Master Thesis

Blockchain and Trade Finance: A Smart Contract-Based Solution

Alexander Blum

Submission Date: November 18, 2019

Supervised by:
Prof. Dr. Fabian Schär
Credit Suisse Asset Management (Schweiz) Professor for
Distributed Ledger Technologies and Fintech
Center for Innovative Finance, University of Basel

Abstract

In this paper, we present a solution how solve the unbalanced risk distribution between two unknown trading parties with the help of blockchain technology, in particular the public Ethereum blockchain. In a first step, we analysed the traditional financial instruments such as letter of credit and documentary collection as well as the popular method of open account payment from a game-theoretical perspective. We found that these payment methods are only incentive compatible in the presence of banks. In a second step, we developed a smart contract which makes intermediaries obsolete, has significantly lower transaction costs and accelerates the overall trade finance process. A legal classification of smart contracts completes our work.

Keywords: Trade Finance, Blockchain, Smart Contracts, Ethereum.
JEL: G23, O31, O33

Contents

List of Tables

List of Figures

University of Basel

Center for Innovative Finance

Plagiatserklärung

Ich bezeuge mit meiner Unterschrift, dass meine Angaben über die bei der Abfassung meiner Arbeit benutzten Hilfsmittel sowie über die mir zuteil gewordene Hilfe in jeder Hinsicht der Wahrheit entsprechen und vollständig sind. Ich habe das Merkblatt zu Plagiat und Betrug vom 22. Februar 2011 gelesen und bin mir der Konsequenzen eines solchen Handelns bewusst.

Alexander Blum

Basel, November 18, 2019

Acknowledgements

I would like to thank Prof. Dr. Fabian Schär for his valuable input and feedback during the Master Blockchain Research Seminar. Further, I would also like to thank Sebastian Blum for proofreading this thesis by great care.

1 Introduction

The greater attraction of open account (OA) financing based on a higher level of security, better communication and more available information has reduced the share of traditional trade finance instruments such as the letter of credit (LC) and documentary collection (DC) from 50 % to 15 % over the last decades (Dab et. al. (2016); Ganesh and Olson (2018)). Further, greater competition between banks puts pressure on the prices, and new regulations make the overall process more cost intensive (Dab et. al., 2016). Defined by the International Chamber of Commerce, trade finance allows firms to access new markets and mitigate risks which occur by importing or exporting products (International Chamber of Commerce, 2019*a*).
According to Dab et. al. (2016), trade finance customers search for secure, fast, transparent and favourable methods for their trading. Also, today in a highly digitalised world, trade finance processes are mostly paper based, not automatized, and include numerous parties who do not trust each other by nature (Dietrich, 2018). Two-thirds of all submitted documents have an offset at first, which requires corrections and makes the process inefficient (Ganesh and Olson, 2018). Many different documents are needed to transfer goods in an international environment; for example, an LC transaction requires more than 30 documents from more than 20 parties (Dietrich, 2018). Based on the estimation of Dietrich (2018), blockchain technology can shorten the process's duration by 90 %. While the use of public blockchains is crucial for applications which require many transactions per second and need to be fast (a few seconds or less), in the case of trade finance, it is not elementary to reach this level of performance (Dietrich, 2018).
Since we use the Ethereum blockchain, the main programming language to develop smart contracts is Solidity; it has various elements from well-known languages like Python, C++ and Javascript (Sharma, 2019).

In this paper, we want to answer the following questions: How can we design a trading process, in which no trust between two unknown parties is needed? What will the source code in Solidity look like? How much cheaper is a trade agreement that uses smart contracts? What are the legal implications of a contract based only on source code?

To answer these questions, we structure this work as follows: in section 2 we give a short overview of the principle of blockchain and take a special look at Ethereum. In section 3 we describe the different payment methods traditionally used in today's trading environment and analyse them from a game theoretical perspective. Finally, in section 4 we develop a smart contract and analyse the main functions in that smart contract based on the Solidity code.

2 Topic Enquiry

This section provides a brief overview of blockchain basics, focussing on the relevant topics of the Ethereum blockchain and the use of smart contracts. Blockchain, based on the distributed ledger technology (DLT), is a decentralized network. This technology relies on a consensus protocol which makes intermediaries redundant and hence creates trust among the network participants (Atzei et al., 2017, p. 164). Miners that serve as nodes are spread all over the world and confirm transactions in blocks and get a transaction fee in case of success; higher transaction fees lead to faster transaction processing. The successful miner who adds a block to the blockchain network obtains a reward, usually cryptocurrency or a transaction fees, depending on the blockchain protocol.

2.1 Ethereum Blockchain

In general, the Ethereum blockchain is a progression of the Bitcoin blockchain which is used to transfer financial values such as Bitcoin in a peer to peer network. The key differences are the Turing-completeness and the inclusion of the state in each block (Vujičić et al., 2018). The Ethereum blockchain is based on an account-based model in which every account has a 20-byte address and a state (Vujičić et al., 2018). Each Ethereum account consists of a nonce, ether balance, contract code hash, and storage root (Buterin et al., 2014). The nonce symbolizes a security mechanism against the double-spending problem so that each transaction can be used just once. The Ether balance shows the amount of the account of Ether (Vujičić et al.,

2018).
We have to differentiate between two different account types - namely, externally owned accounts (EOAs) and contract accounts (CAs) (Ethereum Community, 2019*a*). As we can derive the characteristics from the term EOA, this account type is owned by a person, has an account balance and can send messages. In contrast, CAs are controlled by the written code and the execution is triggered by a transaction or message call (Kenneth, 2018). A transaction via a message call between EOAs can be the transfer of Ether, whereas a call to a CA can be the interaction with or creation of a smart contract (Schütz and Fertig, 2019, pp. 120ff.). All transactions are stored on the blockchain and can be identified by the transaction hash. Further, CAs can communicate with each other by the help of messages; every first message must be lead by a transaction from an EOA (Schütz and Fertig, 2019, pp. 120ff.).
The execution of decentralized applications (DApps) on the Ethereum blockchain takes place under the Ethereum virtual machine (EVM), run by the network's nodes (Ethereum Community, 2019*a*). To protect the EVM from attacks and poorly programmed code that could trigger an infinite loop, Ethereum uses a gas system (Miller, 2019). Every operation consumes a predefined amount of gas - for example, five *Wei* for an add operation - which is equivalent to a fee structure. The gas limit value determines the maximum amount someone is willing to pay for the execution of the transaction. Any unused gas will not be charged against the initiators' account. In addition, the user sets a price per unit of gas, called gas price. As miners can choose from a transaction pool, it makes sense to set a higher gas price if our transaction should be validated in a block and quickly added to the blockchain. The total transaction costs are calculated by multiplying the gas price by the gas used (Wood et al., 2014).

2.2 Smart Contracts

Smart contracts are one application of DApps on the Ethereum blockchain. In spite of their name, these contracts are not as smart as supposed and are not equivalent to process automation. Processes can be mapped whenever they contain a rule following the logic of an if-then condition (Szabo,

1997). An example is a vending machine for drinks; after paying the displayed amount, we get the product (Szabo, 1997). Smart contracts follow the same principle: the execution will proceed only if conditions are satisfied and secured by the consensus protocol of the blockchain (Berentsen and Schär, 2017, p. 289).
The code of a smart contract is written in the Solidity programming language and consists of two parts: the application binary interface (ABI), which contains the functions of the contract, and the byte code that, after compilation, will be executed on the blockchain (Lee, 2019, p. 136).

3 Trade Finance

In general, for a trading deal, we need a seller and a buyer who contract with each other. To finance and secure national and especially international trade, third parties such as banks and insurance companies are involved in a transaction. They finance the gap between the trade agreement and the settlement of payment, and they also manage risks (Fingerand and Schuknecht, 1999, pp. 4f.). Generally, we distinguish between pre-shipment and post-shipment trade financing. In the first case, the financing must cover production costs such as materials or wages, whereas in the second case, a funding is needed to bridge the gap between the shipment of goods and the receipt of money (International Trade Center, 2009, pp. 37ff.). For example, pre-shipment financing can be done by pre-payment. Pre-payment means that the buyer lends money from the bank in order to make it available to the seller, which she or he normally would not get from a bank. The option of post-shipment financing can be factoring or reverse factoring, which we explain more detailed in section 3.2.8.
Because international trade means acting across multiple legal systems and multiple understandings of contractual enforcement, generally speaking, this kind of trade is riskier (Schmidt-Eisenlohr, 2013, p. 7). It is essential to know that banks deal with the documents, not with the goods themselves (International Chamber of Commerce, 2007, Art. 5). This fact causes another problem: as the parties rely on these documents, it opens the opportunity for criminals to obtain money from banks with the help of fake documents (ICC Commercial Crime Services, 2019).

To make trade happen between two unknown parties, trust regarding payment and shipping of goods is essential. To mitigate risks, in traditional trade finance, banks offer various financial products such as LCs and guarantees to cover the payment and documentation stream (UBS, 2015, pp. 3f.). The risks can either be commercial, which means that the party receiving the products is not able to fulfil her or his obligation, or political, which are risks due to regulatory changes for import or export goods or a government's policy (Grath, 2011, pp. 18ff.). In addition, financial or currency risks have to be considered for an international trade deal. Short-term fluctuations of the agreed currency may make a deal more expensive to the buyer. Buyers also have to manage their liquidity through all trade phases; the risks here is that, because different payment stages, the financing may not continue with the chosen bank (Grath, 2011, pp. 18ff.). Another aspect is the enforcement of legal contracts, as it is much easier to take legal actions against the other party within the same country than across borders (Schmidt-Eisenlohr, 2013, pp. 7ff.). Furthermore, the standing and size of a company must be considered as factors of risk, since dealing with small firms is riskier than with large one's.

To ensure that standards are aligned, the Society for Worldwide Interbank Financial Telecommunication (SWIFT) sets up various standards for transmitting messages between banks and for the relationship between customers and banks. In the first case, a message type (MT) 700 streamlines the interbank communication and defines the required information to run a trade deal successfully (Dab et. al., 2016). For the second case, an MT 798 governs the interaction between customers and banks - for example, when someone applies for an LC (Fenyk, 2015). Thus, transaction costs are reduced. Integration of these steps into an organization's existing enterprise resource planning (ERP) system is advantageous from the perspective of efficiency but poses an expensive solution.

Nevertheless, today's process is susceptible to errors, since many different parties interact in a highly manual process, and it requires human actions such as verifying documents (Dab et. al., 2016). For that reason, the International Chamber of Commerce (ICC) adds a supplement, namely eUCP, for electronic presentation, to the known rules of Uniform Customs and Practice for Documentary Credits (UCP) 600 (Croner-i, 2019). They allow pure electronic presentation and are independent of any system require-

ments (International Chamber of Commerce, 2019*b*). Other than that, a mixed use with partial electronic and paper-based usage is also possible.
If a payment is delayed, it is difficult to manage the working capital in a firm (Cong and He, 2019, pp. 1769f.). Moreover, the payment section is isolated from the flow of goods, which we will later combine in our smart contract.

3.1 Roles and Responsibilities

In this section, we introduce the responsibilities of the various parties which participate in a typical trade finance transaction (Grath, 2011, pp. 47ff.). We begin with the most obvious parties: a buyer wants to buy an item and should make a payment to the offering party, who offers a good and should produce it (Brenton and Imagawa, 2005, p. 204). Additionally, the freight company's business model commits it to deliver the products from the seller's location to the buyer's, and the customs broker is responsible for checking whether a trade fulfils import rules and requires fees based on trade policy.
In discussing banks, we must to distinguish between various functions in the context of trade finance. We take these from the guideline UCP 600, the newest version from 2010, published by the ICC. A bank that offers a financial product such as a trade credit, it is called the issuing bank (International Chamber of Commerce, 2007, Art. 2). Moreover, this bank must take risks and must pay if the buyer defaults. The institution that functions as an advising bank checks the authenticity and conformity of the forwarded financial product from the issuing bank and enters into a relationship with the seller (Jones, 2018, pp. 197f.). The labels nominated or confirming bank are common, but their obligations differ from those of an advising bank. A nominated bank can be any bank that makes credit available but that does not have the obligation to pay in case of problems with the buyer's payment as shown in UCP 600 (International Chamber of Commerce, 2007, Art. 2). A confirming bank acts at the request of the issuing bank and verifies whether the details as stated in a trade credit are correct (International Chamber of Commerce, 2007, Art. 2).
Export credit agencies (ECAs) are public organisations which provide cred-

its to firms, who may not have a high credit line in order to participate in international trade (Asmundson et al., 2011, p. 5). Additionally, ECAs and insurance companies offer credit insurance to either the banks or the exporter to facilitate trade.

3.2 Payment Methods

This section deals with the different payment methods that can be used in the trade between the participating parties. For example, an issued credit cannot be cancelled without the agreement of the involved banks and the exporter (International Chamber of Commerce, 2007, Art. 18a). We further assess the benefits and drawbacks of each method and evaluate potential risks for both buyers and sellers in section 3.4. We focus our analysis on the four traditional, most commonly used payment methods as illustrated in the relevant literature (Giovannucci, 2007, pp. 3ff.) and take into consideration a quite new payment method, called bank payment obligations (BPOs), introduced by SWIFT (SWIFT, 2016).

3.2.1 Cash in Advance

This procedure involves a payment by the buyer that is made before the delivery of goods takes place (Grath, 2011, pp. 346f.). The method eliminates liquidity and commercial risk, which is particularly advantageous for the selling party (Grath, 2011, p. 35). As long as guarantees are absent, the buyer bears the risk that the seller will not meet her or his contractual obligations (Grath, 2011, p. 35). Cash in advance (CIA) is especially suitable if the seller has a high initial outlay to produce a tailor-made good (Grath, 2011, p. 35). In such a situation, the seller would suffer from an adverse strategic position: if the buyer cannot credibly commit to paying the stipulated payment, the seller will not enter the contractual relationship. This situation occurs because the buyer can renegotiate the price as soon as the seller has made a specific investment to produce a good (hold-up problem). The CIA principle poses one solution to this strategic problem.

3.2.2 Open Account

When the payment is conducted via OA, the seller ships the goods and forwards the documents to the buyer after signing a sales contract (Nordea, 2018). The buyer can pick up the goods by presenting shipping documents such as the bill of lading (B/L) to the port (Grath, 2011, pp. 5f.). The settlement of payment takes place on a due date in the future. To transfer the default risk of the buyer, the seller can use a trade credit insurance company (Grath, 2011, pp. 5f.). The method of OA is widely used when both parties have a common history in trading goods, since after shipment, the seller loses control over the products (Jones, 2018, p. 37ff.).

3.2.3 Letter of Credit

This irrevocable instrument lowers the risk of non-fulfilment of contractual obligations on the sides of both sellers and buyers because a bank acts as an intermediary (UBS, 2015, pp. 13ff.). If one party does not fulfil her or his obligations, the counterparty's bank pays the outstanding amount if the required documents have been presented and takes legal action against the respective party (Trade Finance Global, 2019*b*). In this connection, it is essential that the documents are in line with the agreed conditions (UBS, 2015). During the LC process, many contractual relationships between the participating parties arise (Niepmann and Schmidt-Eisenlohr, 2016). The graph in figure 1 illustrates the process and the parties involved when an LC is issued.[1]

[1] other LC variations are possible, depending on the agreed terms and conditions

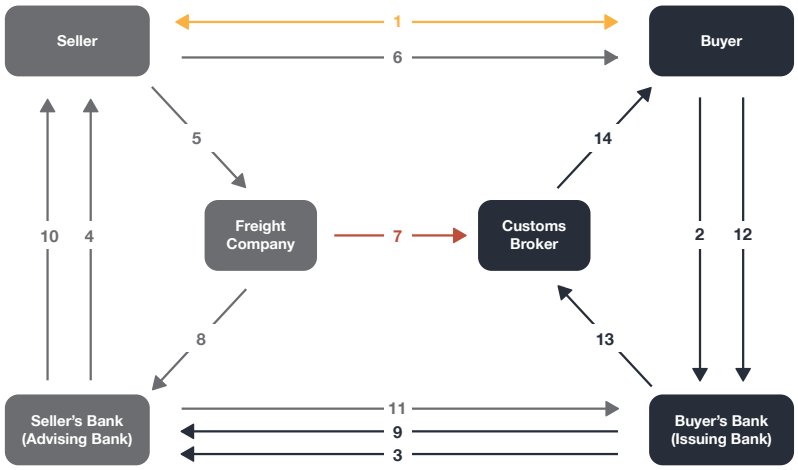

Figure 1: Mechanism of an LC transaction based on Giovannucci (2007)

As shown in figure 1, many different contracts, documents and payment flows are in place. Here, the numbers in the arrows indicate the chronology of events. Referring to the literature (Bergami, 2009, pp. 193ff.), the starting point is the sales contract between the seller and the buyer, which defines a business relationship to trade goods. As an LC stipulates the payment, the buyer will apply for it at her or his bank, and if she or he successfully passes the bank's risk assessment, the next contract between the buyer and her bank accrues (Giovannucci, 2007). The issuing bank gives security to the seller through a conditional, guaranteed payment, which also is a contract between these parties (Niepmann and Schmidt-Eisenlohr, 2016). After the issuing bank informs the advising bank about the LC's issuance, it informs the seller as well (Giovannucci, 2007). That is a signal for the seller to deliver the goods to the freight company and send copies of the documents to the buyer. Straightaway, the freight company sends the products to the customs broker at the buyer's destination and consigns the original documents to the advising bank. The advising bank then informs the issuing bank, which makes a payment to hands of the advising bank, which credits this amount to the seller's bank account. The next step is the transmission of the original documents to the issuing bank, which it sends to the customs broker after the buyer pays the outstanding amount (Gio-

vannucci, 2007). Finally, the buyer can pick up the goods at the customs broker. If the defined conditions from the LC are met in a specific time frame, the issuing bank pays and enforces the local law if the buyer does not pay (International Chamber of Commerce, 2007, Art. 7).
Moreover, if the seller requires additional security because of doubts about the credibility of the issuing bank, the seller can apply for second security at the advising bank (Grath, 2011, p. 53). This bank irrevocably ensures the payment from the issuing bank by its guarantee in the event of adverse circumstances such as political risks or insolvency, as long as, the determined documents are presented, as regulated in the agreement (Bergami, 2009, p. 194).
According to article 6b (International Chamber of Commerce, 2007), we define the different payment modes of LCs:

- *By acceptance*: The seller does not get the payment directly after submitting the documents to the bank but is paid at a later time.
- *By payment at sight*: After banks receive the required documents from the seller as stated in the LC, they complete the payment immediately.
- *By deferred payment*: When shipment is done, the seller is paid on a defined due date as stated in the LC.
- *By negotiation*: The buyer negotiates with the bank to get a discount on the fixed amount in the LC if the buyer presents the required documents.

Below is the required information which must be presented in an LC under the international guideline UCP 600 (International Chamber of Commerce, 2007, Art. 14) and the relevant literature (Grath, 2011, pp. 47ff.):

- *Period of validity:* Following this standard, all LCs are irrevocable and on behalf of the issuing bank. Within an agreed period of time, the documents must be presented to the issuing bank or to a nominated bank in the LC. In the case of a counterparty's default, the issuing or nominated bank have to fulfil the obligation.

- *Time of payment:* Every LC includes an expiry date by which a payment must be settled. Two options can be selected: at sight, which means that the seller receives the payment by presenting the required documents, or at a specified time. Furthermore, the parties must agree on one of these payment options: by acceptance (when handing over the documents) or by deferred payment (on a set due date).

- *Place of document presentation:* Documents must be presented at the issuing or advising bank.

- *Security level:* If the seller has doubts about the issuing bank's solvency, the advising bank can guarantee a payment in the event that the issuing bank is not able to fulfil its obligations. This information is not shared with other parties.

- *Documents to be presented:* The bank can reject the payment if not all required documents are presented within the given time period.

There also exist various versions of LCs, which are preferable under certain circumstances:

- *Transferable credit*: This kind of LC can be transferred by the seller to a third party, usually a supplier or any other cooperating company. Afterwards, the LC cannot be transferred again to others (Eker, 2019).

- *Revolving credit*: If two trading parties have repeating transactions about the same goods during a given time period - for example, for necessary goods for production such as screws - an LC may feature revolving credit, which can be renewed after the defined amount is used or the expiry date is reached (Eker, 2019).

- *Back-to-back credit*: For a supply chain with three parties, the payment is guaranteed by a first LC between the seller and the buyer. In addition, a second LC is opened by the seller in trust to the supplier with the first LC as a guarantor (Eker, 2019).

- *Red clause*: For this type of LC, a partial payout or the whole sum can be paid to the seller before the goods are shipped or any documents

are presented. The seller has to state that the credit will be paid, and the documents must be presented before the expiry date is reached (Credit Suisse, 2016).

- *Green clause*: To get a payment in advance for a green clause LC, the party must present evidence that the goods are stored in a warehouse, in addition to the documents from the red clause LC (Credit Suisse, 2016).

To sum up, an LC is a trade finance instrument with which banks act as intermediaries to mitigate the risks of default or of not receiving goods as defined in the contract for both the buyer and seller. The large amount of contracts between the parties increases the complexity of the process.

3.2.4 Documentary Collection

In this form of payment, a bank acts as a collector of documents, although unlike with LCs, the bank does not bear any risks such as political, transfer or commercial risks (UBS, 2015, p. 77). After the seller gives instructions for a trade transaction to the bank and all documents are collected successfully, the bank delivers these documents to the buyer (Kenton and Murphy, 2019). A DC includes the following documents: commercial invoice, bill of lading, certificate of origin and inspection certificate.[2]
We differentiate two versions of DCs: namely, documents against payment (D/P) and documents against acceptance (D/A) (International Chamber of Commerce, 1996, Art. 6). Figure 2 illustrates the process.

[2]further explanations in section 3.3

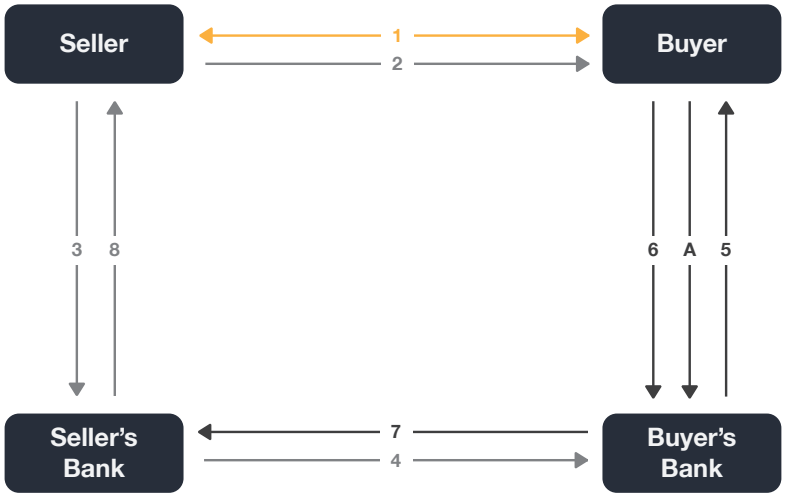

Figure 2: DC mechanism including D/A and D/P based on Jones (2018)

As in figure 2, the numbers in the arrows represent the chronology of the overall process, which starts with the negotiation of the trading terms between a seller and a buyer. After closing a deal, the seller sends the products to the buyer's destination, typically a port or an airport, and additionally sends the documents to the seller's bank, named as the remitting bank (Jones, 2018, pp. 159). The seller's bank sets the conditions of the DC and submits the documents to the buyer's bank, which is called the collecting bank, after which that bank notifies the buyer of the arrival (Credit Suisse, 2016, p. 85).

If the chosen mode is D/P, the buyer then has to pay before receiving the documents from the bank (Credit Suisse, 2016, p. 85). In other words, the buyer cannot get goods without paying beforehand. In the case of a buyer's default or unwillingness to pay, the potential risk for the selling party can be additional costs for storing the items in a warehouse or sending them back to their origin.

The second type, the D/A, allows the buyer to make a payment at a future date (Credit Suisse, 2016, p. 85). Here the buyer signs a bill of exchange (B/E), which is an instrument for short-term financing and normally due within 30 to 180 days. During this period, the seller bears the risk not

getting paid. To mitigate risks, the seller can demand for a bank guarantee or sell the B/E to forfaiter at a lower price (Credit Suisse, 2016, p. 85). After receiving the buyer's payment, the buyer's bank pays the transactional amount to the seller's bank and thus to the seller. The process of a DC is finalized after this step.
Now, we want to show the key differences between a DC and an LC. First, with DCs, banks are not liable for the content, whereas they are with LCs (Credit Suisse, 2016, p. 85). Second, with LCs, payment is guaranteed by a bank. Third, the LC costs more than the DC. According to Trade Finance Global (2019*a*), an LC costs on average between 1 and 2% of the total transaction amount, whereas a DC has lower costs (Goswami, 2012). More detailed information is provided in section 4.6.

3.2.5 Bank Payment Obligation

This new payment variant, developed by SWIFT, should reduce costs and risks compared to LCs and OA payments by transmitting and matching the contractual data electronically via a digital platform, namely the SWIFT Trade Services Utility (TSU) (SWIFT, 2016). As the ICC supports BPOs, the Unified Rules for Bank Payment Obligations (URBPO) No. 750 rules govern the procedure between banks, although not in their relationship to their customer (International Chamber of Commerce, 2013, p. 7). In this set-up, banks again act as intermediaries, but the two parties, the seller and buyer, directly exchange the shipping documents for releasing the goods at the port or airport (SWIFT, 2016). The core element of this process is the transaction matching application (TMA) unit, which checks the contractual input data such as orders, shipping documents or invoices of both parties automatically and, if they match, confirms the data for a trading deal. After this event, the buyer's bank has an obligation to pay the agreed amount at maturity (Büter, 2007, pp. 322f.).
The key difference between LCs and BPOs is the promise to pay: in the first case, it is a relationship between a bank and the buyer, whereas in the second case it is an interbank relationship (Büter, 2007). Moreover, according to Büter (2007), input data for the two instruments is handled differently, since for LCs banks check the documents for compelling content,

but for BPOs a match is enough, regardless of the content. Comparing the level of safety of the two instruments, LCs are safer due to the exchange of documents via banks and the handover against payment (Büter, 2007).

3.2.6 Standby Credit or Demand Guarantee

In general, standby credits and demand guarantees remain in the background and compensate the parties financially in the case of a buyer's default or a non-performing seller (Jones, 2018, p. 3). A standby LC works like a commercial LC with added guarantees (Jones, 2018, pp. 271ff.). The bank pays out a maximum amount, as defined beforehand, to the beneficiary when the beneficiary presents the relevant documents - for example, a B/L to the issuing bank. However, this circumstance can lead to unjustified payments, as a bank pays if the document meets the conditions of the standby LC. In another case, a double payment can happen due to a late payment by the buyer (Jones, 2018, pp. 271ff.). For continuing transactions over long periods, standby LCs usually have an automatic extension mechanism for an additional year, called an evergreen clause. With notification from the seller, or if no deviations from the trading contract occur and a transaction is completed, the standby LC expires.
Demand guarantees can be classified into two groups: unconditional and conditional (Jones, 2018, pp. 295ff.). For the first group, it is sufficient to claim a demand in the cases described in the previous paragraph. With conditional guarantees, the claimant must give evidence that the goods have not arrived or that the buyer is not able to pay (Jones, 2018, pp. 295ff.). The issuer of a demand guarantee is independent of the contractual agreements between the parties and is involved only if one party breaches the contract.

3.2.7 Bank Aval

A bank aval is a warranty that banks give related to their customers - for example, to guarantee the payment of a B/E in DCs to the seller (Credit Suisse, 2016, p. 85). If a trading counterparty demands higher security against financial risks, a bank can guarantee the payable amount with their

creditworthiness (Jones, 2018, pp. 177ff.). If a client defaults, the bank pays money to the seller, and the relationship between the buyer and the buyer's bank is as follows: the buyer's assets serve as collateral or a credit line to the bank. Also, the risk that the buyer does not want to or cannot pay is transferred to the buyer's bank. It gives the seller higher security after losing control over the goods, since the documents are handed over to the buyer's bank (Jones, 2018, pp. 177ff.).

3.2.8 Factoring and Forfaiting

Factoring and forfaiting secure immediate liquidity to either the seller or buyer, depending on the method, when the maturity date of the payments is in the future. In the case of factoring, the seller sells her or his outstanding money to a factor, who pays less than the original amount stated in the invoice - for instance, 80 % (Klapper, 2005, pp. 6f.). The outstanding 20 % will be paid to the seller after the factor receives cash directly from the buyer reduced by interest rates and fees. With this instrument, the seller shifts many hazards to the factor (Klapper, 2005, pp. 6f.). Furthermore, we have to differentiate between recourse and non-recourse. The first means that in the case of a buyer's default, the factor has a claim against the buyer and can force it to court (Klapper, 2005, pp. 6f.). The second situation does not have this functionality, which makes it overall riskier.
One particular case of factoring is reverse factoring, which focuses on the obligations rather than the claims (Swissbilling, 2019). For instance, company A buys goods from another firm B, then the factor pays the invoice, which A receives immediately, and issues a new invoice to A with a future date of payment (Swissbilling, 2019).
Forfaiting is often used for medium and long-term financing (Silitschanu, 2019). The key differences compared to factoring is that the seller gets up to the full amount of the transaction volume immediately and forfaiting deals with financial instruments instead of accounts receivables (Silitschanu, 2019).

3.3 Documents

The documents, as stated in sections 3.2.3 and 3.2.4, are essential to concluding a trade transaction successfully. The necessary information in each document should match the documents from the trade goods according to the guidelines - UCP 600 for LCs and URC 522 for DCs. Otherwise, a transaction will be delayed or cannot be completed (Ellinger and Neo, 2010, p. 246). The match is of great importance in the case of one party's default or unwillingness to make a payment, leaving banks with the obligation to pay. The following documents are significant for international trade.

3.3.1 Commercial Invoice

The legal requirements for a commercial invoice are outlined in article 18, UCP 600. The seller who issues an invoice must provide information about herself or himself and the buyer on this document (International Chamber of Commerce, 2007, Art. 18). When the payment happens with regard to an LC, the descriptions in the LC and the commercial invoice must match, and the indicated currency must match (International Chamber of Commerce, 2007, Art. 18). Other information about the buyer can be neglected.

3.3.2 Bill of Lading

The rules for one of the most important trading documents, the B/L, are defined in article 20, UCP 600. According to this article, a B/L shows details about a good, such as type, quantity and destination, and must be signed by a representative of the shipping company, such as the carrier, agent or master (International Chamber of Commerce, 2007, Art. 20). We have to differentiate between non-negotiable and negotiable B/Ls (Raunek, 2019). Non-negotiable means that the goods can be delivered only to the party named in the bill, whereas with negotiable bills it is possible to deliver goods to a third party as written in the document. A negotiable B/L serves as an additional to a proof of the contractual relationship between the seller and the freight company and as a goods receipt as a title of ownership of the goods. On the contrary, non-negotiable B/Ls fulfil only the first first

mentioned function (Wax, 2018). For example, non-negotiable B/Ls are straight or seaway B/Ls; seaway B/Ls are used for goods transported by ships. This type of B/L is a contract between the freight company and the customer rather than a document to clarify ownership (Ellinger and Neo, 2010, pp. 255ff.).
With the help of digital platforms such as Bolero,[3] the issuance and exchange of electronic B/Ls (eB/Ls) should expedite proceedings (Dab et. al., 2016). Moreover, eB/Ls can accelerate the payment cycle and improve the working capital situation for the seller, since the payment occurs more quickly. As the B/L serves as a title of ownership, it must be unique and secure (Tricks and Parson, 2018). Additionally, it should guarantee that there exists exactly one owner at a time. The legal status of an eB/L is, however, not clear, since rules about the use of eB/Ls do not exist.

3.3.3 Bill of Exchange

When the trading parties agree to pay by DC and further determine on D/A, then the use a B/E is essential. It allows the buyer to pay at a later stage, typically between 30 and 180 days, after presenting the required documents for an after-sight bill and at any specific date for a time bill (Credit Suisse, 2016). After releasing a B/E, the seller carries the risk of a buyer's non-payment, since no bank guarantees the B/E (Credit Suisse, 2016). With the help of bank guarantees, this risk can be eliminated.

3.3.4 Certificate of Origin

This document, as the name suggests, verifies the origin of products for international trade (Credit Suisse, 2016). The basic information includes country of manufacturing, country of destination, details about the exporter, and a detailed description of the product (Federal Customs Administration FCA, 2019).
We have to distinguish two types of certificates of origin: non-preferential and preferential (Schweizerischer Bundesrat, 2018). Exporters use the non-preferential type if there is no free trade agreement between two countries,

[3] detailed information, see section 4.7

while the preferential type is used when countries have negotiated a free trade agreement, as it allows importers to take advantage of reduced or free import tariffs.

3.3.5 Insurance Certificate

The insurance certificate directly shows to all involved parties that the freight is insured against damages or losses (Hinkelman, 2008, p. 293). Requirement for buying an insurance policy is a party who has an interest to insure goods, e.g. through a cargo insurance (Hinkelman, 2008, p. 293). Depending on the shipping mode, either the seller or the buyer has an obligation to insure the shipment (International Chamber of Commerce, 2011). Generally, the seller has an interest in having an insurance until the payment is received, especially if the parties agreed to an OA payment.

3.3.6 Inspection Certificate

Before goods are shipped, an independent inspection authority checks if the products are in line with the information given in the LC documents (Jones, 2018, p. 443). The inspection certificate is essential for a bank when it comes to a default by the buyer, because it verifies the condition of the goods, so that the bank can track the real value (Jones, 2018, pp. 442ff.). As banks only deal with documents, it is important to define the correct parameters in the credit. Moreover, this document is important for securing the quality, quantity and characteristics of trading goods and hence reduces the risk that the buyer will receive low-quality goods different from what have been specified in the contractual agreements (Jones, 2018, pp. 220ff.). Some countries also require additional documents, such as a packing or weight list, which are not covered by the UCP 600 rules.

3.4 Risk Evaluation

In this section, we analyse the risk for the trading parties with regard to the agreed payment methods. We also have to consider that different shipment methods, which are defined in the Incoterms 2010, carry various risks for

either the seller or the buyer, which can be covered by insurance companies (International Chamber of Commerce, 2011). Obviously, both parties are interested in dealing successfully with each other. Figure 3 shows the risk allocation for the different payment methods.

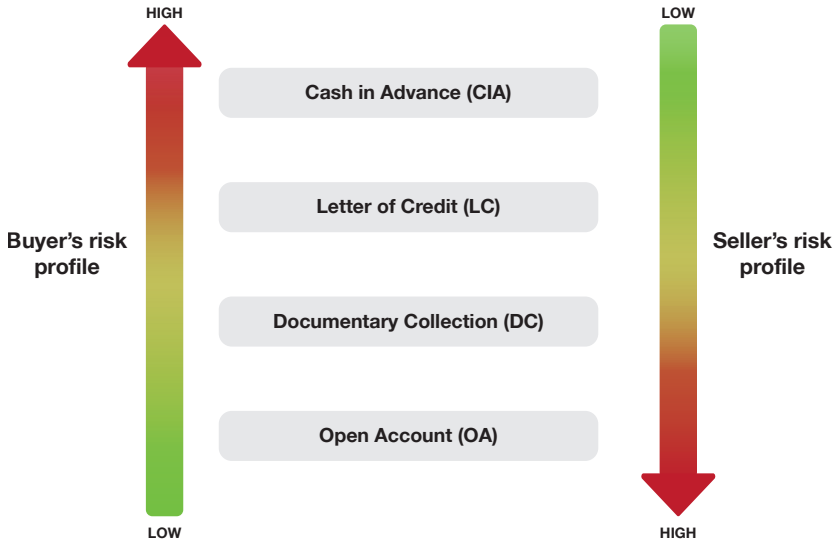

Figure 3: Risk distribution for different payment methods in trade finance based on Jones (2018)

In the case of CIA, the buyer carries all risks because the payment is finalised before the buyer receives the goods (Jones, 2018, p. 4). The opposite issue happens when the parties agree to an OA: the seller then bears high risks due to potential non-payment by the buyer. In other words, CIA can also be called buyer or importer finance, since the buyer pays in advance (Schmidt-Eisenlohr, 2013, p. 2). OA is then called seller or exporter finance, since the seller delivers the goods before receiving a payment from the buyer. In addition, OA is advantageous for the buyer, as faulty products can be returned more easily and cheaply than with a CIA payment, which would require the buyer to claim reimbursement (Lee and Stowe, 1993, p. 286). The other solutions in form of LCs or DCs balance the risks between buyer

and seller equally, due to the participation of banks as intermediaries which are better able to manage risks. LCs are safer for the seller than DCs, which do not cover the risk of a rejected payment or buyer liquidity problems (Credit Suisse, 2016, pp. 6ff.). DCs, in turn, are more favourable for the buyer because they are cheaper and do not require a credit line; payment follows as soon as the goods arrive, including all documents that verify ownership (Jaffeux and Wieser, 2012, p. 266).

3.5 Game Theoretical Analysis

The choice of payment methods for a trading contract depends on the point in time at which one party carries higher risks and the other party has greater bargaining power. In this section, we analyse whether one party has an incentive to deviate from an agreed contract in order to reach a higher expected payoff. We assume all players to be rational and risk neutral.
First, if the buyer has the choice whether to pay or not, the seller chooses to deliver or not. Based on the equations from Onderstal et al. (2014), we start with the model set-up for a CIA one-shot game; thus, after one transaction is completed, no further trades occur:

$$U_B = v - p, \text{ with } v = ae \tag{1}$$

$$U_S = p - C(e), \text{ with } C(e) = be^2 \tag{2}$$

For the efficient outcome, both players maximize their utility functions, and hence we get the value-maximizing effort levels (Onderstal et al., 2014, p. 233):

$$\max_e U_B + U_S = ae - p + p - be^2 \tag{3}$$

$$e^* = \frac{a}{2b} \tag{4}$$

$$U_B + U_S = ae - be^2 = \frac{a^2}{4b} \tag{5}$$

As stated in equation 5, the maximal total gains from trade are $\frac{a^2}{4b}$. Following the argumentation of Onderstal et al. (2014), we suppose that both parties agree to an equal share of the gains before trading. This leads to an efficient price p:

$$a * \frac{a}{2b} - p = \frac{a^2}{8b} \tag{6}$$

$$p = \frac{3a^2}{8b} \tag{7}$$

In the case of equation 7, trade happens and the buyer pays $p = \frac{3a^2}{8b}$. In the situation of CIA, the seller has the full bargaining power and hence can renegotiate after receiving payment (Onderstal et al., 2014, p. 232). Therefore, the seller sets the utility of the buyer U_B to zero. It can be shown that in the first option, the seller keeps the optimal price constant, which leads to less effort for the production of the good, thus the quality decreases. The second option is to keep the effort level constant and increase the price for the buyer. Following equation 1, the new price is $p = \frac{1}{2} * \frac{a^2}{b}$.

The game table for a one-shot game using CIA is as follows:

		Seller	
		deliver	*not deliver*
Buyer	*pay*	$(v - p, p - C(e))$	$(-p, p)$
	not pay	$(v, -C(e))$	$(0, 0)$

Table 1: Payoff matrix for a CIA one-shot game

The Nash Equilibrium (NE) is (not pay, not deliver). Because the buyer can anticipate the seller's behaviour after paying in advance without getting the products immediately, no trade occurs. Thus, the seller is also worse off in the absence of trade and is better off if she or he can reliably commit not to renegotiate after the buyer pays.

Now we consider the same set-up as before, however, we introduce repeated interaction, in which both players desire a future business relationship as

well. Referring to table 1, we can prove by the net present values (NPV) formula whether deviating yields a higher discounting factor or not (Onderstal et al., 2014, p. 142). We consider a grim trigger strategy of the counterparty: as long as both participants cooperate, they receive the respective payoff π^C. As soon as one player deviates from this strategy, that player once obtains the payoff π^D, which is usually higher than the cooperative payoff. In the subsequent periods, both parties receive the NE payoff, since cooperation is no longer feasible. The player's discount factor for future payoffs determines whether it is sensible to deviate from the cooperative strategy.

$$\begin{aligned} \pi^C &= p - C(e) \\ \pi^D &= p \\ \pi^N &= 0 \end{aligned} \tag{8}$$

$$NPV^C \geq NPV^D \tag{9}$$

$$\delta \geq \frac{p - p + C(e)}{p - 0} = \frac{C(e)}{p} \tag{10}$$

From equation 10, we see that if the δ from the seller is higher than $\underline{\delta}$, the fear of being punished by the buyer is high enough to make the seller not deviate (Onderstal et al., 2014, pp. 142f.).

Second, we examine the case in which OA payment is agreed to. The seller acts first, followed by the buyer:

		Buyer	
		pay	*not pay*
Seller	*deliver*	$(p - C(e), v - p)$	$(-C(e), v)$
	not deliver	$(p, -p)$	$(0, 0)$

Table 2: Payoff matrix for an OA one-shot game

As in the case of CIA, the NE again is (not deliver, not pay). For this reason, trade does not occur if both parties agree to OA in a one-shot game setting.

Third, we suppose the case in which an LC serves as a hedge against financial loss, so that banks also participate in this game. The parameter λ (with $\lambda > 1$, because of higher enforcement and legal costs) indicates that the bank pays in the case of a buyer's default. Illustrated in a payoff matrix, the game is as follows:

		Buyer	
		pay	*not pay*
Seller	*deliver*	$(p - C(e), v - p)$	$(p - C(e), v - \lambda p)$
	not deliver	$(0, 0)$	$(0, 0)$

Table 3: Payoff matrix for an LC one-shot game

The unique NE from the payoff matrix in table 3 is (deliver, pay). Now we compare the cooperative payoff with the case of a deviating.

$$\frac{\pi^C}{1-\delta} \geq \pi^D + \frac{\delta \pi^N}{1-\delta}$$

$$\delta \leq 1 \tag{11}$$

Equation 11 shows that for values $\delta \leq 1$, cooperation is always feasible. This also holds true in a one-shot game, since it poses a cooperative NE. We conclude that a financial instrument such as LC is incentive compatible. Schmidt-Eisenlohr (2013) includes in his model the enforcement probability of contracts λ and the interest rates in the seller's country r and the buyer's country r^*. His analysis for an one-shot game reveals that both CIA and OA depend on the probability of enforcement (Schmidt-Eisenlohr, 2013, p. 8). Furthermore, in an LC transaction, these parameters do not influence the optimal result, as banks are involved and bear the risk. To make deviation from the equilibrium path less attractive, repeated interaction with the ability for punishment is needed. In the case of CIA when the buyer pays the agreed amount but the seller does not deliver the goods once, both

will start into a business relationship proportional to λ (Schmidt-Eisenlohr, 2013, p. 8). This is also valid for OA, where the buyer can deviate by receiving the goods but does not making a payment (Schmidt-Eisenlohr, 2013, p. 9). Nevertheless, the situation remains unchanged for LCs, since they were already incentive compatible.

4 Smart Contracts for Trade Finance

This chapter deals with the analysis of how blockchain technology in the form of smart contracts can improve the overall trade finance process. Instead of sharing the aforementioned documents (section 3.3) in a non-transparent way, with the help of blockchain, the trade inputs can be saved in decentralised fashion and shared in real time; each party has the same information at the same time. The increased transparency makes the process faster, and if designed correctly, blockchain technology can reduce the motivation for misconduct (Cong and He, 2019, p. 1756).
Using a public blockchain such as Ethereum for trade finance transactions is problematic in terms of privacy issues, since everyone can participate without proof of identity and everyone is able to track a wallet's transaction history, including the correspondent input data (Cong and He, 2019, p. 1762). Given this, we encrypted some of our inputs using the one-way hash function (Cong and He, 2019, p. 1762).
For our smart contracts in section 4.1, we suppose that the owner knows the roles and the corresponding public addresses of each participant. In addition, we assume the financing party to have information about the buyer such as creditworthiness. With these assumptions, we comply with the anti-money laundering (AML) guidance for blockchain, issued by the *Eidgenössische Finanzmarktaufsicht* (FINMA), which requires entities to know their opposite parties when trading with each other. Any transactions with an anonymous party are forbidden (Mathys, 2019).
In comparison, many companies prefer to utilize permission blockchains such as Hyperledger Fabric, because their scalability and performance is much higher than that of public blockchains (Blockgeeks, 2019). One difference in permission blockchains is that not all nodes have to confirm the validity of a transaction (Cocco and Singh, 2018). In the case of trade finance,

transactions per second are not as critical as in other use cases.
Here we compare blockchains to databases, which are managed and controlled by administrators, often a small group of people (Tabora, 2018). The fact of centralisation makes databases vulnerable to attacks; for example, BPOs rely on databases only for the matching of data (International Chamber of Commerce, 2013). Moreover, when using eB/Ls in the future, it is crucial to replicate the three functions of a paper-based B/Ls for legal reasons and save them centrally, since replication is possible (Tricks and Parson, 2018). With the help of DLT, the problem of replication could be solved, because with DLT, once a transaction is valid, it is immutable.
As mentioned in chapter 3, the supplement to the UCP 600 guidelines, eUCP, manages the handling of electronic documents. Under article e5 (International Chamber of Commerce, 2019*b*), if not specifically mentioned different, no requirements concerning the format are needed (free choice). If a bank would accept a submission via a blockchain transaction, it would be suitable.

4.1 Contract Design

We discuss one possibility for how to use a smart contract for a trade finance transaction.[4] In our solution, we assume no other contract[5] than the smart contract exists. We also note that the absence of a physical contract can lead to legal complications, as we show later in section 4.5. Moreover, we suppose the payment to take place in Ether and that every party is responsible for managing its own local currency conversions to Ether.
Our main smart contract replaces the banks' function as risk manager in traditional LC deals because it takes over an escrow role. To make a payment or to receive money for a fulfilled service, parties interact only through the smart contract. The ex-ante mistrust of transacting parties is solved by the defined code, since misbehaviour, except from bugs or hacks, is impossible. This situation creates trust between totally unknown parties, independent of their locations.
The auxiliary *Whitelist* smart contract serves as a gatekeeper, because if

[4]Extensions with multiple parties and parameters are possible
[5]here in the sense of a physical contract

financiers who want to participate must meet certain conditions; in our particular case, the wallets' balance has to exceed a required minimum amount and score.
Given the results of the section 3.2, we use a hybrid of CIA and OA as the payment method; further explanations follow in section 4.1.2.

4.1.1 Whitelist

The whitelist contract admits wallet addresses that want to finance a deal in our second smart contract. We assume a know your customer (KYC) check has been completed beforehand and the identities behind the wallet addresses are known by the buyer. All other involved parties do not need to know the identities, since the rules implemented in the smart contract manage the payment stream. As the buyer needs funding for her deal, we assume that she can whitelist addresses. By setting the minimum requirements – that is, a balance that is clearly higher than the required funding amount for the trading deal – she or he is able to control who can join the whitelist. In addition, the applicants must exceed a specific score, which is similar to an investment grade. Only financiers who fulfil the conditions of a sufficient wallet balance and score will be approved.

4.1.2 Trade Finance

The function of this smart contract is, as outlined previously, to manage the funds and make payments after certain events occur. Figure 4 illustrates the scheme of our main *TradeFinance* smart contract.

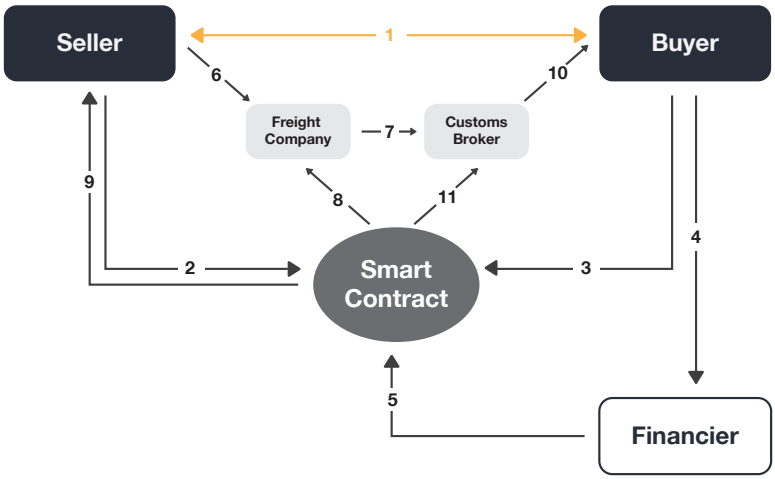

Figure 4: Process flow of a smart contract-based transaction

Before the smart contracts gets deployed, both parties the seller and the buyer negotiate the terms and conditions for their trading agreement in $t = 0$. After finalizing this step, the seller deploys the Smart Contract and adds an order in $t = 1$, including the agreed parameters. If the buyer agrees to the conditions, she or he confirms the order. As she or he knows the value of the trade, she or he sets this amount equal to the minimum amount in the whitelist to search for an appropriate candidate. The fastest financier adds a payment by sending the funds to the smart contract's address. All participating parties can observe that the payment is guaranteed by the financier and will be paid out following the rules of the contract. The seller prepares the goods to be ready for shipment and hands them over to the freight company. When the goods arrive at the freight company, it confirms the receipt of the goods. To prevent this company from keeping the products and not sending them to the next station, the customs broker, the payment takes place after a successful transfer. The same logic applies for the customs broker and the seller is paid by the smart contract after the goods arrived there. Afterwards, the buyer receives the goods and confirms this event. We assume that at $t = 2$, after the transaction through the

smart contract has taken place, the seller can deposit her or his money in the bank or invest it alternatively. Furthermore, the buyer can negotiate individual conditions concerning her or his payback of the financier.
We show that the involved parties have no incentives to deviate from the envisaged rules. Starting with the seller, she or he does not have an incentive not to hand over the goods, since she or he wants to earn money. For the buyer, not paying is impossible, since the financier's payment is a necessary action before the seller sends goods to the buyer. Not confirming receipt of the order is also not an option, since the customs broker does not get the fees. If the buyer decides not to pay the financier, the financier can enforce their rights in court. Both the freight company and the customs broker do not have any incentive to not deliver their service as expected, given the reputational risks, and they do not gain any advantage from a deviation. The same holds true for the financier, who sees no advantage to not paying on behalf of the buyer except the loss of interest payments.

4.2 Smart Contract Development

Proceeding with the technical details, in this section we explain the main functions based on Solidity code extractions,[6] which are essential to run our smart contracts.[7] We use Solidity version `^0.5.11` as the development environment. To avoid an overflow in arithmetic operations, we make use of the *SafeMath* library from OpenZeppelin (2019).
As in section 4.1, we begin with the *Whitelist* smart contract. We decided to separate this contract from the *TradeFinance* contract to ensure that the whitelist can be updated without generating conflicts and we can reutilise our main smart contract. Since we suppose that the buyer needs funding, all functions in the *Whitelist* contract are restricted to the buyer. We expect that if the *addWhitelist* function is open to everyone, the given requirements deny inadequate wallets, but the applicant has an incentive to enter an artificially high score to get access to the whitelist.

[6] for full code, see Appendix
[7] developed with Solidity documentation of Ethereum Community (2019*b*)

The following main functions are important to run the smart contract. We analyse each of them in more detail.

```
function setMinimumRequirement(uint256 _minimumAmount, uint256
↪  _minimumScore) public onlyBuyer
```

This above function allows the buyer to set a minimum wallet amount and score that an applicant – who can be anyone – must fulfil to be accepted to the whitelist.
Further, the functions *addWhitelist* and *removeWhitelist* allow the buyer to manage the whitelist by setting whitelisted addresses to true and others to false. In this context, remove means changing the status of an address rather than deleting it.

Moving to the essential functions of the *TradeFinance* contract, we first explain more about its structure. We use the *import* keyword to make use of the functions from *Whitelist* smart contract. As the constructor is called exactly one time when the contract is deployed, in our case the *msg.sender* is the seller, who is equivalent to the owner. We use *structs* to define our unique data types such as *Order* and *Guarantee*. Moreover, by implementing modifiers and assigning different roles to a wallet address, we can restrict functions to those who are eligible. The following *enum* is important for tracking an order's state.

```
enum Orderstate { Negotiation, Created, Locked, Customs,
↪  Received, Cancelled }
```

Based on the order's state, functions in our smart contract are executable or not. This is a security element to guarantee that fallbacks cannot occur – that is, if the buyer accepts the proposal from the seller, the smart contract changes its state from *Negotiation* to *Created* to ensure that the seller cannot make changes afterwards.

With the next function, the seller manages access for participation and sets a number as salt,[8] to make the smart contract more secure. Moreover, she

[8]A salt is a random number, further explanation follows later in this section

or he enters the *orderAdress* of type bytes32 to have an unique identifier for each order, which she or he declares in the function *addOrder*.

```
function access(address payable _buyer, address payable
↪  _freight, address payable _customs, bytes32 _orderAddress,
↪  uint256 _salt) public onlySeller
```

The seller inserts the public wallet addresses of the buyer, freight company and customs broker so that the matching of addresses and functions is clear. We assume that all parties know about their own roles due to the bargaining before the set-up of this contract. As type we use *address payable*, because this allows the addresses to receive Ether. This function is also essential because the smart contract contains restricted functions which can be executed only by the defined party. Moreover, we require an order state of Negotiation; otherwise, the seller would be able to change the roles during the process.

```
function addOrder(bytes32 _orderAddress, address payable
↪  _seller, address payable _buyer, bytes32 _priceSeller,
↪  bytes32 _quantitySeller, uint256 _weight, string memory
↪  _productname, uint256 _freightrate, uint256
↪  _orderAmountSeller, uint256 _customsDuty) public
↪  onlySeller
```

The function *addOrder* allows the seller to propose an order, including details such as order address, price[9] and quantity. Further, we note that the freight rate[9] is a fixed amount, whereas the customs duty uses a percentage notation. We implement this function so that the seller can make use of this smart contract more than once for different orders. If we assigned specific values to the parameters before deployment, the contract could not be used multiple times. To allow searching on order details with the unique order's address, we mapped the struct Order.
Since all data the seller enters into the function above are publicly observable, rather than using plain-text numbers we use the corresponding hash

[9]currency is ETH

values. Potentially all inputs can be hashed, but since we need some inputs to calculate with, we demonstrate the principle by hashing the price and quantity parameters. For this, the seller hashes the price and quantity off-chain with the help of the one-way hash function and feeds these values to the order. In addition, he must add $0x$ as a prefix, because the hashed value is of the type bytes32, but the transaction requires a hexadecimal format. However, this solution leaves room for brute force attacks; malicious persons could try guessing the hashes of numbers for both the price and quantity until they succeed (Hornby, 2016). A more secure pattern is the use of a salt; however, the addition of this random value, as a pre-fix or postfix, cannot guarantee the highest security. Nevertheless, the exponential increase in computing time resulting from the salt makes a brute force attack less feasible than using hash values only.

In case of a false input, the seller can revise the order if the order state is still *Negotiation*. If the data are correct, the buyer confirms the inputs by using the function *confirmOrder*. To do so, she or he enters the hash values of the agreed parameters – here the price and quantity – along with the total order amount and the order address to be aligned with the seller's order. If the inputs match, the event *OrderConfirmed* is invoked and leaves a comment for informational purposes. If the buyer's data differ from the seller's, however, the event *OrderCancelled* is initiated, and the seller can reset the contract's parameters to use it for a new order.

The next function *addGuarantee* is essential, since at this point the financier transfers Ether to the smart contract *TradeFinance*. We do not set any restrictions to execute this function; if a financier has been whitelisted, we leave it open on a first-come, first-served basis.

```
function addGuarantee(bytes32 _guaranteeAddress, bytes32
↪  _orderAddress, address _to) public payable
```

If we import functions from another smart contract, we have to trust the opposite contract entirely, which is riskier when we have not written it ourselves. Now the smart contract escrows the funds and a payout to the

various parties following the defined rules. As mentioned earlier, we let the financier enter the order number again to have a match between the order and guarantee. If all requirements are fulfilled – for example, the financier amount is at least equal to the agreed transaction value – the two events *GuaranteeActive* and *OrderLocked* change the smart contract's state. The following functions can then be executed and the participating parties informed.

The auxiliary functions *isOrder* and *isGuarantee* offer the opportunity to check whether an order or guarantee is valid. The contract's balance can also be queried using the eponymous function.

The next function, *receiveOrderFreight*, is used by the freight company to signal that the seller has delivered the goods and to inform the other parties about the shipping details.

With the following function, the customs broker confirms receipt of the order from the freight company.

```
function receiveOrderCustoms(bytes32 _orderAddress) public
↪  inOrderState(OrderState.Locked) onlyCustoms returns(bool)
```

As mentioned previously, entering the order address it ensures the right match for each order. We invoke the event *OrderReceivedCustoms* to change the smart contract's state to *Customs*, which is a precondition for the next step, the transfer of the fees to the freight company and the order amount to the seller. We then check whether the contract's balance equals the agreed amount as protection against function recalls from both. If the balance is accurate, we use *transfer* instead of *send* or *call.value* because transfer throws an exception in the event of a failure while *send* and *call.value* return false (Ethereum Community, 2019*b*). Moreover, *transfer* and *send* are safe against reentrancy attacks, since they limit the gas to 2,300, while *call.value* does not (Schütz and Fertig, 2019, pp. 396ff.). With reentrancy attacks, an attacker tries to gain more funds than appropriate by taking advantage of calls to other contracts and execute functions which were not intended by

the smart contract owner (Schütz and Fertig, 2019, p. 396). A well-known example is the DAO, where attacker have stolen a third of the funds (Smith, 2018).

The function *receiveOrder* works in a manner similar to the previously described function *receiveOrderCustoms*. The customs broker get its payment. To calculate the customs payout, we use the *SafeMath* functions from the identically named library. In our case, since we restrict the function to be executable only by the buyer, it is safe against reentrancy attacks because the buyer receives no funds and thus has no motivation for such an attack. Moreover, the contract's balance after this function is called equals zero.

With the last function of the smart contract, *TradeFinance*, we can reset the addresses of the participating parties for this particular order.

```
function reset(bytes32 _orderAddress, bytes32
↪  _guaranteeAddress) public onlySeller
```

Only the seller can use this function, as she or he is the smart contract owner. It requires that the buyer has confirmed receipt of the order and that the contract's balance is zero.

4.3 Oracles

If data that are not available on the blockchain – for instance, the actual tracking position of a parcel – are necessary to run a smart contract properly, the situation creates a dependency between the provider of the data, the Oracle, and the smart contract (Berentsen and Schär, 2017, pp. 294ff.). First, the use of one Oracle contradicts the blockchain principle of independence from third parties because it acts as a central party. Such usage would also require full trust in the data inputs, since anyone with malicious intent could attack the Oracle to redirect the smart contract's execution (Schütz and Fertig, 2019, p. 357). Second, the Oracle could manipulate the data in such a way that the execution returns a wrong result or becomes stuck such that the funds stored in the contract's address could never be

paid out because a transfer depends on these data (Berentsen and Schär, 2017, pp. 294f.).
Furthermore, it would be problematic if an Oracle shut down during a contract's execution. Overall, Oracles lead to risks of manipulation, which stands in contrast to blockchain principles. The use of multiple Oracles, in which the majority have to confirm the same input, can avoid the risk of data manipulation and a lock of funds (Berentsen and Schär, 2017, p. 295).

4.4 Integrity of Goods

One crucial point in our smart contract process flow is the inspection of the integrity of the goods, which are shipped off-chain in the real world. The inspection's difficulty heavily depends on which class of goods a seller sends to the buyer; for example, it is easier to check the condition of products which have integrated sensors than it is to check commodities. Several inspection parties are needed to check the status of goods; this violates one of the core principles of blockchain: namely, the independence from third parties (Ganne, 2019, p. 111). Use of third parties could allow some parties to have an advantage, depending on the contract conditions. Also, the execution with involved third parties is highly inefficient; it can cause delays in the supply chain of firms and is expensive (Wang, 2019). With the help of clear attributes such as product serial numbers or radio-frequency identification (RFID) chips, it is possible to identify goods and transfer a digital copy to the blockchain (Tian, 2016, pp. 65ff.). Nevertheless, if the state of a product changes, it is impossible to depict this without external help.
Internet-of-things sensors can help solve this problem. For instance, the company Smart Containers has developed shipping containers with various sensors to transport pharmaceutical products safely around the world (Lee, 2018). Small changes in temperature or humidity are very critical to these products. Another solution could be the use of digital twins; these are digital copies of a physical good, where the implemented sensors gather data on any changes and report the status of the product in the digital version (Sallaba, 2017).

4.5 Legal Classification

Here we discuss whether a smart contract has a legally binding character as, for example, a sales contract. First, the primary function of a smart contract is to ensure the performance once it is started without being dependent on enforcing a contract, and it is verified by the network (Raskin, 2016, pp. 309ff.). Moreover, no trust is needed between two transacting parties, because the written code executes independently when certain conditions are fulfilled. Additionally, DLT guarantees independence, since all transactions are transparently saved on the blockchain.
With a classical contract, in the case of problems such as lack of fulfilment or the occurrence of mistakes, the parties can go to court or renegotiate to find a solution (Raskin, 2016, pp. 319ff.). This is impossible in an environment of smart contracts, as they cannot be changed once started. Both the obligations and duties are defined clearly in a smart contract without any chance for deviations, as "code is law" (Savelyev, 2017, p. 130).
Furthermore, the language in computer science is precise and thus does not allow room for interpretation in comparison to natural languages (Raskin, 2016, pp. 323ff.). Also, if the code may not behave as expected by the programmer, it is easier to predict a possible outcome than with a human's understanding of a dispute. The agreement phase on the contractual details and conditions in smart contracts are very similar to those in traditional contracts, but instead of signing a contract, a smart contract is accepted by a party through effective action – for example, sending Ether to the contract or entering some mandatory input - as in our *TradeFinance* smart contract. The execution of a smart contract reaches its end when the performance by a party is fulfilled (Lipton and Levi, 2018). What happens, however, if the outcome does not satisfy the opposite party? Smart contracts do not allow renegotiation afterwards (Raskin, 2016, p. 311). This point is critical, especially for individually made goods – for instance, suits or sculptures – since the expectations of buyers can differ significantly.
Following the argumentation of Gyr (2019), whether smart contracts in the form of a programming language can be seen as enforceable contracts in court under Swiss laws mainly depends on both parties' willingness to fulfil their obligations, since the *Obligationenrecht* (Swiss Code of Obligations) is based on the principle of contractual freedom (Gyr, 2019, pp. 108f.).

Generally, it is impossible to identify the parties participating in the public blockchain Ethereum, and thus their legal capacity is not ensured (Gyr, 2019, pp. 110ff.). Since we neglect the point of anonymity in our smart contracts, at least a business's legal capacity is given. If it is possible to verify a person's legal capacity, then smart contracts are in conformity with Swiss laws. In our case, we indeed have assumed all parties know each other's public wallet address, but their real identities and who conducts the transaction cannot be seen from that point. According to Gyr (2019), we have to differentiate between persons who are familiar and able to read a smart contract and those who are unable to do so. For the first group, by using a smart contract, their will is expressed directly in this contract, whereas for the second group, the use of a smart contract lacks the intention to create legal relations. Also important is the fact to consider an option in the smart contract's design to allow the opposite party whether to accept the proposed conditions or not. A party who agrees to a contract expresses her or his legal will. Compared to classical contracts, smart contracts cannot be declared invalid, since they are stored in a blockchain and thus cannot be revoked once execution is started (Gyr, 2019, pp. 140ff.).
As we have the financier issue a guarantee by paying Ether to the smart contract, it is not a legally binding contract, since the smart contract itself is not a legal entity and fails to produce the required written form for a valid deposit contract (Gyr, 2019, p. 206).
Finally, we examine the question of what happens if a smart contract contains a bug or leads to a malfunction due to the wrong input. Gyr (2019) concludes that someone is only responsible only when she or he acts with wanton negligence.

4.6 Cost Comparison

To have an overview of the costs from different payment methods in a trade finance transaction, we suppose a deal volume of 100,000 CHF. We neglect CIA and OA, since both these methods do not have service fees except the market interest rates to finance the deal for either the seller or buyer.

However, we analyse the cost structure[10] for LCs and DCs based on information from various banks (BNP Paribas, 2019; Credit Suisse, 2013; Deutsche Bank, 2013). For a better comparison, we choose data from different regional areas[11] and check whether there are differences in the main services.[12] Starting with an LC deal,[13] we obtain the data in table 4.

Service	Credit Suisse	Deutsche Bank	BNP Paribas
Advising	10 $bp, min.$ 200, $max.$ 1,500	100 $bp, min.$ 166, $max.$ 441	81
Amendment	200	110	54
Acceptance doc.	–	150 $bp, min.$ 166	25 $bp, min.$ 108
Pre-advice	100	83	–
Processing doc.	–	150 $bp, min.$ 166	25 $bp, min.$ 108
Confirmation com.	$min.$ 300	–	–
Deferred payment	200	–	40
Acceptance com.	–	–	40
Discrepant doc.	100	110	101

Table 4: Fees overview for export LCs (in CHF)

As we can see in table 4, the total transaction fees for our deal in case of Credit Suisse are 800 CHF. Generally, any change or mistake in the documents causes service fees.
This equals 0.8 % of the total trade amount for Credit Suisse, but we should not forget this covers only the handling of documents.

We apply the same procedure for DC transactions; it results in table 5.

[10] all amounts in CHF, exchange rates based on query from 2019.10.30 (finanzen.ch, 2019*b*,*c*)
[11] Switzerland, Germany and United Arab Emirates
[12] more fees can be applied for additional services
[13] different perspective (importer or exporter) leads to different costs

Service	Credit Suisse	Deutsche Bank	BNP Paribas
DC commission	20 *bp*, *min.* 200	150 *bp*, *min.* 138	12.5 *bp per month*, *min.* 108
Amendment	–	110	12.5 *bp per month*, *min.* 54
Release of doc. without payment	200	150 *bp*, *min.* 138	–
Release of goods	100	166	–
Deferred payment/ acceptance	200	110	20 *bp*, *min.* 81
Cancellation	400	–	135

Table 5: Fees overview for DCs (in CHF)

We can see that the overall fee structure is lower for DCs than for LCs, but banks still require service fees for collecting documents on a party's behalf. The total fees in this case are 500 CHF for Credit Suisse.

As briefly described in section 2.1, in Ethereum the only transaction costs are for the execution of different operations, such as transferring Ether to a wallet address or making state changes, which nodes validate within the network. Further, the deployment of a smart contract also has costs, whereas calling a function does not, as long as an EOA calls a specific function. Generally, we can calculate the costs by multiplying the gas used by the actual gas price (Wood et al., 2014). Figure 5 below shows the development of Ethereum's gas price[14] from January to end of October 2019 (Etherscan, 2019):

[14]as of 10.30.2019

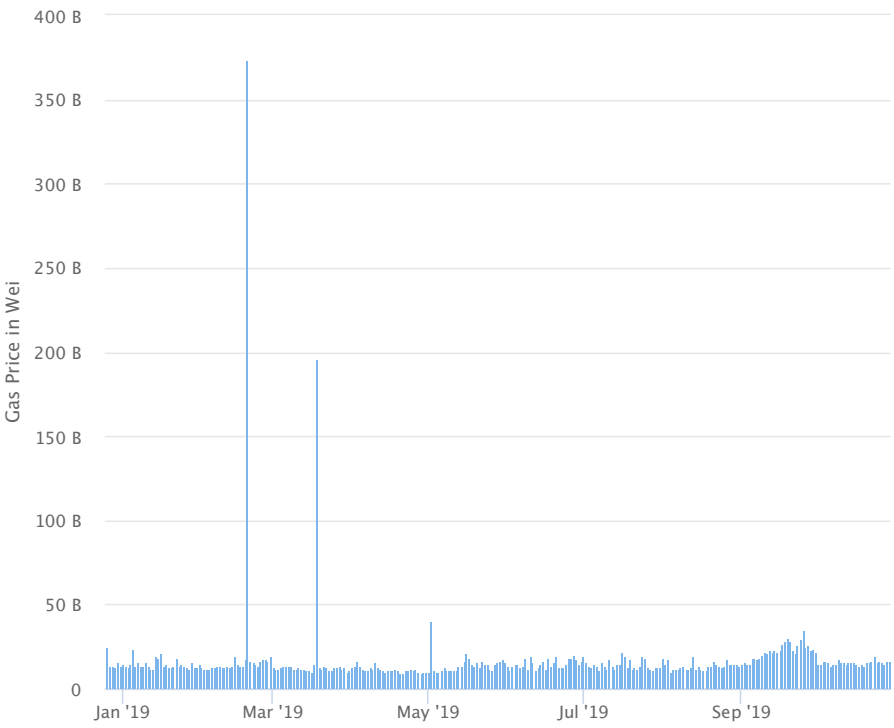

Figure 5: Gas price development for 2019 (in Wei)

We conclude that hype leads to an abnormally high gas price, which can slow down the validation of a transaction, which in the worst case fails, or it makes it more expensive to get into a block (BlockChannel, 2019).
Based on the Ether price[15] and our execution with Ganache, we estimate the following costs for our smart contracts,[16] if we proceed with an average transaction speed, which costs 1.1 Gwei.

[15]Exchange rate at 11.10.2019: 189.35 CHF (CoinGecko, 2019)
[16]exchange rate at 11.10.2019, based on the queries from finanzen.ch (2019*a*)

Process steps	Gas used	Value in ETH	Value in CHF
Deployment SafeMath	76,654	0.0000843	0.0157
Deployment Whitelist	350,033	0.000385	0.0717
Deployment TradeFinance	1,937,526	0.0021313	0.3970
setMinimumRequirement	62,530	0.0000688	0.0128
addWhitelist	44,533	0.000049	0.0091
removeWhitelist	14,236	0.0000157	0.0030
access	109,545	0.0001205	0.0224
addOrder	433,251	0.0004766	0.0888
confirmOrder	52,640	0.0000579	0.0108
cancelOrder	47,028	0.0000517	0.0098
addGuarantee	183,440	0.0002018	0.0376
receiveOrderFreight	75,038	0.0000825	0.0154
receiveOrderCustoms	87,495	0.0000962	0.0182
receiveOrder	60,053	0.0000661	0.0125
reset	33,985	0.0000374	0.0071
Sum	3,472,771	0.0039248	0.7319

Table 6: Transaction costs for our smart contracts

From this table 6, we conclude, that the transaction fees – here for all parties combined – to execute our smart contracts total 0.73 CHF, which corresponds to 0.001 % of the transaction volume.[17] Compared to an LC or DC transaction, the fees reduced by 99 %. We must also consider that the fees to deploy the smart contracts are one-time costs, since we can use contracts multiple times. Excluding these costs would reduce the fees dramatically.

Dietrich (2018) estimates the costs from using the blockchain technology are equal to 10 % of today's costs for an LC, which are 80 CHF, a much higher cost than our estimation. Considering the development costs for the smart contracts as well, our estimation would lead to higher costs.

[17] if rounded to 1 CHF

4.7 Existing Blockchain Trade Finance Projects

This section reviews already existing projects in the area of trade finance that use blockchain technology.
The first example is we.trade, launched by a consortium of various banks. Built on Hyperledger Fabric, this platform should enhance efficiency and lower the process costs of international trade through the use of smart contracts triggered by specific events (we.trade, 2019). All parties who want to participate need to pass a KYC check successfully and must be customers of one bank working with we.trade (we.trade, 2019).
The second example is Marco Polo, developed together by TradeIX and R3, which uses the Corda Blockchain technology (MarcoPolo, 2019). Similarly to we.trade, they also use a central platform to provide their service to the businesses, with an additional ERP integration (Ledger Insights, 2019). The first transaction through this platform has been taken place at the beginning of 2019 and the participating banks aim to offer faster, more efficient and transparent trade finance solutions (Schiller, 2019).
Compared to our solution, customers do not have to be members in order to use these platforms, whereas the *TradeFinance* smart contract can be used and modified before deployment by everyone, thus allowing the addition of specific conditions.

4.8 Limitations

In this section we explore the limitations. We begin with the limited storage for variables, which makes complex processes with numerous variables impossible to run in a single smart contract. If we had added more variables in *structs* to the *TradeFinance* contract, we would have faced this problem. One possible solution is to make use of external calls or inheritance and outsource functions, whereas for inheritance one smart contract gets deployed (Schütz and Fertig, 2019, pp. 314f.).
Second, as everyone can participate in the Ethereum blockchain and develop their own smart contracts, auditing them for security issues is important, especially in a business-related environment. Bugs and insecure coding patterns make these contracts vulnerable, as was seen, for example, in the

DAO case, and today's techniques for finding security issues are limited or are at a very high price (Liu et al., 2018, pp. 814f.). To find already known issues, tests are done before deployment, but to detect functional errors, high manual effort is required which makes it expensive and time-consuming (Permenev et al., 2019).
Third, as smart contracts are irreversible and immutable (which is, from one perspective, favourable since it creates trust among unknown parties), if a smart contract executes as predicted it leaves no option to fix a deployed contract in the case of an erroneous execution (Luu et al., 2016).
Fourth, since Ethereum is a public blockchain, it has limited scalability to do business with huge data streams (Blockgeeks, 2019).
Finally, Ether has high volatility, so the price during a transaction can vary significantly (Martinez, 2019). This fact makes it difficult for firms to manage their working capital; hedging against the currency risk would be a solution, but this implies extra costs. In our *TradeFinance* smart contract, the seller's cash management planning is straightforward, as the time gap between setting up an order and the financier's payment to the smart contract is brief, but the buyer's repayment takes place in the future so that she or he carries a currency risk. In the favourable case, the exchange rate for her or his currency decreases and she or he has to pay less to the financier; in the worst case, she or he must pay more.

5 Discussion

To conclude, we have shown the various payment methods which are relevant for trading between two parties. Particularly if the parties are not in a business relationship, it is difficult for them to trust each other and to agree to an OA payment, since both have an incentive to either not pay or not deliver the goods. Banks as intermediaries solve this problem in form of LCs, as they carry risks on behalf of their customers. Likewise for DCs, banks are involved in the trade, but they do not bear the same risks as with LCs.
Based on a game theoretical analysis, we can confirm these findings, since we have seen that CIA and OA are not incentive compatible in a one-shot setting. One player has the opportunity to deviate to a better option; as

the opposite party can anticipate this behaviour, she or he does not play the optimal choice and trade does not occur. However, in a repeated game set-up, punishment is possible in future periods so that the aforementioned problem is resolved.

Based on these findings, we have further developed a process scheme which can be executed on the Ethereum blockchain and prevents the issues mentioned above, even in a one-shot setting. Generally, public blockchains are anonymous, so that we suppose the smart contract owner – that is, the seller – to know all involved parties' roles in a specific transaction. This is a result of bilateral bargaining beforehand. Moreover, the financing can be taken over by anyone who has fulfilled the minimum requirements and has been whitelisted. For this, we assume that a KYC check will be done.
As information about the products is not available on the blockchain, it is difficult to measure their condition. Additionally, the use of Oracles leads to dependencies, because the rules of a smart contract rely on the inputs. This would open an opportunity to manipulate the data in one's self interest.
With our solution, we showed how to diminish concerns about privacy in connection to the Ethereum blockchain. Compared to other blockchains, however, Ethereum's drawbacks of scalability, Ether and gas system volatility still remain.
From a legal perspective under Swiss laws, the formal use of smart contracts is allowed, but it is crucial to define a person's legal capacity. Additionally, we must differentiate between persons who are familiar with programming smart contracts and those who are not able to read source code.

Finally, we have demonstrated what the source code written in Solidity can look like for a trade transaction deployed on the Ethereum blockchain. Because our smart contract will be used in a business context, we have used the implementation of hash values instead of plain-text values to protect privacy. Further, to avoid brute force attacks effectively, we have introduced the method of salting, which makes these attacks less feasible.
Furthermore, the *TradeFinance* and *Whitelist* smart contracts reduce transaction costs enormously; assuming an LC transaction, costs with our smart contracts are reduced from approximately 0.8 % to 0.001 %, which corre-

sponds to a reduction by 99 %.

Overall, our *TradeFinance* smart contract together with the second *Whitelist* contract reduces the complexity of a typical trade finance transaction via instruments such as an LC or DC. Additional advantages are a high level of security for data integrity and a significant cost reduction. We have also demonstrated that for this particular trading use case, the Ethereum blockchain is utilisable.

Further research must be done on how to reverse a transaction through a smart contract for cases in which products arrive at a buyer's destination damaged. The question of how to reach a decentralised consensus – for example, that goods have arrived at the customs broker or freight company – also needs further analysis. On a final note, the development in terms of legal implications in different countries will be interesting.

References

Asmundson, I., Dorsey, T. W., Khachatryan, A., Niculcea, I. and Saito, M. (2011), 'Trade and trade finance in the 2008-09 financial crisis', *IMF Working Papers* pp. 1–65.

Atzei, N., Bartoletti, M. and Cimoli, T. (2017), A survey of attacks on ethereum smart contracts (sok), *in* 'International Conference on Principles of Security and Trust', Springer, pp. 164–186.

Berentsen, A. and Schär, F. (2017), 'Bitcoin, blockchain und kryptoassets: Eine umfassende einführung', *Aufl. Norderstedt: BoD–Books on Demand* .

Bergami, R. (2009), 'Ucp 600 rules–changing letter of credit business for international traders?', *International Journal of Economics and Business Research* **1**(2), 191–203.

BlockChannel (2019), 'The Case for Ethereum Scalability', `https://medium.com/blockchannel/the-case-for-ethereum-scalability-a66ed08d0bed`. Accessed: 10.31.2019.

Blockgeeks (2019), 'Hyperledger vs Ethereum Training: Which one is better?', `https://blockgeeks.com/guides/hyperledger-vs-ethereum/`. Accessed: 10.03.2019.

BNP Paribas (2019), 'Fees and Charges 2019', `https://cdn-pays.bnpparibas.com/wp-content/blogs.dir/171/files/2019/01/A5-UAE-Tariff-2019-Low.pdf`. Accessed: 10.30.2019.

Brenton, P. and Imagawa, H. (2005), 'Rules of origin, trade, and customs', *CUSTOMS* p. 183.

Büter, C. (2007), 'Außenhandel', *Grundlagen globaler und innergemeinschaftlicher Handelsbeziehungen. Heidelberg* .

Buterin, V. et al. (2014), 'A next-generation smart contract and decentralized application platform', *white paper* **3**, 37.

Cocco and Singh (2018), 'Top 6 technical advantages of Hyperledger Fabric for blockchain networks', `https://developer.ibm.com/tutorials/`

cl-top-technical-advantages-of-hyperledger-fabric-for-blockchain-networks/. Accessed: 10.01.2019.

CoinGecko (2019), 'Ethereum (ETH)', https://www.coingecko.com/de/munze/ethereum. Accessed: 11.10.2019.

Cong, L. W. and He, Z. (2019), 'Blockchain disruption and smart contracts', *The Review of Financial Studies* **32**(5), 1754–1797.

Credit Suisse (2013), 'Standard conditions for letter of credits', https://www.credit-suisse.com/media/assets/private-banking/docs/ch/unternehmen/unternehmen-unternehmer/publikationen/akkreditiv-konditionen-en.pdf. Accessed: 10.30.2019.

Credit Suisse (2016), 'Documentary Credits – Documentary Collections', https://www.credit-suisse.com/media/assets/private-banking/docs/ch/unternehmen/unternehmen-unternehmer/publikationen/akkreditivbroschre-en.pdf. Accessed: 10.19.2019.

Croner-i (2019), 'Letters of Credit: In-depth', https://app.croneri.co.uk/topics/letters-credit/indepth. Accessed: 10.20.2019.

Dab et. al. (2016), 'Digital Revolution in Trade Finance', https://www.bcg.com/publications/2016/digital-revolution-trade-finance.aspx. Accessed: 10.19.2019.

Deutsche Bank (2013), 'Trade Finance List of Prices and Services for Corporate Clients in Germany', https://www.deutsche-bank.de/fk/de/docs/Preis-Leistungsverzeichnis_A4_E_121220_SCREEN.pdf. Accessed: 10.30.2019.

Dietrich (2018), 'Die UBS und die Blockchain: Warum we.trade funktionieren kann', https://blog.hslu.ch/retailbanking/2018/10/02/die-ubs-und-die-blockchain-warum-we-trade-funktionieren-kann/. Accessed: 10.03.2019.

Eker (2019), 'Presentation – Types of Letters of Credit', https://www.letterofcredit.biz/index.php/2019/01/28/presentation-types-of-letters-of-credit/5/. Accessed: 09.01.2019.

Ellinger, E. P. and Neo, D. S. S. (2010), *The law and practice of documentary letters of credit*, Hart.

Ethereum Community (2019*a*), 'Ethereum Development Tutorial', `https://github.com/ethereum/wiki/wiki/Ethereum-Development-Tutorial`. Accessed: 09.25.2019.

Ethereum Community (2019*b*), 'Solidity Documentation', `https://solidity.readthedocs.io/en/v0.5.11/`. Accessed: 09.15.2019.

Etherscan (2019), 'Ethereum Gas Price History', `https://etherscan.io/chart/gasprice`. Accessed: 10.30.2019.

Federal Customs Administration FCA (2019), 'Fact sheet For determining the formal validity of proofs of origin', `http://tiny.cc/ezv_admin_ch`. Accessed: 10.05.2019.

Fenyk (2015), 'SWIFT MT798: Global integrated solution for Trade Finance', `https://cib.db.com/insights-and-initiatives/flow/33381.htm`. Accessed: 10.02.2019.

finanzen.ch (2019*a*), 'Währungsrechner: Ether - Schweizer Franken', `https://www.finanzen.ch/waehrungsrechner/ethereum-schweizer-franken`. Accessed: 11.10.2019.

finanzen.ch (2019*b*), 'Währungsrechner: Euro - Schweizer Franken', `https://www.finanzen.ch/waehrungsrechner/euro-schweizer-franken`. Accessed: 10.30.2019.

finanzen.ch (2019*c*), 'Währungsrechner: VAE Dirham - Schweizer Franken', `https://www.finanzen.ch/waehrungsrechner/dirham-schweizer-franken`. Accessed: 10.30.2019.

Fingerand, K. M. and Schuknecht, L. (1999), *Trade, finance and financial crises*, number 3, WTO Special Studies.

Ganesh and Olson (2018), 'Rebooting a Digital Solution to Trade Finance', `https://www.bain.com/insights/rebooting-a-digital-solution-to-trade-finance/`. Accessed: 10.02.2019.

Ganne (2019), 'Can Blockchain revolutionize international trade?', https://www.ibm.com/downloads/cas/KJDPQKBE. Accessed: 10.31.2019.

Giovannucci, D. (2007), 'Basic trade finance tools: Payment methods in international trade', *Available at SSRN 996765* .

Goswami, R. (2012), 'Collection arrangement: An analysis', *Available at SSRN 2132284* .

Grath, A. (2011), *The handbook of international trade and finance: the complete guide to risk management, international payments and currency management, bonds and guarantees, credit insurance and trade finance*, Kogan Page Publishers.

Gyr, E. (2019), Blockchain und Smart Contracts: die vertragsrechtlichen Implikationen einer neuen Technologie, PhD thesis, Rechtswissenschaftliche Fakultät der Universität Bern.

Hinkelman, E. G. (2008), *DICTIONARY OF INTERNATIONAL TRADE 8th Edition*, Librix. eu.

Hornby (2016), 'Salted Password Hashing - Doing it Right', https://www.codeproject.com/Articles/704865/Salted-Password-Hashing-Doing-it-Right. Accessed: 11.04.2019.

ICC Commercial Crime Services (2019), 'Trade Finance Documents Authentication', https://icc-ccs.org/icc_2527//icc_2527/index.php/site_content/56-services/289-trade-finance-documents-authentication. Accessed: 10.03.2019.

International Chamber of Commerce (1996), 'URC 522', https://www.law.kuleuven.be/personal/mstorme/URC522.pdf. Accessed: 10.17.2019.

International Chamber of Commerce (2007), 'UCP 600', http://static.elmercurio.cl/Documentos/Campo/2011/09/06/2011090611422.pdf. Accessed: 08.25.2019.

International Chamber of Commerce (2011), 'Incoterms rules 2010', https://iccwbo.org/resources-for-business/incoterms-rules/incoterms-rules-2010/. Accessed: 08.25.2019.

International Chamber of Commerce (2013), 'Uniform Rules for Bank Payment Obligations', https://icckauppa.fi/wp-content/uploads/sites/26/2016/06/750-icc-uniform-rules-for-bank-payment-obligations.pdf. Accessed: 10.20.2019.

International Chamber of Commerce (2019*a*), 'Access to trade finance', https://iccwbo.org/global-issues-trends/banking-finance/access-trade-finance/. Accessed: 09.29.2019.

International Chamber of Commerce (2019*b*), 'eUCP Version 2.0', https://cdn.iccwbo.org/content/uploads/sites/3/2019/06/icc-uniform-customs-practice-credits-v2-0.pdf. Accessed: 10.17.2019.

International Trade Center (2009), 'How to Access Trade Finance', http://www.intracen.org/WorkArea/DownloadAsset.aspx?id=28163. Accessed: 09.22.2019.

Jaffeux, C. and Wieser, P. (2012), *Essentials of Logistics and Management*, CRC Press.

Jones, S. A. (2018), *Trade and Receivables Finance*, Springer.

Kenneth (2018), 'Ethereum account', https://medium.com/coinmonks/ethereum-account-212feb9c4154. Accessed: 09.25.2019.

Kenton and Murphy (2019), 'Documentary Collection', https://www.investopedia.com/terms/d/documentary-collection.asp. Accessed: 10.17.2019.

Klapper, L. (2005), *The role of factoring for financing small and medium enterprises*, The World Bank.

Ledger Insights (2019), 'Bank of America joins Marco Polo blockchain trade finance network', https://www.ledgerinsights.com/bank-of-america-joins-marco-polo-blockchain-trade-finance-network/. Accessed: 10.28.2019.

Lee (2018), 'Smart Containers — A New Container Technology Set To Disrupt The Logistics Marketplace', https://medium.com/@cryptolee/smart-containers-a-new-container-technology-set-to-disrupt-

the-logistics-marketplace-part-1-c2e286f0820a. Accessed: 10.02.2019.

Lee, W.-M. (2019), 'Beginning ethereum smart contracts programming'.

Lee, Y. W. and Stowe, J. D. (1993), 'Product risk, asymmetric information, and trade credit', *Journal of Financial and Quantitative analysis* **28**(2), 285–300.

Lipton and Levi (2018), 'An Introduction to Smart Contracts and Their Potential and Inherent Limitations', https://corpgov.law.harvard.edu/2018/05/26/an-introduction-to-smart-contracts-and-their-potential-and-inherent-limitations/. Accessed: 10.10.2019.

Liu, H., Liu, C., Zhao, W., Jiang, Y. and Sun, J. (2018), S-gram: towards semantic-aware security auditing for ethereum smart contracts, *in* 'Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering', ACM, pp. 814–819.

Luu, L., Chu, D.-H., Olickel, H., Saxena, P. and Hobor, A. (2016), Making smart contracts smarter, *in* 'Proceedings of the 2016 ACM SIGSAC conference on computer and communications security', ACM, pp. 254–269.

MarcoPolo (2019), 'Platform', https://www.marcopolo.finance/platform/. Accessed: 10.30.2019.

Martinez (2019), 'Bitcoin, Ethereum and XRP Prepare for Volatility; Bullish or Bearish?', https://www.ccn.com/bitcoin-ethereum-xrp-volatility-bullish-or-bearish/. Accessed: 11.06.2019.

Mathys (2019), 'FINMA guidance: stringent approach to combating money laundering on the blockchain', https://www.finma.ch/en/news/2019/08/20190826-mm-kryptogwg/. Accessed: 10.03.2019.

Miller (2019), 'Gas Economics', https://github.com/LeastAuthority/ethereum-analyses/blob/master/GasEcon.md. Accessed: 09.25.2019.

Niepmann and Schmidt-Eisenlohr (2016), 'Trade Finance around the world', https://voxeu.org/article/trade-finance-around-world. Accessed: 08.27.2019.

Nordea (2018), 'Trade finance is going open account', https://insights.nordea.com/en/ideas/trade-finance-is-going-open-account/. Accessed: 09.28.2019.

Onderstal, S. et al. (2014), *Economics of organizations and markets*, AmsterdamPearson9789043030410.

OpenZeppelin (2019), 'SafeMath', https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/math/SafeMath.sol. Accessed: 10.31.2019.

Permenev, A., Dimitrov, D., Tsankov, P., Drachsler-Cohen, D. and Vechev, M. (2019), 'Verx: Safety verification of smart contracts', *Security and Privacy* **2020**.

Raskin, M. (2016), 'The law and legality of smart contracts'.

Raunek (2019), 'Bill Of Lading in Shipping: Importance, Purpose, And Types', https://www.marineinsight.com/maritime-law/what-is-bill-of-lading-in-shipping/. Accessed: 10.31.2019.

Sallaba, Gentner, E. (2017), 'Grenzenlos vernetzt: Digital Twins', https://www2.deloitte.com/de/de/pages/technology-media-and-telecommunications/articles/digital-twins.html. Accessed: 10.03.2019.

Savelyev, A. (2017), 'Contract law 2.0:'smart'contracts as the beginning of the end of classic contract law', *Information & Communications Technology Law* **26**(2), 116–134.

Schiller (2019), 'Marco Polo Blockchain (R3 Corda) gewinnt immer mehr Kunden', https://blockchainwelt.de/marco-polo-trade-finance-network-blockchain/. Accessed: 10.04.2019.

Schmidt-Eisenlohr, T. (2013), 'Towards a theory of trade finance', *Journal of International Economics* **91**(1), 96–112.

Schweizerischer Bundesrat (2018), 'Verordnung über die Zollansätze für Waren im Verkehr mit EU- und EFTA-Mitgliedstaaten', https://www.admin.ch/opc/de/classified-compilation/20081202/index.html. Accessed: 10.04.2019.

Schütz, A. and Fertig, T. (2019), *Blockchain für Entwickler: Das Handbuch für Software Engineers. Grundlagen, Programmierung, Anwendung. Mit vielen Praxisbeispielen*, Rheinwerk Computing.

Sharma (2019), 'Best programming languages to build smart contracts', https://www.blockchain-council.org/blockchain/best-programming-languages-to-build-smart-contracts/. Accessed: 11.01.2019.

Silitschanu (2019), 'International Trade Finance - Factoring vs. Forfaiting', https://www.americanexpress.com/us/foreign-exchange/articles/financing-international-trade-factoring-vs-forfaiting/. Accessed: 10.31.2019.

Smith (2018), 'The story of the DAO, and how it shaped Ethereum', https://www.coininsider.com/what-happened-to-the-dao/. Accessed: 11.01.2019.

SWIFT (2016), 'Bank Payment Obligation A new payment method', https://www.swift.com/node/35051. Accessed: 10.20.2019.

Swissbilling (2019), 'Reverse-Factoring', https://www.swissbilling.ch/wissen/factoring/reverse-factoring/. Accessed: 10.17.2019.

Szabo, N. (1997), 'Formalizing and securing relationships on public networks', *First Monday* **2**(9).

Tabora (2018), 'Databases and Blockchains, The Difference Is In Their Purpose And Design', https://hackernoon.com/databases-and-blockchains-the-difference-is-in-their-purpose-and-design-56ba6335778b. Accessed: 10.03.2019.

Tian, F. (2016), An agri-food supply chain traceability system for china based on rfid & blockchain technology, *in* '2016 13th international conference on service systems and service management (ICSSSM)', IEEE, pp. 1–6.

Trade Finance Global (2019*a*), 'Documentary Collections (DCs)', https://www.tradefinanceglobal.com/services/documentary-collections/. Accessed: 10.17.2019.

Trade Finance Global (2019*b*), 'Letters of Credit | The TFG Ultimate Guide', https://www.tradefinanceglobal.com/letters-of-credit/. Accessed: 08.15.2019.

Tricks and Parson (2018), 'The Legal Status of Electronic Bills of Lading', https://iccwbo.org/content/uploads/sites/3/2018/10/the-legal-status-of-e-bills-of-lading-oct2018.pdf. Accessed: 10.17.2019.

UBS (2015), 'Documentary Credits and Collections', http://tiny.cc/ubs-international. Accessed: 10.03.2019.

Vujičić, D., Jagodić, D. and Ranđić, S. (2018), Blockchain technology, bitcoin, and ethereum: A brief overview, *in* '2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)', IEEE, pp. 1–6.

Wang (2019), 'Why logistics scenarios matter for the future of the industrial IoT', https://www.weforum.org/agenda/2019/07/industrial-iot-iiot-logistics-supply-chain/. Accessed: 10.05.2019.

Wax (2018), 'Bill of Lading (Konnossement) und Telex Release: Vor- und Nachteile in der Übersicht', https://freighthub.com/de/blog/vor-nachteile-original-bill-lading-telex-release/. Accessed: 10.03.2019.

we.trade (2019), 'The platform', https://we-trade.com/the-platform. Accessed: 10.31.2019.

Wood, G. et al. (2014), 'Ethereum: A secure decentralised generalised transaction ledger', *Ethereum project yellow paper* **151**(2014), 1–32.

Appendix

Solidity Code for SafeMath library[18]

```
pragma solidity ^0.5.11;

library SafeMath {

    function add(uint256 a, uint256 b) internal pure returns
    ↪  (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }

    function sub(uint256 a, uint256 b) internal pure returns
    ↪  (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    function sub(uint256 a, uint256 b, string memory
    ↪  errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;
        return c;
    }

    function mul(uint256 a, uint256 b) internal pure returns
    ↪  (uint256) {
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
```

[18] used from OpenZeppelin (2019)

```
        require(c / a == b, "SafeMath: multiplication
        ↪  overflow");
        return c;
    }

    function div(uint256 a, uint256 b) internal pure returns
    ↪  (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    function div(uint256 a, uint256 b, string memory
    ↪  errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        uint256 c = a / b;
        return c;
    }
}
```

Solidity Code for Whitelist Smart Contract

```
pragma solidity ^0.5.11;

import "https://github.com/SmartOtter/TradeFinance/blob/maste
↪  r/SafeMath.sol";

contract Whitelist {

    using SafeMath for uint256;

    address payable public buyer = msg.sender;
    uint256 internal minimumAmount;
    uint256 internal minimumScore;
    uint256 x = 1000000000000000000;
    address payable public financier;

    constructor() public {
```

```
        msg.sender == buyer;
    }

    mapping(address => bool) public whitelisted;

    modifier onlyBuyer {
        require(msg.sender == buyer);
        _;
    }

    function setMinimumRequirement(uint256 _minimumAmount,
    ↪  uint256 _minimumScore) public onlyBuyer {
        minimumAmount = _minimumAmount.mul(x);
        minimumScore = _minimumScore;
    }

    function addWhitelist(address _financier, uint256 _score)
    ↪  public onlyBuyer {
        require(_financier.balance >= minimumAmount);
        require(_score >= minimumScore);
        whitelisted[_financier] = true;
    }

    function removeWhitelist(address _financier) public
    ↪  onlyBuyer {
        whitelisted[_financier] = false;
    }

    function validateFinancier(address _financier) public
    ↪  view returns(bool){
        return(whitelisted[_financier]);
    }
}
```

Solidity Code for TradeFinance Smart Contract

```
pragma solidity ^0.5.11;

import "https://github.com/Smart0tter/TradeFinance/blob/maste⌋
↪ r/SafeMath.sol";
import "https://github.com/Smart0tter/TradeFinance/blob/maste⌋
↪ r/Whitelist.sol";

contract TradeFinance {

    Whitelist w;

    using SafeMath for uint256;

    bytes32 internal priceSeller;
    bytes32 internal quantitySeller;
    bytes32 internal priceBuyer;
    bytes32 internal quantityBuyer;
    uint256 internal orderAmountSeller;
    uint256 internal orderAmountBuyer;
    uint256 y = 1000000000000000000;
    string internal productname;
    uint256 internal weight;
    uint256 internal quantity;
    string internal shippingmode;
    string internal origin;
    string internal destination;
    uint256 internal freightrate;
    uint256 internal customsDuty;
    uint256 internal balanceSeller1;
    uint256 internal balanceSeller2;
    uint256 internal balanceSeller3;
    uint256 internal balanceCustoms1;
    uint256 internal balanceCustoms2;
    bytes32 public orderAddress;
```

```
bytes32 public guaranteeAddress;
bytes32 internal billAddress;
uint256 internal ordersCount;
uint256 internal guaranteesCount;
address payable public seller = msg.sender;
address payable public buyer;
address payable public freight;
address payable public customs;
address payable public financier;
uint256 internal salt;

event OrderCancelled(string description);

event OrderConfirmed(string description);

event OrderLocked(string description);

event OrderReceivedFreight(string description);

event OrderReceivedCustoms(string description);

event OrderReceived(string description);

event GuaranteeActive(string description);

event GuaranteeInactive(string description);

enum OrderState { Negotiation, Created, Locked, Freight,
↪  Customs, Received, Cancelled }

enum GuaranteeState { Inactive, Active }
GuaranteeState public guaranteestate;

mapping(bytes32 => Guarantee) public guarantees;

mapping(bytes32 => Order) public orders;
```

```
constructor() public {
    msg.sender == seller;
}

struct Guarantee {
    bytes32 guaranteeAddress;
    bytes32 orderAddress;
    address to;
    bool isGuarantee;
}

struct Order {
    bytes32 orderAddress;
    address payable seller;
    address payable buyer;
    bytes32 priceSeller;
    bytes32 quantitySeller;
    uint256 weight;
    string productname;
    uint256 freightrate;
    uint256 orderAmountSeller;
    OrderState orderstate;
    uint256 customsDuty;
    bytes32 guarantee;
    bool isOrder;
}

modifier onlySeller() {
    require(msg.sender == seller);
    _;
}

modifier onlyBuyer() {
    require(msg.sender == buyer);
    _;
```

```
}

modifier onlyFreight() {
    require(msg.sender == freight);
    _;
}

modifier onlyCustoms() {
    require(msg.sender == customs);
    _;
}

modifier inOrderState(OrderState _orderstate) {
    require(orders[orderAddress].orderstate ==
    ↪  _orderstate);
    _;
}

modifier inGuaranteeState(GuaranteeState _guaranteestate)
↪  {
    require(guaranteestate == _guaranteestate);
    _;
}

function access(address payable _buyer, address payable
↪  _freight, address payable _customs, bytes32
↪  _orderAddress, uint256 _salt) public onlySeller {
    require(orders[_orderAddress].orderstate ==
    ↪  OrderState.Negotiation);
    buyer = _buyer;
    freight = _freight;
    customs = _customs;
    orderAddress = _orderAddress;
    salt = _salt;
}
```

```
function addOrder(bytes32 _orderAddress, address payable
↪  _seller, address payable _buyer, bytes32
↪  _priceSeller, bytes32 _quantitySeller, uint256
↪  _weight,
    string memory _productname, uint256 _freightrate,
    ↪  uint256 _orderAmountSeller, uint256 _customsDuty)
    ↪  public onlySeller {
    require(orders[_orderAddress].orderstate ==
    ↪  OrderState.Negotiation, "Error: Order cannot
    ↪  modified anymore!");
    if(isOrder(_orderAddress)) revert();
    orders[_orderAddress].orderAddress = _orderAddress;
    orders[_orderAddress].isOrder = true;
    orders[_orderAddress].orderstate =
    ↪  OrderState.Negotiation;
    orders[_orderAddress].seller = _seller;
    orders[_orderAddress].buyer = _buyer;
    orders[_orderAddress].priceSeller = _priceSeller;
    orders[_orderAddress].quantitySeller =
    ↪  _quantitySeller;
    orders[_orderAddress].weight = _weight;
    orders[_orderAddress].productname = _productname;
    orders[_orderAddress].freightrate = _freightrate;
    orders[_orderAddress].orderAmountSeller =
    ↪  _orderAmountSeller;
    orders[_orderAddress].customsDuty = _customsDuty;
    orders[_orderAddress].guarantee = "";
    ordersCount++;
    orderAddress = _orderAddress;
    priceSeller = keccak256(abi.encode(_priceSeller,
    ↪  salt));
    quantitySeller =
    ↪  keccak256(abi.encode(_quantitySeller, salt));
    weight = _weight;
    productname = _productname;
    freightrate = _freightrate.mul(y);
```

```
    orderAmountSeller = _orderAmountSeller.mul(y);
    customsDuty = _customsDuty.div(100);
}

function isOrder(bytes32 _orderAddress) public view
↪   returns(bool) {
    return orders[_orderAddress].isOrder;
}

function cancelOrder(bytes32 _orderAddress) public
↪   inOrderState(OrderState.Negotiation) onlySeller
↪   returns(bool) {
    require(orders[_orderAddress].orderstate ==
    ↪   OrderState.Negotiation);
    emit OrderCancelled("Order has been cancelled by the
    ↪   seller");
    orders[_orderAddress].orderstate =
    ↪   OrderState.Cancelled;
    return true;
}

function confirmOrder(bytes32 _priceBuyer, bytes32
↪   _quantityBuyer, uint256 _orderAmountBuyer, bytes32
↪   _orderAddress) public
↪   inOrderState(OrderState.Negotiation) onlyBuyer
↪   returns(bool) {
    if (keccak256(abi.encode(_priceBuyer, salt)) ==
    ↪   priceSeller &&
    ↪   keccak256(abi.encode(_quantityBuyer, salt)) ==
    ↪   quantitySeller && _orderAmountBuyer.mul(y) ==
    ↪   orderAmountSeller) {
        require(orders[_orderAddress].orderstate !=
        ↪   OrderState.Cancelled);
        emit OrderConfirmed("Order has been confirmed by
        ↪   the buyer");
```

```
        orders[_orderAddress].orderstate =
        ↪  OrderState.Created;
        return true;
    } else {
        emit OrderCancelled("Order has not been confirmed
        ↪  by the buyer");
        orders[_orderAddress].orderstate =
        ↪  OrderState.Cancelled;
        return false;
    }
}

function addGuarantee(bytes32 _guaranteeAddress, bytes32
↪  _orderAddress, address _to) public payable {
    require(orderAddress == _orderAddress);
    require(orders[_orderAddress].orderstate ==
    ↪  OrderState.Created);
    require(orders[_orderAddress].isOrder = true);
    //require(w.whitelisted[msg.sender] = true);
    require(msg.value >= orderAmountSeller);
        if(isGuarantee(_guaranteeAddress)) revert();
        guarantees[_guaranteeAddress].guaranteeAddress =
        ↪  _guaranteeAddress;
        guarantees[_guaranteeAddress].isGuarantee = true;
        guarantees[_guaranteeAddress].orderAddress =
        ↪  _orderAddress;
        guarantees[_guaranteeAddress].to = _to;
        guaranteesCount++;
        orders[_orderAddress].guarantee =
        ↪  _guaranteeAddress;
        guaranteeAddress = _guaranteeAddress;
        emit GuaranteeActive("Guarantee is Active");
        guaranteestate = GuaranteeState.Active;
        financier = msg.sender;
        emit OrderLocked("Order payment is guaranteed by
        ↪  bank");
```

```
        orders[_orderAddress].orderstate =
        ↪  OrderState.Locked;
}

function isGuarantee(bytes32 _guaranteeAddress) public
↪  view returns(bool) {
    return guarantees[_guaranteeAddress].isGuarantee;
}

function contractBalance() public view returns(uint256) {
    return address(this).balance;
}

function receiveOrderFreight(bytes32 _billAddress, string
↪  memory _shippingmode, bytes32 _orderAddress, uint256
↪  _weight, string memory _origin, string memory
↪  _destination) public inOrderState(OrderState.Locked)
↪  onlyFreight returns(bool) {
    require(orders[_orderAddress].orderstate ==
    ↪  OrderState.Locked);
    emit OrderReceivedFreight("Order arrived at Freight
    ↪  Company");
    orders[_orderAddress].orderstate = OrderState.Freight;
    billAddress = _billAddress;
    shippingmode = _shippingmode;
    weight = _weight;
    origin = _origin;
    destination = _destination;
    return true;
}

function receiveOrderCustoms(bytes32 _orderAddress)
↪  public inOrderState(OrderState.Freight) onlyCustoms
↪  returns(bool) {
    require(orders[_orderAddress].orderstate ==
    ↪  OrderState.Freight);
```

```
    emit OrderReceivedCustoms("Order arrived at Customs
    ↪ broker");
    orders[_orderAddress].orderstate = OrderState.Customs;
    require(orders[_orderAddress].orderstate ==
    ↪ OrderState.Customs);
    require(address(this).balance >= orderAmountSeller);
    address(freight).transfer(freightrate);
    require(address(this).balance >=
    ↪ orderAmountSeller.sub(freightrate), "Error:
    ↪ Contract balance too low!");
    balanceSeller1 = orderAmountSeller.sub(freightrate);
    balanceSeller2 = orderAmountSeller.mul(customsDuty);
    balanceSeller3 = balanceSeller1.sub(balanceSeller2);
    address(seller).transfer(balanceSeller3);
    emit GuaranteeInactive("Guarantee is Inactive");
    guaranteestate = GuaranteeState.Inactive;
    return true;
}

function receiveOrder(bytes32 _orderAddress) public
↪ inOrderState(OrderState.Customs) onlyBuyer
↪ returns(bool) {
    require(orders[_orderAddress].orderstate ==
    ↪ OrderState.Customs);
    emit OrderReceived("Order arrived at the buyer");
    orders[_orderAddress].orderstate =
    ↪ OrderState.Received;
    balanceCustoms1 = orderAmountSeller.sub(freightrate);
    balanceCustoms2 = balanceCustoms1.sub(balanceSeller3);
    require(address(this).balance >= balanceCustoms2);
    address(customs).transfer(balanceCustoms2);
    return true;
}

function reset(bytes32 _orderAddress, bytes32
↪ _guaranteeAddress) public onlySeller {
```

```
        require(address(this).balance == 0, "Error: Contract
        ↪  balance is not zero!");
        require(orders[_orderAddress].orderstate ==
        ↪  OrderState.Received ||
        ↪  orders[_orderAddress].orderstate ==
        ↪  OrderState.Cancelled);
        buyer = address(0);
        freight = address(0);
        customs = address(0);
        financier = address(0);
        guarantees[_guaranteeAddress].guaranteeAddress =
        ↪  bytes32(0);
        guarantees[_guaranteeAddress].orderAddress =
        ↪  bytes32(0);
        guarantees[_guaranteeAddress].to = address(0);
        guarantees[_guaranteeAddress].isGuarantee = false;
        billAddress = bytes32(0);
        salt = 0;
    }
}
```