id_s is a seller the FSP has a long-term agreement with.

3.4 Consensus algorithm

Whenever a new transaction is broadcasted to the network consensus must be reached on the new state of the blockchain. As explained in Chapter 2.3, the first blockchain implementations like Bitcoin used Proof-of-Work to achieve consensus. But in our case participating nodes need to be certified, so a Sybil attack[12] becomes a minimal threat. We can rely on voting per cryptographic identity as they are only valid when the identity is approved by the CA. Our consensus algorithm can therefore depend on a One-certificate-one-vote decision scheme to reach consensus after each new transaction. To implement this functionality, a version of the Tendermint protocol is used. The network is configured such that each factoring node will become a validator with an equal vote. The collateral of each validator node is an intangible asset: If factors choose to misbehave, they can easily be excluded in the network.

3.5 Propagation time

As with most decentralized consensus protocols, it takes time for a transaction to propagate through the network and be irreversible. Only when the transaction is part of a block that is committed to the chain, the transaction is final. Therefore, an FSP can not be certain his notarization has succeeded upon broadcasting a transaction. In the default configuration, the Tendermint consensus protocol commits a new block to the chain one second after the previous block. Depending on several factors, such as the network latency and amount of faulty nodes, the block time has a variable length. However, in the network set-up that we described, the block time would typically not exceed several seconds. For the factoring entities, this notarization time is acceptable. Moreover, note that this regards an invoice notarization and not an invoice lookup. Reading the blockchain data for an invoice is fast, because it only queries data that the node has stored in local storage.

4. IMPLEMENTATION

The implementation is written as a Go[1] application. The application interacts with a configured Tendermint node via the Tendermint Socket Protocol [5]. In this section we explain the characteristics and specifications of this implementation.

4.1 Network

To bootstrap the network, the Certificate Authority is created by generating a key-pair and certificate. Upon entry, each node generates an Ed25519[7] key-pair sk_f and pk_f to use as their persistent identity in the network. Ed25519 is a public-key signature system that is designed to have fast signing, verification and key generation, while not sacrificing security. Furthermore, the keys and signatures can be compressed to be only 32 and 64 bytes, respectively. This makes it an efficient system to use.

The Tendermint consensus protocol also generates a key-pair for secure peer-to-peer communication, so this key-pair

can be shared [20]. Next, if the corresponding FSP has met the CA's requirements, the nodes identity is certified by the CA. Each node is configured with the CA's identity and an initial list of other peers. This list, called the genesis certified node set, contains a list of certified nodes with their certificate and, optionally, the nodes' Internet address.

Each transaction in the network contains a signature from a node that is in the set of certified nodes. However, the set of certified nodes is subject to change as FSPs may enter or leave the network. The changes to this set are recorded in the blockchain via a *Validator set* transaction. Therefore, the set of certified nodes is the genesis certified node set, with mutations from the *Validator set* transactions in the blockchain.

4.2 Transactions

The different functions of transactions define the business logic of DecReg. A transaction can be of three different types, each with a different type of payload data. The *version* field is used to indicate for which protocol version the transaction message is formatted so nodes can respond appropriately. The *version* field helps with backwards compatibility when updates are made to the protocol in a network that is already operating. The *signature* field contains the signature of the node creating the transaction. The *signature* is constructed from all the transaction fields concatenated in the same ordering of the fields in the transaction, with the exception of the *signature* field itself. The *signature* identifies the creator of the transaction, which improves the accountability of the nodes in the network. The *payload length* field is used to indicate the length in bytes of the *payload* field, which can be a maximum of $2^{16} - 1$ bytes. The *payload* can contain arbitrary data, depending on the type of the transaction. The *type* field specifies the type of the transaction. For some types, the first byte of the payload is used to store transaction flags. At this time, only the *Revoke* flag exists that is used to indicate the reversal of a notarization. As of now, a transaction can have one of the following three types:

1. **Validator set** Indicates a change in the current validator set. This transaction is used to change the set of certified nodes in the network. The payload of this transaction contains the public keys of the changing validators. Per public key, the transaction has either a valid certificate or a revocation of the public key, depending on the validator entering or exiting the network. The transactions are used to determine the validity of a peer. The total payload size is dependent on the size of the changing validator set.

2. **Single invoice** Represents the registration of a single-invoice in the network. The payload contains the flag byte, the timestamp of 8 bytes of the current time, and a reference to when the invoice was notarized. Next, it contains the SHA-256 hash of the seller and invoice identifiers, id_s and id_{inv}, respectively as such: $hash(id_s||id_{inv})$. The total payload length is 73 bytes.

3. **Long-term agreement** Signifies the existence of a long-term agreement. The payload consists of the flag byte, the seller identifier id_s, as well as two timestamps, ts_{begin} and ts_{end}. The total payload is at least 17 bytes, depending on the size of the seller identifier.

4.3 Protocols

The ability to uniquely identify sellers and their invoices is essential in DecReg, as otherwise unjust collisions can occur. For id_s we use a combination of a ISO [14] country code and the Chamber of Commerce number. For id_{inv} we use the invoice reference as defined by the seller, as these have to be unique by law[25] in the Netherlands. Combined, id_s and id_{inv} are thus always unique.

Verify As our transaction data is hashed, Algorithm 1 and 2 needs to be executed precisely. The timestamps are not part of the hash, so whenever a timestamp comparison is needed, a node queries the hash first. If a transaction is found, the timestamps can be compared. If the comparison evaluates to false, we proceed querying for the hash.

Notarize Our implementation follows the protocol as described in the previous section, in which the node broadcasts the transaction to all of the nodes known to him.

Revoke We use a separate type for each transaction, e.g. a different type for the single-invoice notarize transaction and revoke transaction. The data fields follow the protocol description as described in the previous section.

Test set-up Our implementation of DecReg is deployed in association with a group of FSPs, as well as a governing authority. Each FSP runs a node with the application implementation. This node is certified by the governing authority, which acts as the network's certificate authority. Each node communicates with its peers and broadcasts all factoring agreements to the network. Furthermore, the FSPs use the network to verify that a single-invoice or long-term agreement does not yet exist for their invoice or customer. There are around 30 FSPs in the Netherlands, almost all participate, so there are 30 nodes (*validators*). In the Netherlands, according to Tim Zoete, Voldaan factoring, about one thousand single invoice agreements are made per day, each represented in the network with one transaction. Of the long-term agreements about a thousand per year are issued, so these are negligible in our analysis.

5. DISCUSSION AND CONCLUSIONS

Although one of blockchain's core principles is its public nature, the information stored in our blockchain is too sensitive to be public. Therefore, we have chosen to use a private blockchain for DecReg. The CA governs access to this private blockchain, preventing any unauthorized parties to obtain the sensitive information. One can argue the CA is a central institution, thus centralizing the blockchain and eliminating the advantages of using a blockchain. However, the only power the CA has is deciding who enters and leaves the network, it can not alter or contaminate any stored data. To our knowledge, the only feasible attack possible would be a collaboration between the CA and a seller, where the CA denies an FSP from accessing the network. On receiving an invoice finance request from this seller, the FSP will not be able to verify any previously made agreement, making him vulnerable to Double-Financing. In order to solve this problem, an FSP should halt invoice financing until it regains access to the blockchain network.

As described by Tendermint, the network can be self-governing: "Tendermint supports dynamic membership safely by requiring a $+2/3$ quorum of validators to approve of membership changes"[4]. This would mean an efficiency improvement and can decrease the costs of governance. However,

this would require a majority of the nodes to be always perfectly able to decide on nodes entering or leaving. As all nodes are competitors of each other, it is decided on the CA to govern the network and thus prevent incentive conflicts.

For our proposal, further optimizations can be made. As invoices have expiration dates, the verification protocol does not need to look for single invoices before a certain date in the past. Invoices have a limited lifespan. Generally, invoices do not have a payment term longer than 30 days and are payed in extreme cases in 90 days. Therefore, we could define a mechanism that prunes the blockchain data after a certain cut-off date. This way, the size of the blockchain is in connection with the amount of financed invoices in the given period and not with all financed invoices since the genesis of the blockchain. Furthermore, the search queries would be faster, due to the set of transactions being smaller.

In the following, we also address two issues that provide ground for future work.

Information exposure In DecReg, FSPs publish every single agreement they have with a seller to all their competitors. With this data, it is possible for an FSP to enumerate operational information about competing FSPs. It is possible to prevent the exposure of such information via the blockchain, while preserving the current functionality. Several group signature schemes[10] provide a solution to this problem. By using a group signature scheme in the network, nodes would see that one of the certified nodes has signed a transaction but they cannot identify which node. Due to the traceability property, a CA is able to see the identity of the signer in case of malicious behavior.

Invoice identifier uniqueness Although Dutch law requires invoice numbers to be unique, in practice sellers easily make mistakes. Factoring is mostly used by SMEs, so setting the invoice number is often a manual job. For DecReg to work, FSPs need to require correctly formatted invoice numbers from their customers, the sellers. Otherwise, Double-Finance attempts could be detected that actually concern different invoices that have the same number. Therefore, use of unique invoice numbers should be enforced.

In this paper, we presented a framework that deploys the blockchain technology to solve the double-factoring problem in finance. Our protocols that are designed to support the business logic in factoring are effectively realized in our blockchain variant and shown to be efficient to be used in practice. In fact, our framework presented in this paper is being deployed to be used in the financial industry as of February 2017 in the Netherlands.

6. REFERENCES

[1] Anonymous. The go programming language. https://golang.org/, 2009.

[2] Anonymous. Bitcoin wiki. https://en.bitcoin.it/wiki/Technical_background_of_version_1_Bitcoin_addresses, 2017. Accessed: 1 Jan 2017.

[3] Anonymous. R3 consortium website. http://www.r3cev.com/, 2017. Accessed: 17 Jan 2017].

[4] Anonymous. Tenderming vs PBFT. https://tendermint.com/blog/tendermint-vs-pbft, 2017. Accessed: 14 Jan 2017.

[5] Anonymous. Tendermint ABCI overview. https://tendermint.com/intro/abci-overview, 2017. Accessed: 15 Jan 2017.

[6] A. Back. Hashcash-a denial of service counter-measure. http://www.hashcash.org/papers/hashcash.pdf, 2002.

[7] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, 2012.

[8] C. Bryant and E. Camerinelli. Supply chain finance. 2012.

[9] E. Buchman. Introduction to tendermint. https://github.com/tendermint/tendermint/wiki/Introduction, 2017. Accessed: 15 Jan 2017.

[10] D. Chaum and E. Van Heyst. Group signatures. In *Workshop on the Theory and Application of Cryptographic Techniques*, pages 257–265. Springer, 1991.

[11] S. Deetman. Bitcoin could consume as much electricity as Denmark by 2020. http://motherboard.vice.com/read/bitcoin-could-consume-as-much-electricity-as-denmark-by-2020, 2016.

[12] J. J. Douceur. The sybil attack. In *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, January 2002.

[13] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988. section 4, algorithm 2'.

[14] I. O. for Standardization. Iso 3166-1 alpha-3, 1974.

[15] E. Foundation. White paper. https://github.com/ethereum/wiki/wiki/White-Paper, 2017. Accessed: 13 Jan 2017.

[16] Y. Joshi, D. Das, and S. Saha. Mitigating man in the middle attack over secure sockets layer. In *IEEE International Conference on Internet Multimedia Services Architecture and Applications (IMSAA)*, pages 1–5. IEEE, 2009.

[17] I. Justitia. European payment report. page 14, 2015.

[18] J. Kwon. Tendermint: Consensus without mining. http://www.the-blockchain.com/docs/Tendermint%20Consensus%20without%20Mining.pdf, 2014.

[19] J. Kwon. Byzantine consensus algorithm. https://github.com/tendermint/tendermint/wiki/Byzantine-Consensus-Algorithm, 2017. Accessed: 15 Jan 2017.

[20] J. Kwon. P2P authenticated encryption. https://github.com/tendermint/tendermint/wiki/P2P-Authenticated-Encryption, 2017. Accessed 15-January-2017.

[21] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[22] C. G. Moore. Factoring-a unique and important form of financing and service. *The Business Lawyer*, 14(3):703–727, 1959.

[23] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[24] H. R. Randau and O. Medinskaya. Types of taxes. In *China Business 2.0*, pages 125–137. Springer, 2015.

[25] R. van State. Wet van de omzetbelasting. artikel 35a, lid 1, sub b, 1968.