A Fraud-Resilient Blockchain-Based Solution for Invoice Financing

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

***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 became the third largest financing market. Nonetheless, the risk of frauds is still very high, 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 due to the lack of confidentiality in public blockchain. In this article, we propose an Invoice financing platform based on a public blockchain supporting both fully open and group-restricted auctioning of invoices. We addressed the confidentiality issue by storing the confidential data encrypted in IPFS and the corresponding hash in the smart contract hosted on Ethereum blockchain. Our blockchain-based solution ensures data confidentiality and benefits from the main properties of the public blockchain required in Invoice financing systems, such as transparency, immutability, trustworthiness, and security. Furthermore, our platform introduces a reputation system based on the past behavior of entities, computed using the blockchain global ledger. Such a reputation system allows insurance companies to modulate the cost of the insurance contracts they offer. This combination guarantees the complete transparency and tamperproofness of a public blockchain, while it allows reducing insurance costs and fraud possibilities.**

***Index Terms*—Auction, blockchain, ethereum, invoice factoring, IPFS, reputation system, smart contract.**

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 significant challenge for any enterprise, but even more so for small to medium enterprises (SMEs). To mitigate cash flow issues, SMEs commonly rely on invoice financing solutions such as invoice factoring.

The process of factoring can be described as follows: an enterprise sells an invoice to a factoring company (typically a financial institution such as a bank) for 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 helps the enterprises in steadying their cash flow, and it represents a significant source of income for the financial institutions. However, the financing process exposes the factoring companies to serious fraud risks mainly because of the lack of a unified view of all the invoice factoring instances. As an example, a well-known invoice factoring fraud risk is the case of double financing. Such fraud happens when the enterprise sells the same invoice to more than one financial institution leveraging the lack of a joint, unified view of the situation among financial institutions. At the due date of the invoice, the buyer will pay the invoice to just one institution, leaving the other(s) unpaid. Another significant source of risk is the reliability of the original buyer. Unfortunately, with a traditional invoice financing system, the financial institution has very little or no information about the original buyer. Hence, it has no way to easily predict if there is a chance that the original buyer will refuse to pay the invoice on the due date.

Beside double financing and trust issues, investors are exposed to other fraud risks. Indeed, the seller may knowingly submit false, modified invoices with the intent to commit fraud, either acting alone or in cooperation with the buyer. To this aim, invoice financing platforms may introduce insurance companies that can refund the defrauded investors in exchange for the subscription to an insurance policy. However, without concrete countermeasures to reduce the fraud opportunity, the cost required for the insurance policy 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 factoring instances.

However, centralized systems are typically expensive and considered a single point of failure. Indeed, they are prone to privacy infringement, data manipulation, and attacks, which may make them unreliable and untrustworthy.

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 [3] that relies on a consensus algorithm to validate the data to be included [4].

A preliminary and partial version of the ideas presented in this article has been introduced in [5], where an auction-based invoice financing solution that leverages the InterPlanetary File

Manuscript received June 30, 2019; revised October 23, 2019, December 5, 2019, and January 27, 2020; accepted January 29, 2020. Date of publication June 30, 2019; date of current version October 9, 2020. Review of this manuscript was arranged by Department Editor Dr. K.-K. R. Choo. *(Corresponding author: Francesco Palmieri.)*

Meriem Guerar, Alessio Merlo, and Luca Verderame are with the DIBRIS, University of Genoa, 16126 Genoa,, Italy (e-mail: meriem5mias@gmail.com; alessio.merlo@unige.it; luca.verderame@unige.it).

Mauro Migliardi is with the Department of Information Engineering, University of Padua, 35122 Padova,, Italy (e-mail: mauro.migliardi@unipd.it).

Francesco Palmieri is with the Department of Computer Science, University of Salerno, 84084 Fisciano,, Italy (e-mail: fpalmieri@unisa.it).

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TEM.2020.2971865

System (IPFS) [6] and the Ethereum blockchain [7] has been proposed. In this article, we reengineered the basic model by consolidating its workflow in order to allow the investors to withdraw immediately their bids that were overbid instead of waiting until the end of the auction. We also improved all the integrity mechanisms in order to use stronger encryption practices and minimize the number of hashes to be stored within the smart contract. Finally, a proof of concept implementation has been presented, allowing us to provide an estimation of the cost of the transactions in terms of ETH and USD. Besides, the security of the proposed scheme has been analyzed in detail by describing the main properties that make our solution robust against the most common frauds.

The rest of this article is organized 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 article.

II. Blockchain-Based Invoice Financing Solution

A. System Overview

We present a proof-of-concept implementation of an invoice financing platform designed to fulfill the needs of SMEs. However, there is no preclusion for larger players too. Our implementation relies on the IPFS and 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 offered only to banks and financial companies. On the contrary, any investor can register into the platform and participate in auctions in order to try acquiring the right to finance an invoice. At the end of the auction, among the offers that satisfy the requirements (e.g., a minimum requested amount), the best one wins the auction. Such a mechanism enables the enterprise looking for 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 save the entire 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 by using pretty good privacy (PGP).[1] Then, we store the IPFS hash into the smart contract. In this way, there is no possibility to alter the invoice content, as the new version would have a different IPFS hash, which does not match with the hash stored within the smart contract. At the same time, only authorized entities will be able to access it, thus ensuring the confidentiality of the invoice data.

In order to access the stored data, the authorized entity needs to perform the following two steps:

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

Moreover, to reduce frauds, we introduce a reputation system based on the past behavior of entities stored inside the ledger. Since the blockchain is resilient to modifications, the history of the entities participating in our system is very reliable. Moreover, as the sensitive part of the invoice is not stored in the public blockchain, the reputation system is capable of preventing unwanted disclosures.

For instance, the list of invoices paid on time, as well as the ones that have gone unpaid or delayed by a buyer, can be quickly built without compromising the confidentiality of data. Such functionality can help investors in the selection of trustworthy counterparts while pushing malicious buyers to the fringes of the system.

B. System Design

The platform offers its users the creation of a new account at registration time. A user can register by using a different role (e.g., seller or investor) and needs to provide a valid identity certificate. Such choice prevents from jumping off an identity that has been tarnished by the past behavior, by creating a different account every time.

The type of account defines the services that are provided by the platform. The roles of the entities can be summarized as follows.

1) *Seller:* is a company that has goods to be sold, packaged, and transferred to the buyer. The seller 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.
2) *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).
3) *Transporter:* transports goods and provides information about the shipping status.
4) *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.
5) *Insurance:* is responsible for reimbursing the authorized investor in case the buyer refuses to pay.

Fig. 1 illustrates a general overview of the proposed invoice financing solution. The seller writes the invoice data into IPFS and creates a smart contract that specifies some information (e.g., the minimum amount required to participate in the auction, the hash to retrieve the invoice from IPFS), which are described in details in Section III-A. 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 transporter ships the goods and updates the shipment status.

Once the buyer confirms the delivery of the goods, the investors can participate in the auction and thus make an offer after verifying the following conditions:

1) the buyer has accepted the invoice;
2) the `Invoice ID` has not been submitted before;

[1][Online]. Available: https://en.wikipedia.org/wiki/Pretty_Good_Privacy

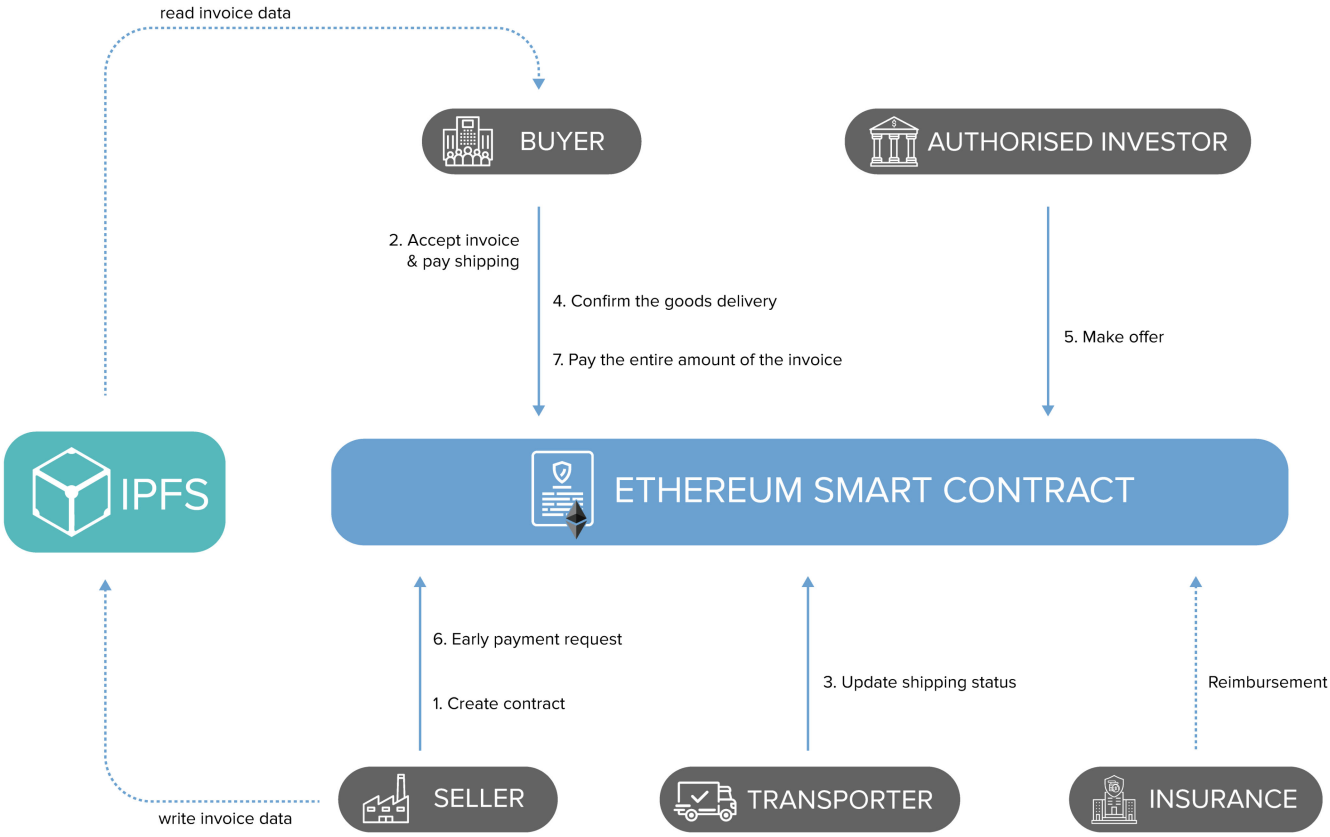


Fig. 1. Overview of the proposed invoice financing solution highlighting the key components (i.e., smart contract, IPFS) and the main participating entities along with their successful transactions.

3) the transporter confirms that he received the goods to be shipped;
4) the buyer confirmed the delivery in order.

Furthermore, the investor can check, by using the reputation system, if any of the entities involved in the transaction is untrustworthy. Still, if an investor decides to finance an invoice anyway, he is fully responsible for his decision. In case of fraud, his request for a refund will be rejected by the insurance.

Besides protection against double financing and submitting false or modified invoices, our platform mitigates the risk of a buyer that does not pay as agreed. Thanks to the reputation system provided by our platform, the buyer will be marked as untrustworthy, and investors make a fully informed decision if they want to run the risk. Thus, our platform facilitates invoice financing for SME and reduces the risk of fraud.

C. Invoice Financing Process Workflow

Using our platform, the seller can open his auctions to all the investors in the platform or only to some predefined investors. In this section, we assume that the seller authorizes only two investors (namely 1 and 2) to facilitate the description of the workflow. In the first step, the seller generates a random key offline and encrypts the invoice data using this key. Then, to allow only the two selected investors to participate in the auction and the buyer to verify the invoice, he encrypts the random key with the public key of investor 1, 2, and the buyer.

Fig. 2 shows the message sequence diagram of the process of selling the invoice through an auction. Such a process opens two possible scenarios. In the first scenario, the buyer pays the invoice on the due date. In the second one, 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 the IPFS by 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.
3) The transporter verifies if the buyer has accepted the Invoice, 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 pays-out the transporter for the shipment.
5) The investors verify the participation conditions mentioned in Section II-B 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.

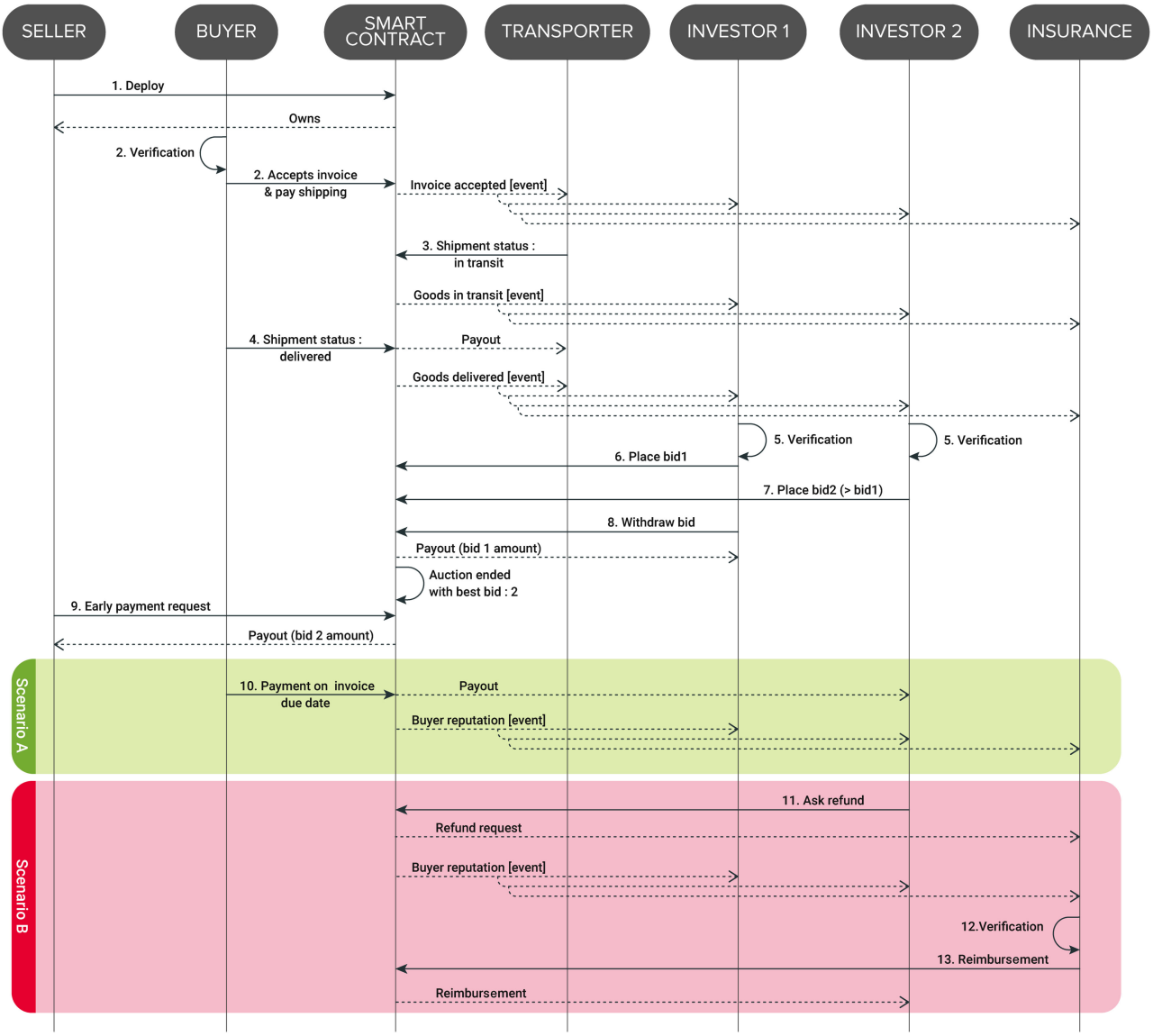


Fig. 2. Sequence diagram showing the interactions of the different parties with the invoice financing contract, the automated payment of transactions when certain conditions are met, and the events used to track the invoice, goods, and the buyer's reputation considering two different scenarios A and B.

7) The second investor places his bid, which should be higher than the highest bid (i.e., bid 1). The highest bidder became the owner of the Invoice when the auction is ended.
8) The investor 1 asks to withdraw his bid that was overbid. The smart contract sends the investor 1 his corresponding bid amount.
9) The seller asks for early payment when the auction is ended. The smart contract transfers the highest bid to the seller.
10) *In scenario A*, the buyer pays the entire amount on the 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's reputation and in notifying all parties.
11) *In scenario B*, the buyer has not paid on the due date the invoice as agreed, and thus, investor 2 sends a refund request. Two events will be triggered `RefundRequest(msg.sender, "Refundrequest")` to notify the Insurance and `BuyerReputation(BuyerAddress,"Unpaid Invoice on due date")` to create a notification and save a log about the buyer's reputation. In this scenario, the buyer's profile will show that this buyer is untrustworthy.
12) The insurance verifies both that investor 2 has not already asked for a refund, and that he has carried out the required verification before participating in the auction.
13) The insurance refunds investor 2 through the smart contract.

It is worth to mention that in step 9, 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 relying on step 9. Nevertheless, we added step 8 to let the investors withdraw their funds rather than push funds to them automatically for the following security reasons: 1) Sending Ether back to all the investors that did not win auctions could run out of Ethereum gas. Gas is a cost associated with the transaction whose aim is to make Ethereum resilient to denial of service attacks.[2] 2) Sending Ether to unknown addresses could lead to security vulnerabilities [8].

III. Implementation and Testing

This section presents the implementation and the testing/evaluation results of the proposed Invoice financing platform, thereby detailing the smart contracts, the validation for a set of relevant test cases, and the cost analysis on the blockchain network.

A. InvoiceFinancing Contract

We describe the proposed smart contract-related interface and algorithms. The smart contract is written by using the 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-purpose utility functions. In details, the implementation of the InvoiceFinancing contract relies on the following global variables:

1) *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;
2) *msg.value:* It is the number of Wei sent with the message. Wei is the smallest unit of Ether, 1 Wei $= 10^{-18}$ ETH;
3) *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 their constructor. The constructor is a special function that is called only once at contract deployment. The main state variables are as follows:

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.

[2][Online]. Available: https://ethgas.io/

Algorithm 1: Accept Invoice And Pay Shipping.

```
if msg.sender= BuyerAddress ∧ msg.value= ShippingCost ∧ InvoiceAccepted=
 false then
    InvoiceAccepted ← true ;
    InvoiceStatus("Invoice Accepted");
else
    Revert() ;
end
```

Algorithm 2: Goods Received.

```
if msg.sender= TransporterAddress ∧ ShipmentStatus= InStock ∧
 InvoiceAccepted= true then
    ShipmentStatus ← InTransit ;
    ShipmentStatusChanged("Goods in transit");
else
    Revert() ;
end
```

Algorithm 3: Goods Delivered.

```
if msg.sender = BuyerAddress ∧ ShipmentStatus = InTransit then
    ShipmentStatus ← Delivered ;
    ShipmentStatusChanged("Goods delivered");
    Send (TransporterAddress, contract's balance) ;
else
    Revert() ;
end
```

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 holds the corresponding amount of Ether, changes the Invoice state `InvoiceAccepted` (which is initialized by default to false) to `true`, and broadcasts a notification to inform the counterparts 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`.

`GoodsDelivered()`

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`

Algorithm 4: Place Bid.

```
initialization: NewBid ;
if msg.sender ≠ Owner ∧ ShipmentStatus = Delivered ∧ now ≥ AuctionStart ∧
 now ≤ AuctionStart + AuctionDuration then
    NewBid ← Bids[msg.sender] + msg.value ;
    if NewBid > MinimumBid ∧ NewBid > HighestBidingBid then
        Bids[msg.sender] ← NewBid ;
        HighestBidingBid ← NewBid ;
        HighestBider ← msg.sender ;
    else
        Revert() ;
    end
else
    Revert() ;
end
```

Algorithm 5: Early Payment Request.

```
initialization: NewHighestBiddingBid ;
if msg.sender= Owner ∧ now > AuctionStart + AuctionDuration ∧
 Bids[HighestBidder] > 0 then
    NewHighestBiddingBid ← Bids[HighestBidder] ;
    Bids[HighestBidder] ← 0 ;
    Send (Owner, NewHighestBiddingBid); ;
else
    Revert() ;
end
```

Algorithm 6: Withdraw Bid.

```
initialization: NewBid ;
if Bids[msg.sender]>0 ∧ msg.sender ≠ HighestBidder then
    NewBid ← Bids[msg.sender] ;
    Bids[msg.sender] ← 0 ;
    Send (msg.sender, NewBid) ;
else
    Revert() ;
end
```

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 cannot 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.

`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 did not withdraw it yet.

Algorithm 7: Pay On Due Date Of Invoice.

```
if now > AuctionStart + AuctionDuration ∧ now ≤ InvoiceDueDate ∧
 msg.value= TotalAmount ∧ HighestBidder ≠ Address(0) then
    BuyerPaidOnDueDate ← true ;
    BuyerReputation(BuyerAddress,"Buyer paid on due date");
    Send (HighestBidder, msg.value) ;
else
    Revert() ;
end
```

Algorithm 8: Ask Refund.

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

Algorithm 9: Reimbursement.

```
if msg.sender= InsuranceAddress ∧ now > InvoiceDueDate ∧
 BuyerPaidOnDueDate= false ∧ HighestBider ≠ Address(0) then
    Send (HighestBidder, msg.value) ;
else
    Revert() ;
end
```

`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 is equal to the total Invoice amount and that the highest bidder exists to transfer this amount to his account, change the value of `BuyerPaidOnDueDate` to `true` and notify the counterparts 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 counterparts 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 value of `BuyerPaidOnDueDate` is equal to `false` and that 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 work 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 `InTransit` can be done only by

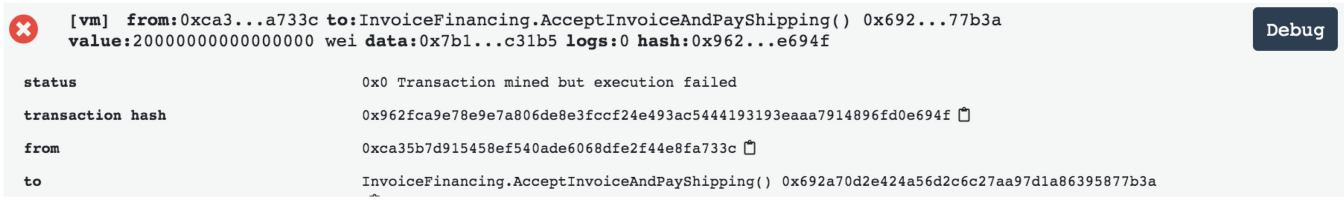


Fig. 3. Details of a failed transaction due to role restriction.

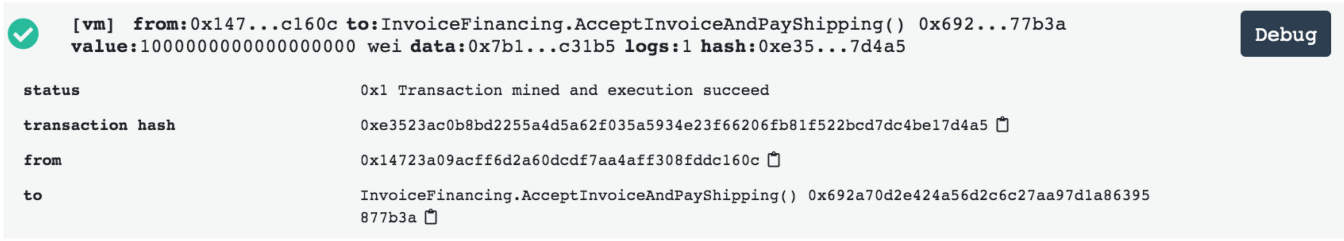


Fig. 4. Successful transaction of buyer payment of shipping amount upon accepting the Invoice.

Fig. 5. Failed transaction made by an investor who tried to place a bid with an amount lower than the minimum bid.

the transporter. All the functions of the invoice financing contract have been tested successfully for role restriction. As shown in Fig. 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 `ShippingCost` in the contract to 1 ETH, the `MinimumBid` to 17 ETH, and the `TotalAmount` to 20 ETH. 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: `PlaceBid`, `AcceptInvoiceAndPayShipping`, `PayOnDueDateOfInvoice`, and `Reimbursement`. `PlaceBid` requires that the collected funds are superior or equal to the `MinimumBid`. While the other functions require that the received funds in Ether are equal to the `ShippingCost`, `TotalAmount`, and `ReimbursementAmount`, respectively.

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 succeeds, as shown in Fig. 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 Fig. 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.

1) 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.
2) We checked whether the highest bid amount had been transferred successfully to the seller account when he asked for early payment as well as if all investors who

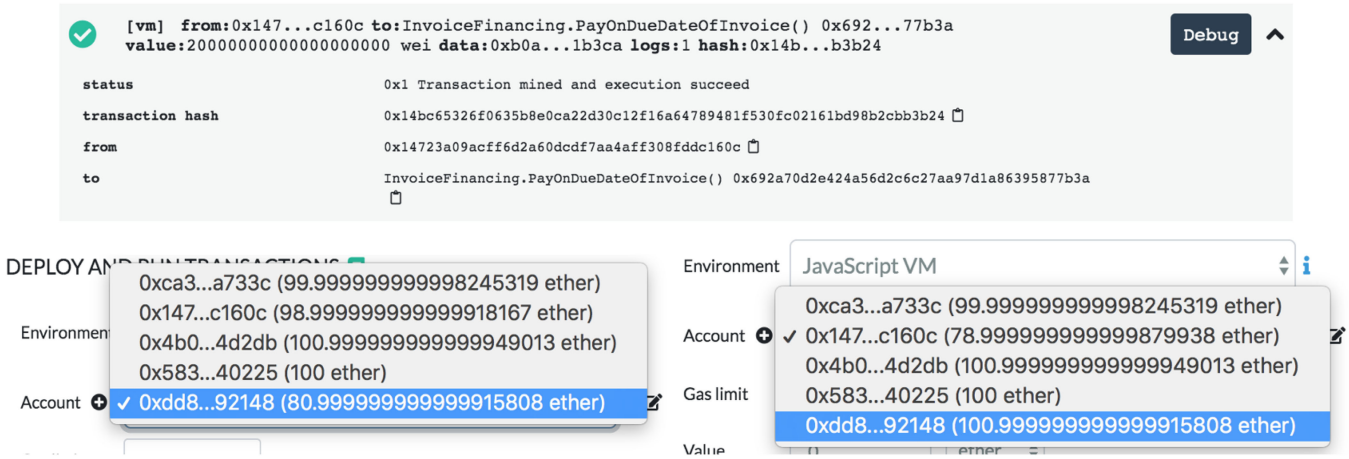


Fig. 6. Successful transfer of total Invoice amount from the buyer to the investor.

did not win the auction were able to withdraw their bids successfully.

3) In the case of scenario A, we checked whether the winner of the auction received the total Invoice amount successfully on the due date upon the buyer payment. In the case of scenario B, we checked whether the winner of the auction received the invoice amount successfully from the insurance. Fig. 6 shows a successful transaction of the buyer payment of 20 ETH on the due date of the invoice. It is important to note that the Ether balance of the buyer whose address is `0x14723a09acff6d2a60dcdf7aa4aff308fddc160c` decreased by 20 ETH plus the transaction fees, while the Ether balance of the winner investor whose address is `0xdd870fa1b7c4700f2bd7f44238821c26f7392148` 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 cannot place the bid until the goods are marked as `Delivered` by the Buyer. On the other hand, the Buyer cannot change the shipment status to `Delivered` if the shipment status was not marked `InTransit` by the transporter. Finally, the transporter 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 Fig. 7, an investor will be able to place the bid. Otherwise, the operation fails. When the auction has ended, the seller is able to ask for early payment. Transactions in Scenarios A and B depend on state variable `BuyerPaidOnDueDate`, which determines whether the Buyer paid the total Invoice amount on the due date or not. Figs. 8 and 9 show event logs of transactions that occurred in scenarios 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 ETH equals 10^9 Gwei. The sender of the transaction defines the price, 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 cost of the transaction of operations offered by the proposed Invoice Financing Contract, we deployed it two times into the Rinkeby testnet blockchain using Metamask and Remix tools.

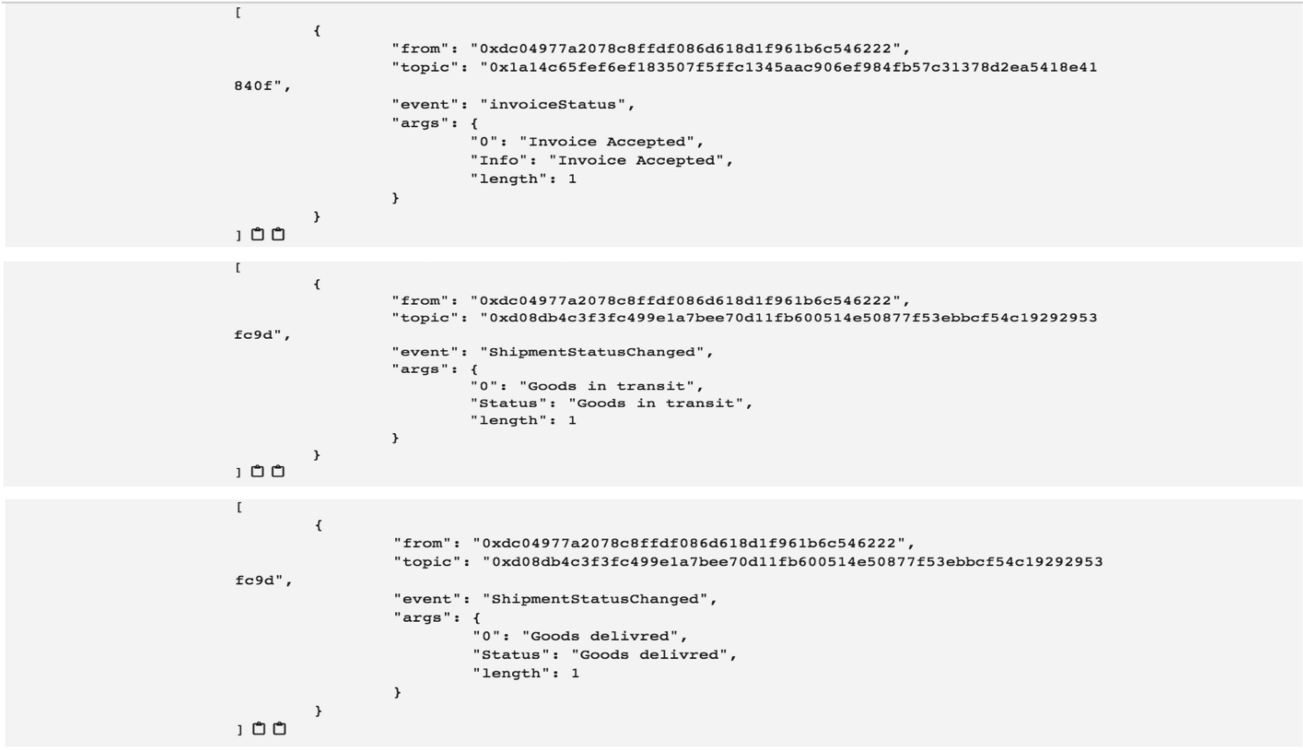
The transactions can be seen online[3] using the address `0x77F723F074d8F18A922e1582B58ae95A98EDFE91` for scenario A and `0x01c73Ab3Cc74176e83c4D9Da2EC30bB7127FaEb9` for scenario B.

The addresses used by each participant are as follows:

1) *Seller (Owner):* `0x5EAf9217A42c5EC5685115a994d2341C828979AD`
2) *Buyer:* `0x4A8d82A7b433b99FCFAB1b8eB48195Cbf7Eaa5B9`
3) *Transporter:* `0xa924b74EB1a5Da31CbB2f8D3EAA6b7eeF9CDc079`
4) *Investor 1:* `0xb9898F4f32EAfD528Bf65E26a13936588e6A7DB6`
5) *Investor 2:* `0xd2E1a76B03937B3bF9e009837EF7C53d8c392Cd6`
6) *Insurance:* `0x0de9478544661D916761c1e88c287a87D06fdC29`

Table I summarizes the processing costs measured by our experiment in gas as well as the respective value in ETH and US Dollars (USD). A url link has been used determine

[3][Online]. Available: https://rinkeby.Etherscan.io/



```
[
        {
                "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 delivred",
                        "Status": "Goods delivred",
                        "length": 1
                }
        }
]
```


Fig. 7. Event logs before an investor makes 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 transactions that occurred in scenario A.

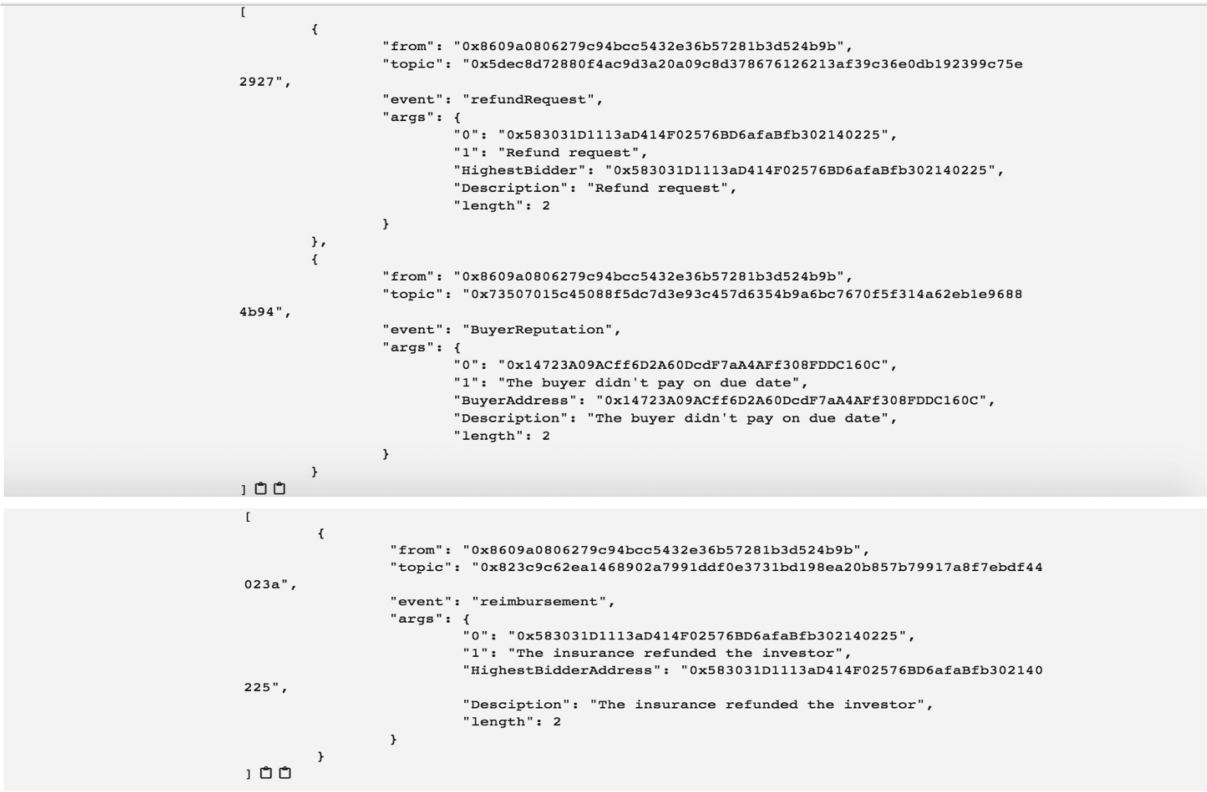
TABLE I
GAS COSTS OF THE SMART CONTRACT FUNCTIONS

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
GoodsDelivered	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

the actual gas costs for each call to the smart contract[4]. When we experimented, on June 29, 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 depend 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 did not pay on the due date, the investor will call the

[4][Online]. Available: https://ethgasstation.info/



```
[
        {
                "from": "0x8609a0806279c94bcc5432e36b57281b3d524b9b",
                "topic": "0x5dec8d72880f4ac9d3a20a09c8d378676126213af39c36e0db192399c75e
2927",
                "event": "refundRequest",
                "args": {
                        "0": "0x583031D1113aD414F02576BD6afaBfb302140225",
                        "1": "Refund request",
                        "HighestBidder": "0x583031D1113aD414F02576BD6afaBfb302140225",
                        "Description": "Refund request",
                        "length": 2
                }
        },
        {
                "from": "0x8609a0806279c94bcc5432e36b57281b3d524b9b",
                "topic": "0x73507015c45088f5dc7d3e93c457d6354b9a6bc7670f5f314a62eb1e9688
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",
                        "Desciption": "The insurance refunded the investor",
                        "length": 2
                }
        }
]
```


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

function `AskRefund` to ask refund from the Insurance, and thus the cost increases to $0.074. However, in case the investor 2 did not 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.

1) *Authorization*
   a) *Only the buyer can accept an invoice (N1):* This is guaranteed in our system by the fact that the function `AcceptInvoiceAndPayShiping` can be executed only if the Ethereum address of the caller matches the buyer address defined in the smart contract.
2) *Traceability*
   a) *Only invoices accepted by the buyer can be financed (N2):* This property is guaranteed in our system by the fact that the function `PlaceBid` cannot be executed if the goods are not in the state `Delivered`. This implies that all the transactions performed before the confirmation of the goods are successful, including the Invoice acceptance by the buyer.
   b) *Only invoices with a confirmed shipping status can be financed (N3):* 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 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 `InTransit` by the transporter.
   c) *Investors finance only invoices that come from trustworthy parties (N4):* This property is guaranteed in our system by the fact that an Insurance will not refund an investor in case he purchases 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.
3) *Availability (Invoice ID)*
   a) *investorA duplicate invoice cannot be accepted by the buyer nor it can be financed by an investor (N5):* This is guaranteed in our system by storing the hash of the unique Invoice ID each time the seller submits a new Invoice for sale in a tamper-proof log, which can be accessible by anyone in the network thanks to the transparency feature offered by the Ethereum public blockchain.
4) *Integrity*
   a) *Integrity of the invoice data (N6):* In the proposed solution, it is important to ensure that the Invoice data stored in IPFS cannot be modified once the buyer approves that it is genuine. Investors want to make sure that they are participating in the auction to purchase an Invoice, which is the same approved by the buyer. Otherwise, there will be no sense of 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.

b) *Integrity of the transactions and logs (N7):* In our system, the reputation profile is built on the transaction history stored in the logs. Thus, it is important to ensure that no one can change it or delete it. Our solution guarantees the integrity of the transactions and the information stored in the logs thanks to the immutability feature offered by the blockchain consensus.

5) *Confidentiality*

a) *Confidentiality of the invoice data (N8):* 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 his own Invoices in IPFS, he generates a symmetric key, called Session Key, and encrypts the Invoice data by using this key. Then, the seller encrypts the Session Key with the public keys of the buyer and the authorized investors and stores 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 one to decrypt the Invoice data.

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 N6, 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 the two parties is still not enough to steal the funding. Investors would 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, N6, and N7.

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 N5 and N6 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 does not 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 `InTransit`, it means the transporter is the fraudster otherwise the seller. Thus, our solution is resilient to this fraud, thanks to the security properties N3 and N7.

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 properties N4 and N7. 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 this may be seen as a significant case of misuse. In our system properties, N8 prevents any party that is 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.* [1] 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 claim 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. In contrast to the proposed solution, DecReg was designed to address the double financing issue solely. Besides, it prevents access to the confidential data solely from entities outside the network, while the entities inside the network, even those who are not concerned, still have access to this data.

Hofmann *et al.* [9] stated that the registration of Invoices 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.* [10] stated that blockchain could 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 prevents fraudulent organizations from extracting funds from multiple financial institutions by using the same Invoice.

In [11], Omran *et al.* 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.* [12] 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, the authors did not take into consideration the fraud risks if the supplier or customer is untrustworthy or malicious. In addition to that, storing an Invoice in the public blockchain is very expensive both from the storage and from the computational point of view. Similarly, Euler Hermes Digital Agency [13] developed a blockchain application for trade finance that stores the invoice data in the public blockchain. The platform allows the seller and their counterparts to register and manage their Invoices on the Ethereum Blockchain. However, besides the cost of transactions and smart contract deployment, their solution does not ensure the confidentiality of the Invoice sensitive data.

Kayal *et al.* [14] stated that blockchain technology could be a powerful tool to tackle the financing problems of SMEs. In addition, they conducted exploratory research into the appetite of the stakeholders involved in Invoice factoring and inventory finance for adopting the blockchain technology. In [15], Hyvärinen *et al.* developed a blockchain-based prototype for managing dividend flows using the Ethereum smart contract. Their solution increases the transparency of the transactions and overcomes the double-spending problem in the public taxation sector.

Recently, Battaiola *et al.* [2] introduced a permissioned blockchain-based solution for Invoice factoring that uses Hyperledger. In their security protocol, the secret data are stored locally by the peer who owns it, and it can be shared with the involved participants through a private, secure channel. At the same time, the Invoice hashes, and the status information are stored in the distributed ledger in order to enable the factors to check whether an Invoice has been already factored. However, unlike the proposed solution, their system does not address the frauds in the case of scenarios 2, 4, and 5 presented in Section IV. Also, the security of their solution relies on the trust of a set of parties called payment service providers who are responsible for making the money transfers and updating the factor status.

Besides financial frauds, blockchain has also been used to prevent odometer fraud [16] and work history related fraud [17]. In [16], Chanson *et al.* store the hash of mileage and GPS data in the blockchain. While, in [17], Sarda *et al.* store both the work history data encrypted with the employees' public key and the hash of the unencrypted work history data in the Ethereum blockchain.

VI. Conclusion

Invoice financing is a significant 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 proposed a distributed platform based on IPFS, smart contracts hosted on the Ethereum blockchain, and a reputation system that allowed modulating the costs of Insurance. This will bring significant benefits for any entity involved, ranging from customers to banking organizations, essentially due to improvements in transparency of the whole process and the following risk reduction for the involved financial institutions. That is, the blockchain will make available exhaustive, trustworthy, and transparent recordings of any transaction, with a success history that can be used for taking better decisions in financial planning and request/granting activities. In addition, the use of smart contracts makes the whole invoice financing process quicker and reliable, making any invoice a more valuable asset and facilitating the automation of all the associated activities, after the agreement of the involved parties. This will also be effective in fraud-prevention and in reducing the administrative costs associated with credit, with obvious effect on online marketplaces, by also creating new financial opportunities for small and medium enterprises that will take advantage of smart contracts through more flexible and reliable invoice financing practices. Furthermore, the use of blockchain will give new opportunities to open innovation networks for developing more sustainable business models. Nevertheless, while the involved technologies may be greatly helpful in the design of new financial solutions, yet it needs to be managed so that it does not introduce additional costs that may marginalize the advantages obtained. In particular, the management of technologies such as blockchain, has to be carefully planned to avoid incurring in unexpected pitfalls. In this article, we presented a design where most of the management activities were automated through the introduction of decentralized control implemented in smart contracts, and the proof-of-concept implementation presented showed that it is possible to provide an open system with a high level of transparency without public disclosure of all the sensitive data pertaining to a financial transaction. Furthermore, our implementation showed that the resilience to tampering provided by a public blockchain allowed 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 that we performed clearly showed, our solution is capable of preventing most practical cases of frauds and, by providing better guarantees, it allows managing the system with lower expenditures for the Insurance that is still needed to protect the involved parties from the residual fraud cases. In future work, we plan to analyze the implemented Invoice financing contract for security vulnerabilities using tools such as Oyente [18] and Securify [19], develop a web UI to interact with the smart contract, investigate the possibility to replace the transporter by an IoT device fixed in the truck to update the shipping status.

REFERENCES

[1] H. Lycklama à Nijeholt, J. Oudejans, and Z. Erkin, "Decreg: A framework for preventing double-financing using blockchain technology," in *Proc. ACM Workshop Blockchain, Cryptocurrencies Contracts*, 2017, pp. 29–34. [Online]. Available: http://doi.acm.org/10.1145/3055518.3055529

[2] B. Ettore, M. Fabio, N. N. Chan, and S. Pierantonia, "Blockchain-based invoice factoring: From business requirements to commitments," in *Proc. DLT@ ITASEC*, 2019, pp. 17–31.

[3] 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.

[4] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[5] M. Guerar, L. Verderame, A. Merlo, and M. Migliardi, "Blockchain-based risk mitigation for invoice financing," in *Proc. 23rd Int. Database Appl. Eng. Symp.*, 2019, pp. 20:1–20:6. [Online]. Available: http://doi.acm.org/10.1145/3331076.3331093

[6] J. Benet, "IPFS—Content addressed, versioned, P2P file system," *arXiv:1407.3561*, 2014. [Online]. Available: http://arxiv.org/abs/1407.3561

[7] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger eip-150 revision (759dccd - 2017-08-07)," 2017, Accessed: 2018-01-03. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[8] 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, pp. 116:1–116:27, Oct. 2018. [Online]. Available: http://doi.acm.org/10.1145/3276486

[9] E. Hofmann, U. M. Strewe, and N. Bosia, *Discussion—How Does the Full Potential of Blockchain Technology in Supply Chain Finance Look Like?* Cham, Switzerland: Springer, 2018, pp. 77–87. [Online]. Available: https://doi.org/10.1007/978-3-319-62371-9_6

[10] B. Nicoletti, *Fintech and Procurement Finance 4.0*. Cham, Switzerland: Springer, 2018, pp. 155–248. [Online]. Available: https://doi.org/10.1007/978-3-030-02140-5_6

[11] Y. Omran, M. Henke, R. Heines, and E. Hofmann, "Blockchain-driven supply chain finance: Towards a conceptual framework from a buyer perspective," in *Proc. IPSERA*, Budapest, Balatonfüred, Apr. 2017, pp. 1–15. [Online]. Available: https://www.alexandria.unisg.ch/251095/

[12] A. Bogucharskov, I. Pokamestov, K. Adamova, and Z. Tropina, "Adoption of blockchain technology in trade finance process," *J. Rev. Global Econ.*, vol. 7, no. 7, pp. 510–515, Nov. 2018. [Online]. Available: https://doi.org/10.6000/1929-7092.2018.07.47

[13] EHDA, [Online]. Available: https://github.com/eulerhermesda/Smart_Factor, 2018.

[14] A. Kayal, J. Yao, J. Redi, and E. C. Schnoeckel, *Financing Small & Medium Enterprises With Blockchain: An Exploratory Research of Stakeholders Attitudes*. Singapore: World Scientific, 2019, ch. 4, pp. 65–83. [Online]. Available: https://www.worldscientific.com/doi/abs/10.1142/9781786346391_0004

[15] H. Hyvärinen, M. Risius, and G. Friis, "A blockchain-based approach towards overcoming financial fraud in public sector services," *Bus. Inf. Syst. Eng.*, vol. 59, no. 6, pp. 441–456, Dec. 2017. [Online]. Available: https://doi.org/10.1007/s12599-017-0502-4

[16] M. Chanson, A. Bogner, F. Wortmann, and E. Fleisch, "Blockchain as a privacy enabler: An odometer fraud prevention system," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput. Int. Symp. Wearable Comput.*, 2017, pp. 13–16. [Online]. Available: http://doi.acm.org/10.1145/3123024.3123078

[17] P. Sarda, M. J. M. Chowdhury, A. Colman, M. A. Kabir, and J. Han, "Blockchain for fraud prevention: A work-history fraud prevention system," in *Proc. 17th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun./12th Int. Conf. Big Data Sci. Eng.*, Aug. 2018, pp. 1858–1863.

[18] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 254–269. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978309

[19] P. Tsankov, A. M. Dan, D. Drachsler-Cohen, A. Gervais, F. Buenzli, and M. T. Vechev, "Securify: Practical security analysis of smart contracts," in *Proc. CoRR*, vol. abs/1806.01143, 2018, pp. 67–82. [Online]. Available: http://arxiv.org/abs/1806.01143

Meriem Guerar received the master's degree in information systems and networks, in 2011, and the Ph.D. degree in computer science, in 2017, both from the University of Sciences and Technology of Oran, Oran, Algeria.

She is currently a Postdoctoral Research Fellow with the University of Genova, Genova, Italy. Her main research interests include the areas of authentication, security and usability, blockchain and smartphone security.

Alessio Merlo received the M.Sc. and Ph.D. degrees in computer science from the University of Genova, Genova, Italy, in 2005 and 2010, respectively.

He is currently serving as a Senior Assistant Professor of Computer Engineering at the University of Genoa. His research activity mainly focuses on Mobile and IoT Security. He coauthored more than 90 scientific papers spanning from Distributed Systems to Cybersecurity research topics.

Mauro Migliardi received the M.S. degree in electronic engineering and the Ph.D. degree in computer engineering from the University of Genoa, Genova, Italy, in 1991 and 1995, respectively.

He is currently an Associate Professor of Cybersecurity with the University of Padua, Padua, Italy, and an Adjunct Professor with the University of Genoa. His main research interest include the engineering of secure, energy aware mobile, distributed and IoT systems. He tutored more than 100 bachelor's, master's and Ph.D. students at the Universities of Genoa (IT), Padua (IT), and Emory (US). He has authored or coauthored more than 150 scientific papers.

Francesco Palmieri received the M.S. and Ph.D. degrees in computer science from the University of Salerno, Fisciano, Italy.

He is currently a Full Professor of Computer Science with the University of Salerno. His research interests include advanced networking protocols and architectures and network security. He has been the Director of the Networking Division, University of Naples "Federico II" and contributed to the development of the Internet in Italy as a Senior Member of the Technical-Scientific Advisory Committee and of the CSIRT of the Italian NREN GARR. He has authored more than 200 scientific papers, served as the Editor-in-Chief of an international journal and has participated in the editorial board of many other highly-reputed ones.

Luca Verderame received the Ph.D. degree in electronic, information, robotics, and telecommunication engineering from the University of Genoa, Genova, Italy, in 2016, where he worked on mobile security.

He is currently working as a Postdoctoral Research Fellow with the Computer Security Laboratory, and he is also the CEO and Co-founder of Talos, a cybersecurity startup and university spin-off. His research interests mainly cover information security applied, in particular, to mobile and IoT environments.