

A Fraud-resilient, Blockchain-based solution for Invoice Financing

Meriem Guerar, Mauro Migliardi, Luca Verderame, Francesco Palmieri, and Alessio Merlo

Abstract

Invoice financing has been a steadily growing component of the financing market as a whole for the last few years and, in 2016, it has grown to be the third largest financing market. Nonetheless, the risk of frauds is still very acute and most solutions proposed so far are based on private, proprietary platforms that cannot match the global nature of such a market. Even the most recent proposals based on blockchain are mainly adopting a private, permissioned blockchain. In this paper, we propose a platform based on a public blockchain allowing both fully open and group-restricted auctioning of invoices. Furthermore, our platform introduces a reputation system based on the past behavior of entities, as it is photographed by the global ledger that is the blockchain, that allows insurance companies modulating the cost of the insurance contracts they offer. This combination guarantees the complete transparency and tamperproof-ness of a public blockchain, while it allows reducing insurance costs and fraud possibilities.

Index Terms

Blockchain, Ethereum, Smart contract, Auction, Invoice factoring, IPFS

I. INTRODUCTION

Cash flow is a major source of complexity in the daily life of companies. The delay between the invoice date and its actual payment may represent a major challenge for any enterprise but even more so for Small to Medium Enterprises (SMEs). To tackle this problem a common practice is to leverage some mechanism of invoice financing such as invoice factoring, in fact, this allows SMEs to ease the cash flow even if it has a cost. The process of factoring can be described as follows: an enterprise sells the invoice to a factoring company (which quite often is a financial institution such as a bank) for an immediate payment of an agreed percentage of the invoice amount, the financial institution then gets the payment of the full amount of the invoice from the original buyer on the due date. This process both helps the enterprises in steadying their cash flow and it represents a significant source of income for the financial institutions, yet, it exposes the factoring companies to serious fraud risks mainly because of the lack of a unified view of all of the instances of invoice factoring. As an example, a well known invoice factoring fraud risk is the case of double financing; this happens when the enterprise sells the same invoice to more than one financial institution leveraging the lack of a common, unified view of the field among financial institutions. Obviously, the buyer will pay the invoice just once, paying only one institution and leaving the other(s) unpaid. Another significant source of risk is the reliability of the original buyer. The financial institution has very little to no information about the original buyer, hence it has no way to know in advance if there is the chance that the original buyer will refuse to pay the invoice on the due date.

Another significant risk is the Seller knowingly submitting false, modified invoices with the intent to commit a fraud, either acting alone or in collusion with the Buyer. A solution might be to add risk insurance to refund the Investor; however, in the absence of significant countermeasures aiming at reducing the fraud opportunity, the cost of such an insurance will make the whole operation economically unfeasible. Hence, the simple addition of an insurance is not considered a viable solution.

The unified vision that is needed could be provided by means of an invoice financing platform with a centralized database of all the invoices, the enterprises, the buyers and the instances of factoring.

However, centralized systems can be expensive, they are a single point of failure, and they are prone to privacy infringement, data manipulation and attacks which may make them unreliable and untrustworthy. Furthermore, storing all invoice data in a public database raises issues of confidentiality.

With the raising of blockchain technology and smart contracts, we argue that we no longer have to rely on centralized systems. Blockchain technology may be used to implement a tamper-proof, trusted, and decentralized ledger [1] that relies on a consensus algorithm to decide which data is appended [2].

In this paper, we propose an auction-based invoice financing solution that leverages the InterPlanetary File System (IPFS) [3] and the Ethereum blockchain [4]. The invoice data is stored on the IPFS to remove confidentiality issues while its corresponding IPFS hash is stored into a blockchain smart contract in order to ensure integrity, traceability and authenticity of the invoice. Moreover, we introduce into the proposed solution a reputation system which contributes to identify and quench the risk of frauds. The paper is structured as follows: in section II we introduce our invoice financing solution; in Section III we present the implementation and testing details; in Section V we present related work. Finally, Section VI concludes this paper.

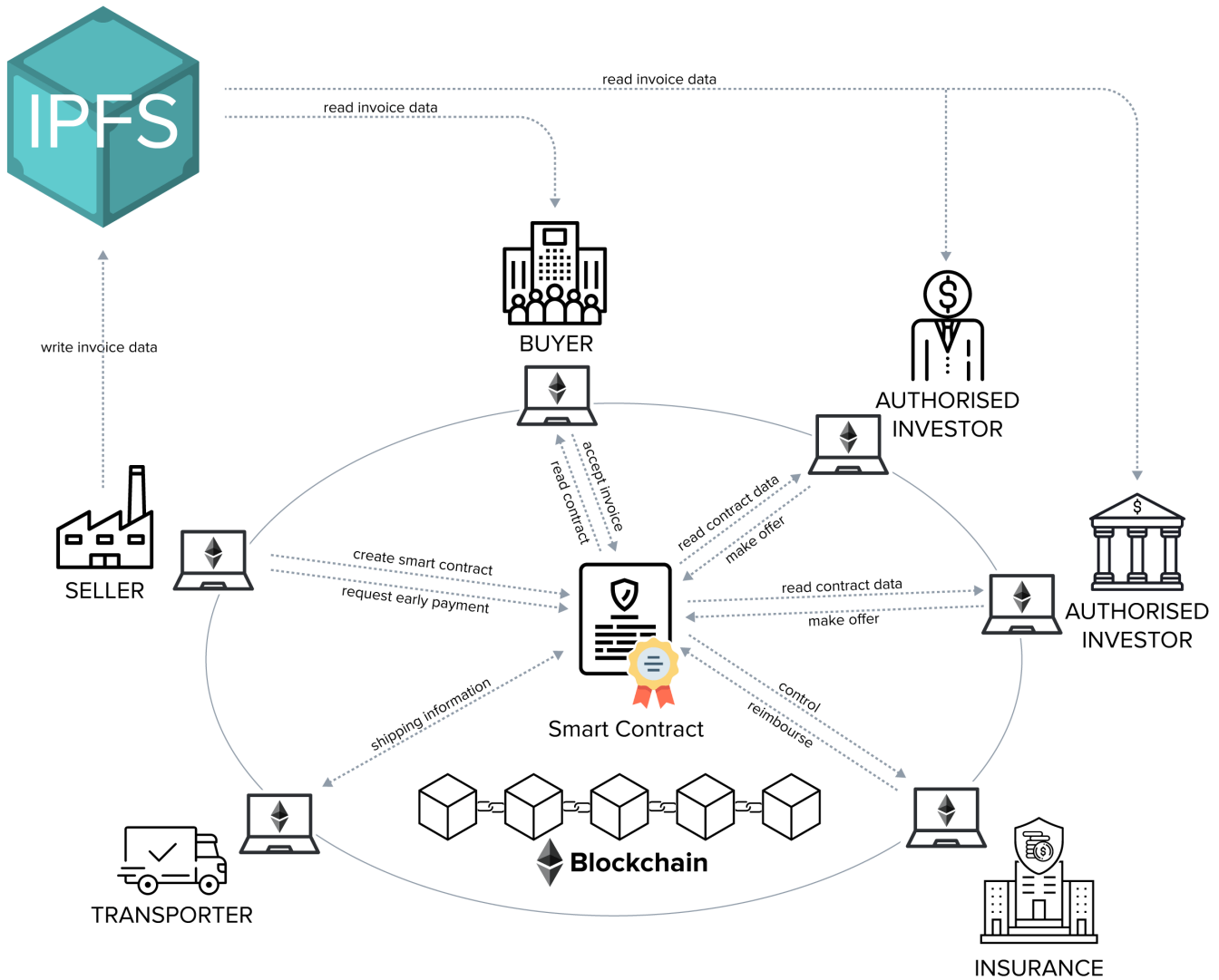


Fig. 1. Invoice financing solution based on blockchain and IPFS.

II. THE PROPOSED INVOICE FINANCING SOLUTION

A. System overview

In this paper, we propose a proof of concept implementation of an invoice financing platform. This platform is designed to fulfill the needs of SMEs; however there is no preclusion for larger players too. Our implementation is based on Inter Planetary File System (i.e., IPFS), it supports reputation profiles, and leverages smart contracts hosted on the Ethereum blockchain. Unlike the traditional invoice financing model, our platform provides an open environment where the chance to finance an invoice is not provided only to banks and financial companies. On the contrary, any Investor can register into the platform and participate to auctions to try acquiring the right to finance an invoice. At the end of the auction (different end condition might be coded in the smart-contract, e.g., time or an immediate buyout value) among the offers that satisfy the requirements (e.g., a minimum requested amount) the best one wins the auction. This enables the enterprise looking for a financing to invite a large number of Investors from around the world and allows getting the best financing offer in a shorter time and with less effort. Since the invoice data are very sensitive and storing this data directly in the blockchain is very expensive, we do not plan to store the whole invoice inside the blockchain. On the contrary, we propose to use IPFS to store these data in a decentralized, distributed manner that is publicly and globally accessible through the use of IPFS hashes. Before storing the invoice data in IPFS, we encrypt it using the Pretty Good Privacy (PGP). Then, we store the IPFS hash into the smart contract. In this way, there is no possibility to retrieve any modification of the invoice content, as the new version of the invoice will have a different IPFS hash that would then not match the hash stored within the smart contract. At the same time, the confidentiality of invoice data is ensured because only the authorized entities will be able to access it. To access the stored data the authorized entity need to perform the following two steps:

- decrypt the generated random key using their private keys;
- use this random key to decrypt the invoice data.

Moreover, in order to reduce frauds, we introduce a reputation system which is based on the past behavior of entities as it is registered inside the ledger. As the blockchain is resilient to modifications, the history of the entities participating into our system is very reliable, and, as the sensitive part of the invoice is not stored in the public blockchain, capable of preventing unwanted disclosures. For instance, the list of invoices that have been paid on time as well as the ones that have gone unpaid or delayed by a buyer can be easily built without compromising the confidentiality of data. This can help Investors in the selection of trustworthy counterparts while pushing malicious buyers to the fringes of the system.

B. System design

The platform allows selecting an account type at registration time. A user can register a Seller account, an Investor account, etc. At the same time, the user needs providing an identity certificate which is unique; this allows preventing the simple creation of new, clean accounts to jump off an identity that has been tarnished by past behavior. The type of account defines the services that are provided by the platform, and every time the contract data changes, a notification is sent to the counterpart. The roles of the entities can be summarized as follows:

- **Seller:** is a company that has goods to be sold, packaged and transferred to the buyer; it is looking to improve its cash flow by creating a smart contract capable of selling the invoice to one of the Investors enrolled in the platform through an auction. This kind of company is often an SME.
- **Buyer:** is a company that would like to purchase the goods from the Seller by paying the shipping amount on delivery and benefits from the delayed payment of the full invoice amount (i.e., the price of goods plus taxes).
- **Transporter:** transports goods and provides information about the shipping status.
- **Authorized Investor:** is a person or a financial institution that is allowed to participate in the auction to buy the invoice at a price lower than its real value to gain a profit.
- **Insurance:** is responsible to reimburse the authorized Investor in case the buyer refuses to pay.

As shown in Figure 1, the Seller writes the invoice data into IPFS and creates a smart contract that specifies the minimum amount required to participate in the auction, the hash to retrieve the invoice from IPFS, etc. Then, he deploys it into the Ethereum blockchain. If the invoice is genuine, the buyer accepts the invoice and pays the shipping amount. When he accepts the invoice the buyer states that he verified all the information mentioned in the invoice and he agreed to pay the shipping amount immediately and the entire amount on the due date as specified in the invoice. Afterward, the Investors can participate in the auction and thus read the invoice data and make an offer after checking the following conditions:

- 1) the invoice has been accepted by the buyer;
- 2) the Invoice ID has not been submitted before;
- 3) the buyer confirmed the delivery in order;
- 4) the reputation profiles of both the Seller and the buyer show that they are trustworthy.

If the reputation profile shows that one of them is untrustworthy or the invoice does not meet one of the above mentioned requirements, then it will not be funded by the Investors. An Investor that decides to finance an invoice in spite of the above mentioned problems is fully responsible of his decision and knows that, in case of fraud, his request of refund will be rejected by the insurance.

Beside protection against double financing and submitting false or modified invoice, our platform mitigates the risk of a buyer that does not pay as agreed. In fact, in our platform the reputation profile will show that a buyer is untrustworthy and Investor may freely take a fully informed decision if they want to run the risk. Thus, our platform facilitates the invoice financing for SME and reduces the risk of frauds.

C. The workflow of the invoice financing process

As mentioned above, the Seller has the choice to open the auction to all the Investors in the platform or only to some predefined Investors. In this subsection, we assume that the Seller authorized only two Investors to facilitate the description of the workflow. This means that the Seller generated a random key offline and encrypted the invoice data using this key, and in order to allow only these two Investors to participate in the auction and the Buyer to verify the invoice, he encrypted the random key with the public key of Investor 1, 2 and the buyer.

Figure 2 shows the message sequence diagram of the process of selling the invoice through an auction; this process is presented with two possible scenarios. In the first scenario the buyer pays on due date of the invoice while in the second the buyer refuses to pay. The different entities involved in the process interact among them and with the smart contract as follows:

- 1) The Seller creates a smart contract and deploys it in the Ethereum blockchain.
- 2) The Buyer decrypts the random key stored in IPFS using his private key and verifies the invoice data. If the invoice is genuine the Buyer accepts the invoice and performs a safe payment of the shipping price. The smart contract holds this amount of Ether until the delivery.

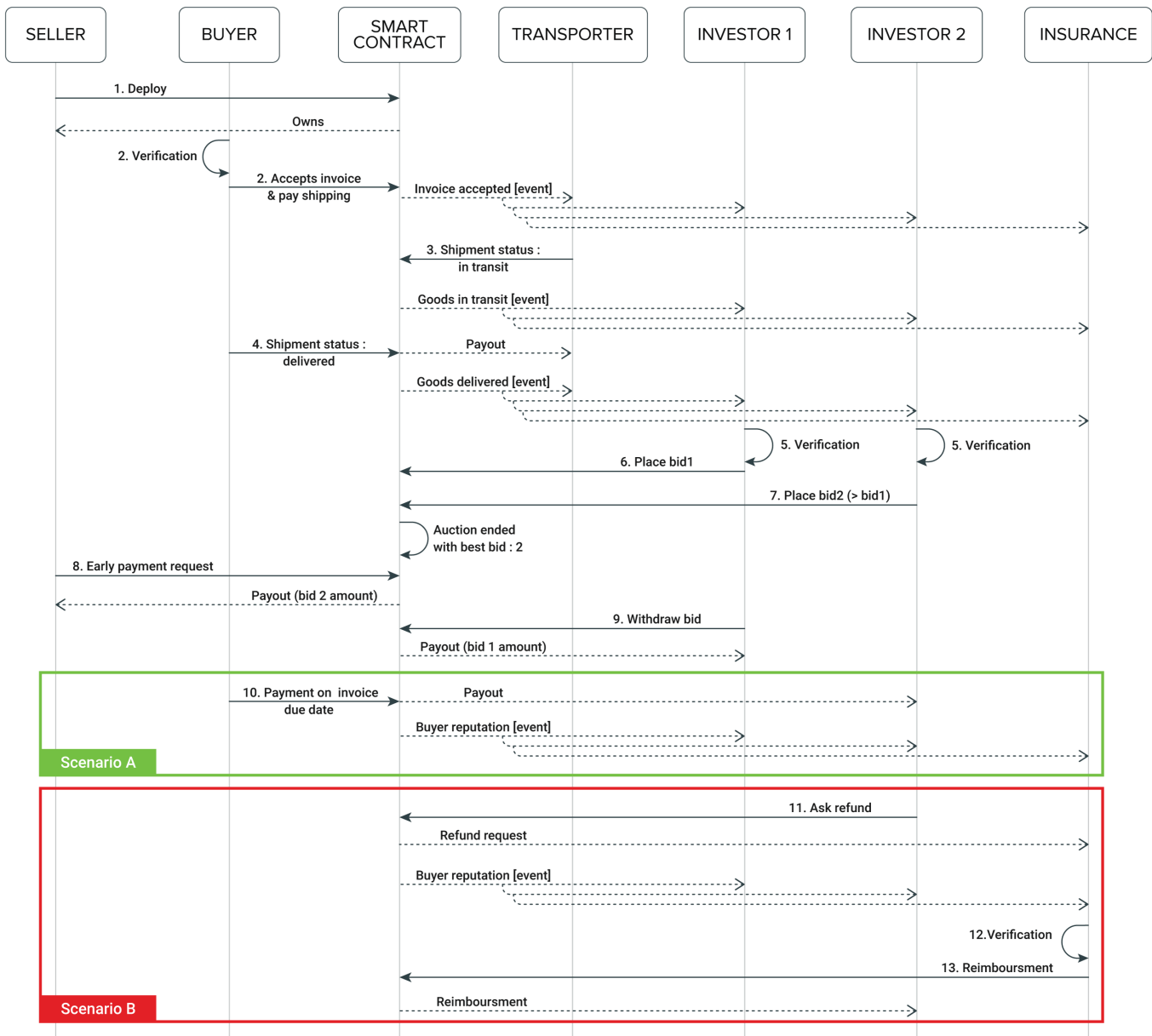


Fig. 2. Sequence diagram of the invoice financing workflow.

- 3) The Transporter verifies if the invoice has been accepted by the Buyer, and then he updates the shipment status on the smart contract to `InTransit` upon receiving the goods.
- 4) The Buyer verifies if the shipment status on the smart contract is `InTransit` then, updates it to `Delivered` once the goods are received. The smart contract payout the Transporter for the shipment.
- 5) The Investors verify the participation conditions mentioned above in order to decide whether to bid on this invoice or not.
- 6) In case all the conditions are met, the first Investor places his bid which should be higher than the minimum bid requested by the Seller.
- 7) The second Investor places his bid which should be higher than the highest bid (i.e., bid 1). The highest bidder become the owner of the invoice when the auction ended.
- 8) The Seller asks for an early payment when the auction ended. The smart contract transfers the highest bid to the Seller.
- 9) The Investor 1 asks to withdraw his bid that was overbid. The smart contract sends to the Investor 1 his corresponding bid amount.
- 10) **In scenario A**, the Buyer pays the entire amount on due date of the invoice to Investor 2 through the smart contract. An event `BuyerReputation(BuyerAddress, "invoice paid on due date")` will be triggered to help in

tracing the Buyer reputation and in notifying all parties.

- 11) **In scenario B**, the Buyer did not pay on due date the invoice as agreed and thus Investor 2 sends a refund request. Two events will be triggered `RefundRequest(msg.sender, "Refund request")` to notify the insurance and `BuyerReputation(BuyerAddress, "Unpaid invoice on due date")` to create notification and save a log about the Buyer reputation. In this scenario, the Buyer profile will show that this Buyer is untrustworthy.
- 12) The insurance verifies if the Investor 2 did not already ask for refunding, and he made the necessary verification before participating in the auction.
- 13) The insurance refunds the Investor 2 through the smart contract.

It is important to mention that in step 8, the Seller manually invokes the smart contract when the auction ends to get his money, as the contract cannot autonomously self-activate; however, automating the reimbursement for Investors that did not win the auction is made possible by executing step 8 and step 9. Nevertheless, we added step 9 to let the Investors withdraw their funds rather than push funds to them automatically for the following security reasons: i) Sending ether back to all the Investors that did not win auction could run out of Ethereum gas ¹. ii) Sending ether to unknown addresses could lead to security vulnerabilities [5].

III. IMPLEMENTATION AND TESTING

This section introduces the implementation and the testing results of the proposed invoice financing platform, thereby detailing *i)* the implementation of the Smart Contracts, *ii)* the validation for a set of relevant test cases and *iii)* the cost analysis on the blockchain network.

A. InvoiceFinancing contract

In this section, we describe the proposed smart contract related interface and algorithms. The smart contract is written in Solidity language and compiled and tested using the Remix IDE which provides the necessary tools for testing and debugging.

Solidity offers a set of system-wide variables, known as Global Variables, that are mainly used to provide information about the blockchain or are general-use utility functions. In Details, the implementation of the InvoiceFinancing contract relies on the following global variables:

- **msg.sender**. it contains the sender of the message to the contract (current call). When the smart contract is deployed, `msg.sender` stores the address of the contract creator (i.e., the Seller of the invoice), while when a smart contract function is called, `msg.sender` contains the address of the function caller;
- **msg.value**. it is the number of wei sent with the message. Wei is the smallest unit of ether, 1 Wei = 0.000000000000000001 ETH;
- **now**. it contains the current block timestamp (alias for `block.timestamp`).

InvoiceFinancing Contract Initialization: This process defines some state variables which are permanently stored in the contract storage and they are initialized by the Seller through the constructor. The constructor is a special function that is called only once at contract deployment. The main state variables are:

- 1) `InvoiceID`, which defines the hash (32 bytes) of the unique identifier of the Invoice.
- 2) `IPFSHash` string, which defines the location from which the encrypted confidential invoice data can be retrieved.
- 3) `TotalAmount`, which defines the amount of Ether the Buyer has to pay on the due date of the invoice.
- 4) `ShippingCost`, which defines the amount of Ether the Buyer has to pay for shipping the goods.
- 5) `InvoiceDueDate`, which defines when the Buyer should pay the full invoice amount.
- 6) `MinimumBid`, which defines the minimum bid accepted by the Seller.
- 7) `AuctionStart`, initialized with the current block timestamp through the special variable `now`;
- 8) `AuctionDuration`, which defines the duration of the auction.

The Invoice Financing contract defines the following nine core functions which are part of the contract interface and can be either called internally or via messages:

AcceptInvoiceAndPayShipping() This function can only be executed by the Buyer. Once the Buyer invokes this function and pays the correct shipping price mentioned in the contract, the smart contract *i)* holds the corresponding amount of Ether, *ii)* changes the invoice state `InvoiceAccepted` (which is initialized by default to `false`) to `true` and *iii)* broadcasts a notification to inform the counter-parties that the Buyer has accepted the invoice. This change in the invoice state is a requirement for the Transporter to be able to hand in the goods.

GoodsReceived() This function can only be executed by the Transporter. The Transporter can update the shipping status to `InTransit` only if the `ShipmentStatus` is `InStock` and `InvoiceAccepted` is `true` to ensure that the Buyer has paid the shipping fees and the smart contract will pay him out upon the delivery. This change in the invoice state is necessary for the Buyer to be able to update the status to delivered.

¹Gas is a cost associated to transaction whose aim is to make Ethereum resilient to Denial of Service attacks. More information can be found at: <https://ethgas.io/>

Algorithm 1: AcceptInvoiceAndPayShipping

```

if msg.sender = BuyerAddress  $\wedge$  msg.value = ShippingCost  $\wedge$  InvoiceAccepted = false
  then
    | InvoiceAccepted  $\leftarrow$  true ;
    | InvoiceStatus("Invoice Accepted");
  else
    | Revert() ;
  end

```

Algorithm 2: GoodsReceived

```

if msg.sender = TransporterAddress  $\wedge$  ShipmentStatus = InStock  $\wedge$  InvoiceAccepted =
  true then
    | ShipmentStatus  $\leftarrow$  InTransit ;
    | ShipmentStatusChanged("Goods in transit");
  else
    | Revert() ;
  end

```

Algorithm 3: GoodsDelivred

```

if msg.sender = BuyerAddress  $\wedge$  ShipmentStatus = InTransit then
    | ShipmentStatus  $\leftarrow$  Delivered ;
    | ShipmentStatusChanged("Goods delivred");
    | Send (TransporterAddress, contract's balance) ;
  else
    | Revert() ;
  end

```

Algorithm 4: PlaceBid

```

initialization: NewBid ;
if msg.sender  $\neq$  Owner  $\wedge$  ShipmentStatus = Delivered  $\wedge$  now  $\geq$  AuctionStart  $\wedge$  now
 $\leq$  AuctionStart + AuctionDuration then
    | NewBid  $\leftarrow$  Bids[msg.sender] + msg.value ;
    | if NewBid > MinimumBid  $\wedge$  NewBid > HighestBidingBid then
        | Bids[msg.sender]  $\leftarrow$  NewBid ;
        | HighestBidingBid  $\leftarrow$  NewBid ;
        | HighestBider  $\leftarrow$  msg.sender ;
    | else
        | Revert() ;
    | end
  else
    | Revert() ;
  end

```

GoodsDelivred() This function can only be executed by the Buyer. To receive the goods from the Transporter, the Buyer should call this function which updates the *ShipmentStatus* to *Delivered* and payout the Transporter. This function requires that *ShipmentStatus* is *InTransit* to prevent the Buyer from calling this function more than once.

PlaceBid() The Seller can not execute this function because he is the owner of the invoice. An Investor can place the bid only when *ShipmentStatus* shows that the goods have been delivered and the auction is still open. His bid should be higher than the minimum bid and the highest bid to become the highest bidder. When the auction end, the highest bidder win.

EarlyPaymentRequest() This function allows the Seller to send the highest bid to his account but only after the auction ended and the highest bid is not equal to zero to ensure that at least one Investor has participated in the auction or to prevent the Seller from calling this function more than once.

Algorithm 5: EarlyPaymentRequest

```

initialization: NewHighestBiddingBid ;
if msg.sender = Owner  $\wedge$  now > AuctionStart + AuctionDuration  $\wedge$ 
  Bids[HighestBidder] > 0 then
  | NewHighestBiddingBid  $\leftarrow$  Bids[HighestBidder] ;
  | Bids[HighestBidder]  $\leftarrow$  0 ;
  | Send (Owner, NewHighestBiddingBid); ;
else
  | Revert() ;
end

```

Algorithm 6: WithdrawBid

```

initialization: NewBid ;
if Bids[msg.sender] > 0  $\wedge$  msg.sender  $\neq$  HighestBidder then
  | NewBid  $\leftarrow$  Bids[msg.sender] ;
  | Bids[msg.sender]  $\leftarrow$  0 ;
  | Send (msg.sender, NewBid) ;
else
  | Revert() ;
end

```

Algorithm 7: PayOnDueDateOfInvoice

```

if now > AuctionStart + AuctionDuration  $\wedge$  now  $\leq$  InvoiceDueDate  $\wedge$  msg.value =
  TotalAmount  $\wedge$  HighestBidder  $\neq$  Address(0) then
  | BuyerPaidOnDueDate  $\leftarrow$  true ;
  | BuyerReputation(BuyerAddress, "Buyer paid on due date");
  | Send (HighestBidder, msg.value) ;
else
  | Revert() ;
end

```

Algorithm 8: AskRefund

```

if msg.sender = highestBidder  $\wedge$  now > InvoiceDueDate  $\wedge$ 
  BuyerPaidOnDueDate = false then
  | RefundRequest(msg.sender, "Refund request") ;
  | BuyerReputation(BuyerAddress, "The Buyer didn't pay on due date") ;
else
  | Revert() ;
end

```

WithdrawBid() This function allows Investors to withdraw their bids that were overbid. Hence, it requires that the *msg.sender* is different from the address of the highest bidder and their bids are higher than zero to ensure that they have placed a bid that they didn't withdraw it yet.

PayOnDueDateOfInvoice() This function can only be executed by the Buyer before or on the invoice due date. It requires that the amount of ether sent by the Buyer equal to the total invoice amount and the highest bidder exist to transfer this amount to his account, change the value of *BuyerPaidOnDueDate* to *true* and notify the counter-parties that the Buyer has paid the invoice on the due date.

AskRefund() This function can only be executed by the highest bidder if the Buyer did not pay on the due date of the invoice. The Smart Contract creates two notifications, the first to inform the Insurance that the highest bidder is asking for a refund, while the second one to notify all the counter-parties that the Buyer is untrustworthy because he did not pay the invoice on the due date.

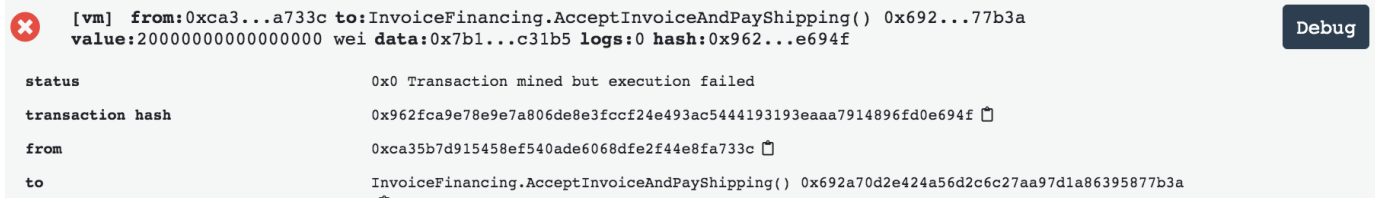
Reimbursement() This function can only be executed by the Insurance after the invoice due date. It requires that the

Algorithm 9: Reimbursement

```

if  $msg.sender = InsuranceAddress \wedge now > InvoiceDueDate \wedge$ 
   $BuyerPaidOnDueDate = false \wedge HighestBidder \neq Address(0)$  then
  | Send (HighestBidder, msg.value) ;
else
  | Revert() ;
end

```



[vm] from:0xca3...a733c to:InvoiceFinancing.AcceptInvoiceAndPayShipping() 0x692...77b3a
value:2000000000000000 wei data:0x7b1...c31b5 logs:0 hash:0x962...e694f Debug

status	0x0 Transaction mined but execution failed
transaction hash	0x962fca9e78e9e7a806de8e3fccf24e493ac5444193193eaaa7914896fd0e694f
from	0xca35b7d915458ef540ade6068dfe2f44e8fa733c
to	InvoiceFinancing.AcceptInvoiceAndPayShipping() 0x692a70d2e424a56d2c6c27aa97d1a86395877b3a

Fig. 3. Details of a failed transaction due to role restriction

value of BuyerPaidOnDueDate equal to false and the highest bidder exist to refund him.

B. Testing and validation

This section describes the details of testing the smart contract code using Remix IDE. We tested all functions for several important aspects and test cases to ensure that the logic and the state of the contract works correctly.

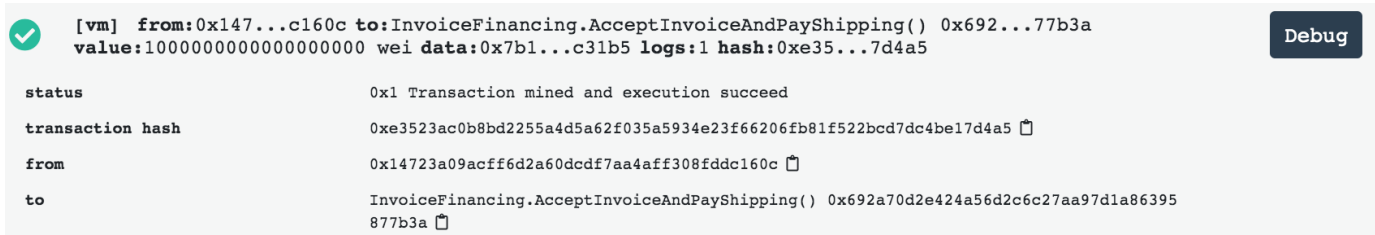
1) *Test Case 1 - Role Restriction:* The contract functions are restricted based on the role of each participant. For example, the invoice can be accepted only by the Buyer, while the change of the shipment status to "in transit" can be done only by the Transporter. All the functions of the Invoice Financing Contract have been tested successfully for role restriction. As shown in Figure 3, if the Seller (that owns the Ethereum address 0xca35b7d915458ef540ade6068dfe2f44e8fa733c) tries to accept an invoice, the transaction fails thereby generating an error.

2) *Test Case 2 - Payable functions:* In this experiment, we tested all payable functions. In particular, we made tests on the requirements related to the received amount, as well as, whether this amount has been transferred to the appropriate recipient as expected for each function. For testing purpose, we set the shipping cost in the contract to 1 ether, the minimum bid to 17 ether and the TotalAmount to 20 ether. We used the default accounts provided by Remix. Each account is loaded with 100 ETH by defaults.

The Invoice Financing contract has the following payable functions: AcceptInvoiceAndPayShipping, PlaceBid, PayOnDueDateOfInvoice, and Reimbursement. AcceptInvoiceAndPayShipping, PayOnDueDateOfInvoice and Reimbursement functions require that the received funds in ether equal to the ShippingCost, TotalAmount, and ReimbursementAmount, respectively. While PlaceBid requires that the collected funds are superior or equal to the MinimumBid.

Those restrictions ensure that the transaction fails if any of the conditions are not met, which prevents the function caller from changing the Invoice Financing Contract state by sending an amount of ether different or inferior to the required amount. For instance, if the Buyer whose address is 0x14723a09acff6d2a60dcdf7aa4aff308fddc160c sends the exact amount of shipping cost (i.e., 1 ETH) upon accepting the invoice, the transaction succeed as shown in Figure 4. However, if an Investor whose address is 0xdd870fa1b7c4700f2bd7f44238821c26f7392148 makes an offer below the minimum bid (i.e., 15 ETH), the transaction fails, and the contract state reverts to its original state as shown in Figure 5.

Besides requirements or conditions related tests, we tested if the amount of ether received by the contract has been transferred successfully to the appropriate recipient. In particular, we did the following tests:



[vm] from:0x147...c160c to:InvoiceFinancing.AcceptInvoiceAndPayShipping() 0x692...77b3a
value:1000000000000000 wei data:0x7b1...c31b5 logs:1 hash:0xe35...7d4a5 Debug

status	0x1 Transaction mined and execution succeed
transaction hash	0xe3523ac0b8bd2255a4d5a62f035a5934e23f66206fb81f522bcd7dc4be17d4a5
from	0x14723a09acff6d2a60dcdf7aa4aff308fddc160c
to	InvoiceFinancing.AcceptInvoiceAndPayShipping() 0x692a70d2e424a56d2c6c27aa97d1a86395877b3a

Fig. 4. A successful transaction of Buyer payment of shipping amount upon accepting the invoice.



Fig. 5. A failed transaction made by an Investor who tried to place a bid with an amount inferior than the minimum bid.

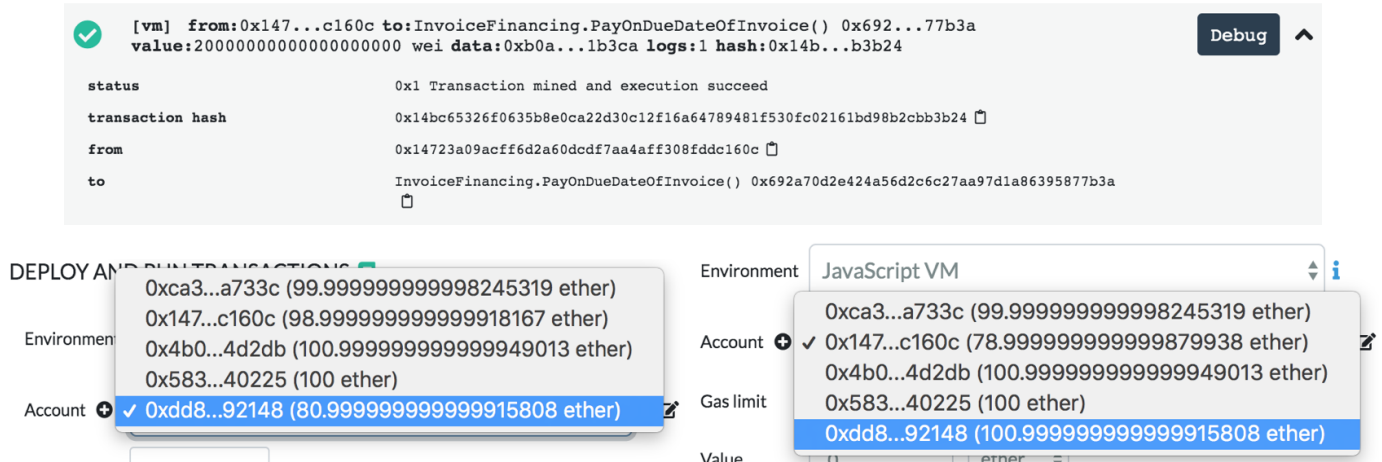


Fig. 6. Successful transfer of total invoice amount from the Buyer to the Investor.

- We checked if the shipping cost sent by the Buyer has been transferred successfully to the Transporter once the Buyer confirmed the delivery of the goods.
- We checked whether the highest bid amount has been transferred successfully to the Seller account when he asked for an early payment as well as if an Investors who did not win the auction were able to withdraw their bids successfully.
- In case of scenario A, we checked whether the winner of the auction received successfully the total invoice amount on the due date upon the Buyer payment. In case of scenario B, we checked whether the winner of the auction received successfully the covered invoice amount from the insurance. Figure 6 shows a successful transaction of the Buyer payment of 20 ETH on due date of the invoice. It is important to notice that the ether balance of the Buyer whose address is 0x14723a09acff6d2a60dcd7aa4aff308fddc160c decreased by 20 ETH plus the transaction fees, while the ether balance of the winner InvoiceFinancing.PlaceBid() whose address is 0xdd870falb7c4700f2bd7f44238821c26f7392148 increased by 20 ETH.

3) *Test Case 3 - Transactions order*: The Invoice Financing Contract functions are designed to be executed in a specific order. This sequence is ensured by using state variables such as `InvoiceAccepted`, `InTransit`, `Delivered` as a requirement to complete the transactions.

For instance, the Investor can not place the bid until the goods are marked as `Delivered` by the Buyer. While the Buyer can not change the shipment status to `Delivered` if the shipment status was not marked `InTransit` by the Transporter and the Transporter in his turn cannot change the shipment status to `InTransit` if the Buyer has not accepted the invoice.

Hence, the following order should be respected by the participants: the Buyer has to accept the invoice first then, the Transporter has to change the shipment status to `InTransit`, after that the Buyer has to change the shipment status to `Delivered`. Only if these transactions have been executed successfully, as shown in event logs in Figure 7, an Investor will be able to place the bid. Otherwise, the operation fails. When the auction ended, the Seller will be able to ask for early payment. Transactions in Scenario A and B depends on state variable `BuyerPaidOnDueDate`, which determines whether the Buyer paid the total invoice amount on the due date or not. Figure 8 and Figure 9 show event logs of transactions that occurred in scenario A and B, respectively.

C. Cost analysis

In Ethereum, every transaction performed on the blockchain network costs Gas. The amount of gas depends on the amount of processing effort required to execute the transaction. The more complex a transaction is, the more gas it requires. The transaction cost is calculated by multiplying the amount of gas with the gas price. The gas price is measured in Gwei, where 1

```
[
  {
    "from": "0xdc04977a2078c8ffdf086d618d1f961b6c546222",
    "topic": "0x1a14c65fef6ef183507f5ffc1345aac906ef984fb57c31378d2ea5418e41
840f",
    "event": "invoiceStatus",
    "args": {
      "0": "Invoice Accepted",
      "Info": "Invoice Accepted",
      "length": 1
    }
  }
] ☐ ☐
```

```
[
  {
    "from": "0xdc04977a2078c8ffdf086d618d1f961b6c546222",
    "topic": "0xd08db4c3f3fc499e1a7bee70d11fb600514e50877f53ebbcf54c19292953
fc9d",
    "event": "ShipmentStatusChanged",
    "args": {
      "0": "Goods in transit",
      "Status": "Goods in transit",
      "length": 1
    }
  }
] ☐ ☐
```

```
[
  {
    "from": "0xdc04977a2078c8ffdf086d618d1f961b6c546222",
    "topic": "0xd08db4c3f3fc499e1a7bee70d11fb600514e50877f53ebbcf54c19292953
fc9d",
    "event": "ShipmentStatusChanged",
    "args": {
      "0": "Goods delivered",
      "Status": "Goods delivered",
      "length": 1
    }
  }
] ☐ ☐
```

Fig. 7. Event logs before an Investor make an offer.

```
[
  {
    "from": "0xdc04977a2078c8ffdf086d618d1f961b6c546222",
    "topic": "0x73507015c45088f5dc7d3e93c457d6354b9a6bc7670f5f314a62eb1e9688
4b94",
    "event": "BuyerReputation",
    "args": {
      "0": "0x14723A09ACff6D2A60DcdF7aA4AFf308FDDC160C",
      "1": "Buyer paid on due date",
      "BuyerAddress": "0x14723A09ACff6D2A60DcdF7aA4AFf308FDDC160C",
      "Description": "Buyer paid on due date",
      "length": 2
    }
  }
] ☐ ☐
```

Fig. 8. Event logs of transaction that occurred in scenario A.

ETH equals 1,000,000,000 Gwei. The price is defined by the sender of the transaction but it is miner's choice in which order they will execute transactions. Since miners prioritize transactions with a higher gas price, the higher the Gas Price is, the faster the transaction will be executed (mined). Vice versa, the lower the price is, the slower the transaction will be executed.

In order to define the transactions cost of operations offered by the proposed Invoice Financing Contract, we deployed it two times into the Rinkeby testnet blockchain using Metamask and Remix tools.

The transactions can be seen at <https://rinkeby.etherscan.io/> using the address 0x77F723F074d8F18A922e1582B58ae for scenario A and 0x01c73Ab3Cc74176e83c4D9Da2EC30bB7127FaEb9 for scenario B.

The addresses used by each participant are as follow:

- **Seller (Owner):** 0x5Eaf9217A42c5EC5685115a994d2341C828979AD
- **Buyer:** 0x4A8d82A7b433b99FCFAB1b8eB48195Cbf7Eaa5B9
- **Transporter:** 0xa924b74EB1a5Da31CbB2f8D3EAA6b7eeF9CDc079
- **Investor 1:** 0xb9898F4f32EafD528Bf65E26a13936588e6A7DB6
- **Investor 2:** 0xd2E1a76B03937B3bF9e009837EF7C53d8c392Cd6
- **Insurance:** 0x0de9478544661D916761c1e88c287a87D06fdC29

Table I summarizes the processing costs measured by our experiment in gas as well as the respective value in ETH and US

```
[
  {
    "from": "0x8609a0806279c94bcc5432e36b57281b3d524b9b",
    "topic": "0x5dec8d72880f4ac9d3a20a09c8d378676126213af39c36e0db192399c75e
2927",
    "event": "refundRequest",
    "args": {
      "0": "0x583031D1113aD414F02576BD6afaBfb302140225",
      "1": "Refund request",
      "HighestBidder": "0x583031D1113aD414F02576BD6afaBfb302140225",
      "Description": "Refund request",
      "length": 2
    }
  },
  {
    "from": "0x8609a0806279c94bcc5432e36b57281b3d524b9b",
    "topic": "0x73507015c45088f5dc7d3e93c457d6354b9a6bc7670f5f314a62e1e9688
4b94",
    "event": "BuyerReputation",
    "args": {
      "0": "0x14723A09ACff6D2A60DcdF7aA4Aff308FDdC160C",
      "1": "The buyer didn't pay on due date",
      "BuyerAddress": "0x14723A09ACff6D2A60DcdF7aA4Aff308FDdC160C",
      "Description": "The buyer didn't pay on due date",
      "length": 2
    }
  }
] }

[
  {
    "from": "0x8609a0806279c94bcc5432e36b57281b3d524b9b",
    "topic": "0x823c9c62ea1468902a7991ddf0e3731bd198ea20b857b79917a8f7ebdf44
023a",
    "event": "reimbursement",
    "args": {
      "0": "0x583031D1113aD414F02576BD6afaBfb302140225",
      "1": "The insurance refunded the investor",
      "HighestBidderAddress": "0x583031D1113aD414F02576BD6afaBfb302140
225",
      "Description": "The insurance refunded the investor",
      "length": 2
    }
  }
] }
```

Fig. 9. Event Logs of transactions that occurred in scenario B.

Dollars (USD). We used <https://ethgasstation.info/> to determine the actual gas costs for each call to the smart contract. When we experimented, on 29 June 2019, the average gas cost which is accepted by the top miners and usually reflects the default wallet price was 3 Gwei = $3 \cdot 10^9$ ETH. The contract creation operation is performed only once by the seller, and the cost is \$1.557. The cost of all the other operations is minimal as all of them are less than \$0.1.

Using the proposed Invoice financing contract, the fees needed to be paid by each participant depends on the scenario and the operations assigned to them. For example, the total cost to be paid by the Investor 2 in case he wins the auction, and the Buyer pays on the due date of the invoice is \$ 0.050. While in case the Buyer didn't pay on the due date, the Investor will call the function `AskRefund` to ask refund from the Insurance and thus the cost increase to \$ 0.074. However, in case the Investor 2 didn't win the auction, he will call `WithdrawBid` instead to get his currency back and thus the total cost is \$ 0.069.

IV. SECURITY ANALYSIS

In this section, we describe the main security properties of the proposed invoice financing solution, then we explain how these properties make our system resilient to the potential fraud scenarios.

- **Security property N1.** Only the Buyer can accept an invoice. This is guaranteed in our system by the fact that the function `AcceptInvoiceAndPayShipping` can be executed only if the Ethereum address of the caller match the Buyer address defined in the smart contract.
- **Security property N2.** Only invoices accepted by the Buyer can be financed. This is guaranteed in our system by the fact that the function `PlaceBid` can not be executed if the goods are not in the state "Delivered". This is means that all the transactions before the confirmation of the goods delivery have been performed successfully including the invoice acceptance by the Buyer.
- **Security property N3.** Only invoices with a confirmed shipping status can be financed. Similar to the security property N2, this is guaranteed in our system by the fact that the function `PlaceBid` cannot be executed unless the goods are not in the state "Delivered". This means that all the transactions before the confirmation of the goods delivery have been performed successfully, including changing the shipment status to "In transit" by the Transporter.

TABLE I
GAS COSTS OF THE SMART CONTRACT FUNCTIONS

Function	Gas used	Cost (Ether)	Cost (USD)
Contract creation	1690138	0,00507	1.557
AcceptInvoiceAndPayShipping	44155	0.00013	0.041
GoodsReceived	29336	0.00009	0.027
GoodsDelivred	37130	0.00011	0.034
PlaceBid (Investor 1)	84170	0.00025	0.078
PlaceBid (Investor 2)	54170	0.00016	0.050
EarlyPaymentRequest	21353	0.00006	0.020
WithdrawBid	20148	0.00006	0.019
PayOnDueDateOfInvoice	38207	0.00011	0.035
AskRefund	26456	0.00008	0.024
Reimbursement	32230	0.0001	0.030

- **Security property N4** A duplicate invoice can not be accepted by the Buyer neither financed by an Investor. This is guaranteed in our system by storing the hash of the unique invoice ID each time the Seller submit a new invoice for sale in a tamper proof logs which can be accessible by anyone in the network thanks to the transparency feature offered by the Ethereum public blockchain.
- **Security property N5.** Inly invoices associated with a successfully delivered good can be financed. This property is guaranteed in our system by the fact that the function `PlaceBid` cannot be executed if the Buyer did not confirm the delivery of the goods.
- **Security property N6.** Investors finance only invoices that come from trustworthy parties. This property is guaranteed in our system by the fact that an Insurance will not refund an Investor in case he purchase an invoice in which at least one of the involved parties (i.e., the Seller and the Buyer) is identified as untrustworthy. Therefore, Investors will be interested in accepting only invoices that come from trustworthy actors.
- **Security property N7.** Integrity of the invoice data. In the proposed solution it is important to ensure that the invoice data stored in IPFS can not be modified once the Buyer approve that it is genuine. Investors wants to make sure that they are participating in the auction to purchase an invoice which is exactly the same approved by the Buyer. Otherwise, there will be no sense to the whole invoice financing system.

In our system, the integrity of the invoice data stored in IPFS is guaranteed by storing the IPFS hash in the Invoice Financing Contract. Indeed, any change to the invoice data will lead to a new hash that will not match the hash in the Invoice Financing Contract.

- **Security property N8.** Confidentiality of the invoice data. The confidentiality of the invoice data is crucial as they may contain data exploitable by adversaries. In our system, the invoice data are stored in IPFS because, besides cost reasons, the Ethereum blockchain is public, and thus, anyone can read the data. Furthermore, in our invoice financing platform, the confidentiality of the invoice data is guaranteed through PGP, a hybrid cryptosystem that combines symmetric-key encryption and public-key encryption. Before the Seller uploads its invoices in IPFS, it generates a symmetric key, called Session Key, and encrypts the invoice data using this key. Then, the Seller encrypts the Session Key with the public keys of the Buyer and the authorized Investors and store it together with the encrypted invoice data. In this way, only the accredited Investors can decrypt the session key using their private keys and use this last to decrypt the invoice data.
- **Security property N9.** Integrity of the transactions and logs. In our system, the reputation profile is built on the transactions history stored in the logs. Thus, it is important to ensure that no one can change it or delete it. Our solution guaranteed the integrity of the information stored in the logs thanks to the immutability feature offered by the blockchain consensus.

We now analyze some typical fraud scenarios and we show how the security properties of the system make these scenarios impossible.

Scenario 1: The Seller knowingly submits a false or modified invoice. In our solution, the invoice will not be funded by an Investor if the Buyer has not previously accepted it. Thus, it is essential to ensure that only the Buyer can confirm the invoice. The Buyer will be interested in accepting the invoice only if it is genuine because his reputation is at stake and he could lose the shipping amount. Thus, thanks to the security properties N1, N2, and N7, our solution is resilient to this kind of fraud.

Scenario 2: The Buyer colludes with the Seller, he accepts the false invoice submitted by the Seller to commit fraud and to split with the Seller the amount of Ether received from the Investor. The cooperation between two parties is still not enough to steal the fundings. In fact, Investors will not purchase an invoice if the Transporter did not confirm that he received the goods, which make this fraud not possible unless they cooperate with the Transporter as well. In this case, both the Seller and the Buyer will be marked as untrustworthy. Hence, our solution is resilient to this kind of fraud thanks to the security properties

N3, N7, and N9.

Scenario 3: The Seller submits a duplicate invoice in order to have double financing. In our solution both the Buyer and the Investor can verify whether or not the invoice has been submitted before using the hash of the "Invoice ID" before accepting the invoice or participating in the auction. Thus, the security properties N4 and N7 prevent double financing of the invoice.

Scenario 4: The Buyer refuses to pay the Investor on the due date because he did not receive the goods. It is essential that the Investor can check if the products have been delivered with a confirmation from the Buyer before participating in the auction. Indeed if the Buyer didn't receive the goods, he is legitimated not to pay the invoice. Thus, the Investor will not get paid due to a fraud committed by the Seller or the Transporter. However, The source of the fraud can be identified easily from the logs. If the records show that the goods are in the state "In Transit" it means the Transporter is the fraudster otherwise the Seller. Thus, our solution is resilient to this fraud, thanks to the security property N5 and N9.

Scenario 5: The Buyer receives the goods but refuses to pay on the due date of the invoice. In this case, the Investor will be refunded by the Insurance, and the Buyer will be marked as malicious and untrustworthy through his reputation profile. Our solution reduces the possibility of this kind of fraud thanks to property N6 and N9. The Buyer will be interested in paying in time to keep the trust of the other entities.

Scenario 6: A third party collects data about the activities of one of the parties involved in the invoice financing activities and leverages it to gain an improper advantage. Although this is not precisely a fraud scenario, we think it's a significant case of misuse. In our system properties, N8 prevents any party not involved in an invoice financing activity to access the confidential data. Hence our solution is resilient to this kind of problem.

V. RELATED WORK

Most researchers, when proposing blockchain based solutions for invoice financing focus mostly on the issue of double financing.

Nijeholt et al. [6] proposed DecReg, a framework based on blockchain technology to address the "double-financing" issue in factoring. The framework has been implemented on a private blockchain. The access to the blockchain is controlled by a central authority (CA). Authors pointed out that the only feasible attack would be a collusion between the seller and the CA, where the CA prevents the financial institution from accessing the network which makes it vulnerable to double-financing. Hence, the financial institution should halt invoice financing until it regains access to the blockchain network.

Hofmann et al. [7] stated that the registration of invoice on the blockchain provides the opportunity to prevent fraud and double-financing issues in invoice discounting and factoring. Each invoice distributed across the network is hashed, timestamped, and given a unique identifier to prevent multiple financing on that particular invoice. However, authors did not provide implementation details such as whether the invoice is registered in public or private blockchain and how the different parties interact with each other.

Similarly, Nicoletti et al. [8] stated that blockchain can play an important role in preventing fraud during procurement finance solution implementation and notably reverse factoring. Blockchain provides complete traceability and real-time visibility on invoices status which prevent the fraudulent organizations from extracting funds from multiple financial institutions by using the same invoice.

In [9], authors proposed a conceptual framework based on blockchain technology for reverse factoring and dynamic discounting. Efficiency, transparency, and autonomy were identified as blockchain value drivers that will improve supply chain finance solutions.

Bogucharskov et al. [10] presented possible interaction between supplier, customer and factor in blockchain-based factoring application. In their interaction model, the factor provides funding to the supplier upon the confirmation of the customer that he received the goods. However, authors did not take in consideration the fraud risks if the supplier or customer are untrustworthy or malicious. In addition to that storing invoice in the public blockchain is very expensive both from the storage and from the computational point of view.

Kayal et al. [11] stated that blockchain technology can be a powerful tool to tackle the financing problems of SMEs. In addition, they conducted an exploratory research into the appetite of the stakeholders involved in invoice factoring and inventory finance for adopting the blockchain technology.

VI. CONCLUSION

Invoice financing is a major component of the global financing market, yet, the lack of a holistic vision of the transactions makes it rife with opportunity for fraudsters. To tackle this problem, we have proposed a distributed platform based IPFS, the Ethereum blockchain smart contracts and a reputation system that allows modulating the costs of insurance. The proof-of-concept implementation we have presented in this paper shows that it is possible to provide an open system with a high level of transparency without public disclosure of all the sensitive data pertaining a financial transaction. Furthermore, our implementation shows that the resilience to tampering provided by a public blockchain allows building a very reliable, evidence based reputation system that can be used to mitigate the risks connected to having to deal with unknown parties. Finally, as the security analysis we have performed clearly shows, our solution is capable of preventing most practical cases of frauds

and, by providing better guarantees, it allows lowering the costs of insurance that is needed to protect the involved parties from the residual fraud cases.

REFERENCES

- [1] H. R. Hasan and K. Salah, "Blockchain-based proof of delivery of physical assets with single and multiple transporters," *IEEE Access*, vol. 6, pp. 46 781–46 793, 2018.
- [2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009. [Online]. Available: <http://www.bitcoin.org/bitcoin.pdf>
- [3] J. Benet, "IPFS - content addressed, versioned, P2P file system," *CoRR*, vol. abs/1407.3561, 2014. [Online]. Available: <http://arxiv.org/abs/1407.3561>
- [4] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger eip-150 revision (759dccc - 2017-08-07)," 2017, accessed: 2018-01-03. [Online]. Available: <https://ethereum.github.io/yellowpaper/paper.pdf>
- [5] N. Grech, M. Kong, A. Jurisevic, L. Brent, B. Scholz, and Y. Smaragdakis, "Madmax: Surviving out-of-gas conditions in ethereum smart contracts," *Proc. ACM Program. Lang.*, vol. 2, no. OOPSLA, pp. 116:1–116:27, Oct. 2018. [Online]. Available: <http://doi.acm.org/10.1145/3276486>
- [6] 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*, ser. BCC '17. New York, NY, USA: ACM, 2017, pp. 29–34. [Online]. Available: <http://doi.acm.org/10.1145/3055518.3055529>
- [7] E. Hofmann, U. M. Strewe, and N. Bosia, *Discussion—How Does the Full Potential of Blockchain Technology in Supply Chain Finance Look Like?* Cham: Springer International Publishing, 2018, pp. 77–87. [Online]. Available: https://doi.org/10.1007/978-3-319-62371-9_6
- [8] B. Nicoletti, *Fintech and Procurement Finance 4.0*. Cham: Springer International Publishing, 2018, pp. 155–248. [Online]. Available: https://doi.org/10.1007/978-3-030-02140-5_6
- [9] Y. Omran, M. Henke, R. Heines, and E. Hofmann, "Blockchain-driven supply chain finance: Towards a conceptual framework from a buyer perspective," in *IPSERA 2017*, Budapest - Balatonfüred, April 2017, pp. 1–15. [Online]. Available: <https://www.alexandria.unisg.ch/251095/>
- [10] A. Bogucharskov, I. Pokamestov, K. Adamova, and Z. Tropina, "Adoption of blockchain technology in trade finance process," *Journal of Reviews on Global Economics*, vol. 7, no. 7, pp. 510–515, nov 2018. [Online]. Available: <https://doi.org/10.6000/1929-7092.2018.07.47>
- [11] A. Kayal, J. Yao, J. Redi, and E. C. Schnoeckel, *Financing Small & Medium Enterprises with Blockchain: An Exploratory Research of Stakeholders Attitudes*, ch. Chapter 4, pp. 65–83. [Online]. Available: https://www.worldscientific.com/doi/abs/10.1142/9781786346391_0004

Meriem Guerar Biography text here.

PLACE
PHOTO
HERE

Alessio Merlo Biography text here.

PLACE
PHOTO
HERE

Mauro Migliardi Biography text here.

PLACE
PHOTO
HERE



Luca Verderame Biography text here.



Francesco Palmieri Biography text here.