

A Blockchain-Based System for Managing Payments in The Construction Supply Chain

Samer HAFFAR¹, Eren ÖZCEYLAN^{2*}

Keywords

Blockchain,
Construction industry,
Payments,
Supply chain,
Ethereum


Abstract – The global construction sector, employing 7% of the worldwide workforce and contributing 13% to global gross domestic product (GDP), is among the largest industries. It encompasses various stakeholders, including owners, contractors, subcontractors, and suppliers. However, the management of payments within this supply chain encounters numerous hurdles, such as delays, rework, errors, late payments, and inadequate supervision and financial oversight. This paper presents a blockchain-based payment handling system for construction supply chains. The system leverages blockchain's features of data transparency and sharing, and decentralization and immutability to provide a secure and trusted tool for handling payments. The system enables tracking work progress, the payment of installments for completed work, and provides a facility for resolving disputes between buyers and sellers on-chain. The system ensures smooth execution and commitments by all parties with blockchain data transparency, escrow payments and independent risk assessment. The paper provides a detailed description of the system design and results of function tests.


1. Introduction

The construction supply chain consists of the activities that lead to starting and completing a construction project of one or more buildings. These activities include identifying the demand for the buildings, construction activities (such as laying the foundation, concreting, welding, plastering, plumbing, etc.), maintaining the building during use, and the demolition of the building. The construction industry is a \$10 trillion industry, accounting for 13% of global GDP. A construction supply chain, thus, employs a lot of stakeholders that contribute to its activities, including the owners of the construction project, contractors, suppliers of raw materials as well as engineers and architects (Studer and De Brito Mello, 2021). Construction supply chains are characterized by the flow of materials, information, and finance across all stakeholders to ensure the smooth implementation of construction projects (Xue et al., 2007).

A smooth flow of funds, however, is a rare occurrence in the construction supply chain due to high number of stakeholders involved in construction projects as well as influences and disruptions in the flows of materials and information. Nanayakkara et al. (2021) reported several issues and problems that occur in the construction supply chain when handling payments among stakeholders which include: delays in completing work due to supply chain issues, going over budget, rework and errors, late payments, improper supervision and financial controls, improper withholding of payments, lack of trust between stakeholders. Ramachandra and Rotimi (2015) identified several factors causing delays in payments in the New Zealand construction industry which include: cash flow difficulties due to non-payment on other projects, disputes over payment claims, and dishonesty of the payers.

Blockchain is a distributed tamper-proof record of ordered transactions that is secured cryptographically. The blockchain ledger is stored on a network of computers, each computer having an identical copy of the entire ledger. The tamper-proof nature of the blockchain is ensured by the collective effort of all computers on the network (Levis et al., 2021). The Ethereum blockchain was released in 2015 and brought the concept of smart contracts to blockchains. Smart contracts allow writing and executing software applications on the blockchain (Zhou et al., 2022). Thus, all blockchains share several characteristics which include “immutability”, which

¹ Gaziantep University, Natural and Applied Sciences, 27410 Gaziantep, Türkiye. E-mail: sam.hafar@gmail.com  OrcID: 0000-0002-9317-0257

^{2*} **Corresponding Author.** Gaziantep University, Industrial Engineering Department, 27410 Gaziantep, Türkiye. E-mail: erenozceylan@gmail.com  OrcID: 0000-0002-5213-6335

Citation: Haffar, S., and Özceylan, E. (2024). A blockchain-based system for managing payments in the construction supply chain. *Natural Sciences and Engineering Bulletin*, 1(1), 28-47.

means it's impossible to make any changes to the data that is stored on a blockchain; "transparency", all data stored on the blockchain is publicly available and accessible.

In this paper, we propose a blockchain-based system that aims to manage and handle the processing of payments in the construction supply chain. The purpose of this system is to provide a solution that addresses the issues with construction supply chain payments reported in the literature. The system has the following features:

- **Tracking Work and Installments.** The system allows the supply chain stakeholders to track work on projects and manage the payment of installments through the supply chain.
- **Dispute Handling.** If any of the stakeholders fails to comply with the agreement, the system can handle initiating and resolving disputes.
- **Risk Mitigation.** The system mitigates the risks associated with payments by utilizing escrow accounts and credit risk scores for all stakeholders, thus creating a motivation for them to commit to their agreements and comply with their obligations.

To the knowledge of the authors, this is the first system in the literature that combines the features of work tracking, payment and installment handling, dispute handling and risk mitigation in one blockchain-based system, which is the main contribution of this study.

1.1. Literature Review

Researchers in the construction supply chain have explored the potential of blockchain technology to tackle various challenges. In their study, Shemov et al. (2020) investigated the application of blockchain as a digital platform to address common issues within the construction supply chain. Despite acknowledging security concerns associated with blockchain use, the authors concluded that the technology offers a viable solution to many challenges, particularly those related to trust and project delays. In another study, Lu, Wu, et al. (2021) developed a blockchain-based model for government supervision of construction work (GSCW) that incorporates an incentive mechanism. Their model facilitates information sharing, preserves privacy, and seamlessly integrates into existing GSCW teams' workflows without disruption. Lu, Li, et al. (2021) introduced a novel solution utilizing smart construction objects (SCOs) to address the challenge of bridging blockchain systems with real-life construction projects. The research demonstrates the effectiveness of this approach in ensuring data accuracy and recording reputation scores. Zhang et al. (2023) explored the potential of blockchain technology in addressing flaws within construction contract management (CCM), such as information sharing and payment processes.

Blockchain technology is not only used for the construction supply chain, but also in other supply chains/supply chain applications as well. The exploration of blockchain technology's application in supply chains is an extensively studied subject in academic literature, driven by its attributes including traceability, transparency, decentralization, immutability, and automation. According to Han and Fang (2024), blockchain technology is used in a variety of supply chain functions, including logistics traceability, supply chain finance, supply chain collaboration, sustainable management, and risk management. Ioannou and Demirel (2022) reported that blockchain can be harnessed in supply chain finance to tackle issues such as limited visibility within the supply chain, cumbersome manual paperwork processes, and the risk of fraud. Archa et al. (2018) presented a system based on blockchain to tackle the problem of drug counterfeiting within the pharmaceutical supply chain. This system monitors the quantities of drugs held by each party and traces the movement of drugs across all parties involved in the supply chain. Rogerson and Parry (2020) outlined several instances where blockchain is employed to combat counterfeiting and enhance trust and visibility within the food supply chain. Kumar et al. (2021) proposed a holistic framework based on permissioned Blockchain technology to address challenges in international trade practices. The framework aims to enhance supply chain and logistics operations by addressing issues such as traceability, data integrity, and decentralized decision-making.

Ensuring the uninterrupted flow of funds within a construction supply chain is crucial for averting issues and delays (Studer and De Brito Mello, 2021). However, due to the rarity of a seamless fund flow, numerous studies have focused on pinpointing the challenges and obstacles encountered by stakeholders in managing payments. Some of these studies aimed to compile a spectrum of issues, as demonstrated by the works of Ramachandra and Rotimi (2015) and Swai et al. (2020). Ramachandra and Rotimi (2015) delved into the root causes of payment

problems in the New Zealand construction sector, identifying factors such as cash flow constraints stemming from delays and non-payments on other projects, disputes over payment claims, prevalent payment culture, payer dishonesty, inadequate supervision, financial control, and cost overruns. On the other hand, Swai et al. (2020) explored the factors contributing to unfair payment practices within the UK construction industry. Their findings implied late payments to contractors, conditional payment practices, downward pressure on contractor rates, and retention payments as among the leading factors. Other studies have proposed targeted solutions to specific challenges. For instance, Xie et al. (2019) investigated the impact of payment cycles at two critical junctures in the construction supply chain: from owner to general contractor and from general contractor to subcontractor. Their research revealed that shortening the payment periods at these junctures could expedite fund flow and facilitate the provision of advance funds, thereby ensuring smoother progress on construction projects.

Researchers utilize blockchain technology to automate payments and overcome challenges within supply chains. Kaid and Eljazzar (2018) presented a blockchain-based system aimed at resolving trust and visibility concerns among stakeholders, incorporating a straightforward mechanism for managing payment installments within the supply chain parties. Their approach involves buyers and sellers agreeing on a rule, such as withholding payment until 50% of the contracted services are fulfilled, and subsequently sharing service-related information on the blockchain. In a blockchain-based scheme for supply chain finance proposed by Tsai (2023), three key actors are involved: a large enterprise acting as a buyer, a small and medium enterprise (SME) acting as a seller, and a financial institution. The process begins with the buyer creating a purchase order, followed by the seller shipping the products. Subsequent steps, such as invoicing and receipt confirmation, are then completed to finalize the purchase operation. Financial institutions then issue a loan to the seller, with the buyer responsible for repaying the seller's loan. Omar et al. (2021) proposed a blockchain-based solution to enhance the efficiency of Group Purchasing Organization (GPO) contracts within the healthcare supply chain (HCSC), addressing current inefficiencies in procurement processes. By integrating blockchain technology and decentralized storage, the solution aims to streamline communication among stakeholders, minimize procurement timelines, and ensure transparency, thus potentially reducing pricing discrepancies and inaccuracies. Alnıpak and Toraman (2024) explored the adoption of Blockchain technology for payment transactions in Turkey's maritime industry, aiming to measure stakeholders' intentions through Technology Acceptance Model (TAM). Findings reveal strong positive relationships between usage intention and perceived usefulness, as well as perceived ease of use.

Researchers explored the application of blockchain technology to address payment-related challenges in the construction supply chain. In their work, Motawa and Kaka (2009) introduced an IT system capable of modeling various payment systems, allowing stakeholders within the construction supply chain to collectively determine the most suitable payment mechanism. This approach aims to ensure smooth cash flow and safeguard the supply chain against potential disruptions, thereby satisfying all involved parties. Meanwhile, Hamledari and Fischer (2021) examined the potential of blockchain-based smart contract solutions to automate progress payment tracking in the construction supply chain. Additionally, Das et al. (2020) presented a decentralized blockchain-based model for managing interim payments in construction projects. Notably, this model eliminates the need for trust among stakeholders and can autonomously enforce the terms and conditions of interim payments. Moreover, it facilitates the confidential sharing of sensitive financial information among stakeholders. Sigalov et al. (2021) introduced the implementation of smart contracts in construction projects to address issues with complex contract structures, delayed payments, and lack of transparency. By integrating Building Information Modeling (BIM) with blockchain-based smart contracts, automated and transparent payment processing is achieved. Their solution facilitates automatic payments upon acceptance of construction work, enhancing efficiency and trust in the construction industry.

2. Materials and Methods

In this section, we provide a detailed description of the proposed payment system with its features and capabilities. We propose a payment process workflow that is adopted in the payment system, so we provide a description of that process. Lastly, we provide a description of the system components, and the blockchain smart contract that was built to implement the proposed system functions.

2.1. System Features

2.1.1. Actors

There are four actors (or users) that have access to the system, each user plays a role in carrying out the payment process, namely: the buyer, the seller, the auditor, and the credit scoring agency. The **buyer** is the supply chain stakeholder that's buying goods and services and wants to handle the payment of the price of these goods and services on the system. The **seller** is the individual or entity selling those goods and services to the buyer. The **auditor** is a 3rd party entity that has two tasks: verifying the work that was completed by the seller and confirming its compliance with the terms and conditions of the agreement between the buyer and the seller. The other task of the auditor is resolving disputes between the buyer and the seller. The **credit scoring agency** is an institution that evaluates the financial situation of the buyer and the seller and gives them a credit score.

2.1.2. Work and Installment Tracking

The proposed system relies on the concept of “progress payments”, which associates work with payments. This is the reason why there are two components of the system, one is for tracking work, and the other is for tracking installments. These components are independent of each other (i.e., an installment can be paid by the buyer without necessarily having the work associated with it completed). This way, if the buyer is satisfied with the work being done by the seller and they trust them enough, they can pay the installments without having to track the work associated with it on the system. **Work Tracking** consists of three stages: first, the seller announces that they completed a portion of the work as agreed, then the auditor verifies the completed work to make sure it complies with the terms and conditions of the agreement, then the buyer confirms work completion. All that progress is committed to the blockchain in the form of “events”. **Installment Tracking** works by defining the installments that need to be paid by the buyer; the buyer pays the installments one after the other to the system, and the seller then withdraws those installments from the system. Thus, the system acts as a “trusted middleman” between the buyer and the seller.

2.1.3. Credit Risk and Escrows

The system requires the buyer and the seller to declare their credit score. Credit risk assessment serves as a valuable tool for both buyers and sellers in evaluating each other's financial stability. This assessment informs decisions regarding various installment payment terms, including the escrow amount, down payment, and installment count. Financial information provided by both parties is evaluated by a credit assessment agency to determine credit risk. Subsequently, this information is shared with both the buyer and seller as part of the payment process. Credit assessment is based on several indicators of a business. For example, the information taken into account when evaluating the credit risk of SMEs may include short term debt/equity, cash/assets, EBITDA/assets, retained earnings/assets and EBITDA/interest expenses (Altman and Sabato, 2007).

Based on the credit risk information, the buyer, and the seller both agree on an “escrow” amount that each of them pays as a guarantee for the smooth implementation and compliance with the terms and conditions of the agreement. Escrow and payment retention are common practices within the construction industry aimed at safeguarding the interests of owners. In the study by Antipin and Trufanova (2021), escrow accounts serve to protect shareholders of a construction project from developers with uncertain financial standing. Funds from shareholders are deposited into the escrow account, which is then utilized by a bank to finance project operations. The developer only receives funds from the escrow account upon project completion and commissioning. Further, as discussed in Swai et al. (2020), payment retention ensures that contractors fulfill their contractual obligations, thereby safeguarding the interests of owners. The buyer and the seller deposit their escrow payments in the system, and then they can withdraw their escrows once the project is over.

2.1.4. Dispute Handling

If one party (buyer or seller) fails to comply with their obligations as defined in the terms and conditions of their contract, the other party can dispute that noncompliance from within the system. If a dispute is initiated, all further work on the system is frozen, including deposits and withdrawals, until the dispute is resolved. There are two possible resolutions for a dispute; either both parties agree to proceed with the agreement as described or they terminate the agreement. If the buyer and seller agree to terminate the agreement, a settlement is made

where the buyer and the seller are refunded a portion of whatever amount that is left in the system on their agreement's balance (which consists of the escrows that have been deposited as well as any installments that are still not withdrawn). The auditor is in charge of handling dispute resolution as well as settling balances.

2.1.5. Blockchain

The system is implemented on top of blockchain technology. The use of blockchain technology has several benefits for payments handling which include:

- **Immutability:** All information about a deal is tamper-proof and cannot be changed by anyone outside the functions that the system offers. This makes the data extremely reliable and trustworthy.
- **Transparency:** All information and data, including those of previous deals performed by the buyer and/or the seller, are publicly available and accessible. This allows both the buyer and the seller to learn about the reputation, credibility, and history of dealings (work completion, disputes, delays, etc.) of the other party and ensure that they're a reliable entity to work with.
- **Information Sharing:** Blockchain is a great platform for information sharing. Once a piece of information becomes available on the blockchain, it can be accessible by all parties in real-time all over the world.

2.2. Payment Process Workflow

The system's workflow consists of five stages, namely: Deal Creation, Depositing the Escrow, Work Tracking and Installments, Disputes, and Completion. Figure 1 is a flow chart of the proposed payment process stages. Below is a description of each stage:

- **Deal Creation:** In this stage, a contract (in the system, known as a "Deal") that defines the terms of the service is established between the buyer and the seller. The system is designed to have the seller create the contract and define its terms and buyer to approve. The buyer's approval of the deal is indicated by their depositing the escrow amount in the next stage. When creating the deal, the seller specifies the following: a link to the full contract document, the buyer, the auditor, the credit scoring agency, the number of installments and their amounts, the escrow amount of the buyer and the escrow amount of the seller.
- **Deal Guarantees:** In this stage, the buyer and the seller have to deposit an amount of money in the smart contract that serves as a guarantee against failure of compliance with the terms and conditions of the contract. This stage consists of two steps, credit score evaluation and escrow deposit.
 - o **Credit Score:** The independent scoring agency provides the credit score of both the buyer and the seller directly to the smart contract.
 - o **Escrow Deposit:** In this stage, both the buyer and the seller deposit their escrow amounts as defined in the deal. This stage concludes when both escrow amounts are deposited. The system prevents any further steps until escrows are deposited. However, the buyer can still deposit installments even if they have not deposited their escrow yet.
- **Work, Installments:** At this stage, the seller starts working on delivering the services as agreed in the contract that is linked to by the deal in the system. This stage has two activities that run concurrently, namely: tracking work and paying installments.
 - o **Work Tracking:** Whenever the seller completes a portion of the service, they can announce that to the buyer and the auditor through the system. The auditor then verifies the completed work, then announces that they approve it. Lastly, the buyer reviews the completed work and approves it.
 - o **Paying Installments:** If the buyer is satisfied with the service, they deposit the installment that corresponds to the portion of work that is completed. The buyer can choose to pay the installment at any time, even if work is still in progress or has not started yet. The buyer can, thus, pay all installments even before the project starts.
- **Disputes:** This stage is optional and can only be entered if the buyer or the seller decides so. If the buyer or the seller are not satisfied with the other party's compliance with the deal terms and conditions, they can initiate a dispute. When initiating a dispute, the system allows the user to mention the reasons they think the other party is noncompliant with the terms of the deal. When a dispute is initiated, the

system freezes all further activities until that dispute is resolved. The auditor then reviews the dispute and resolves it as per the terms of the contract. The result of the dispute is either “dispute resolution” or “deal termination”.

- **Dispute resolution:** If the dispute is resolved, work resumes from the point it halted due to the dispute and continues according with the previous stage.
- **Deal termination:** If the deal is terminated, the auditor then “settles the balances” by refunding the balance of the deal (which consists of the escrows and any unwithdrawn installments) back to the buyer and the seller as per the contract terms and conditions.
- **Completion:** Once all installments are deposited by the buyer, work on the deal officially concludes. At this stage, two main activities take place as follows:
 - **Installment Withdrawal:** The seller continues to withdraw whatever amounts from the deposited installments that are still not withdrawn. Then, once all amounts are withdrawn, the system marks the deal as completed.
 - **Escrow Withdrawal:** Once the deal is marked as completed, the system then allows the buyer and the seller to withdraw the escrows that are deposited at the beginning of the deal.

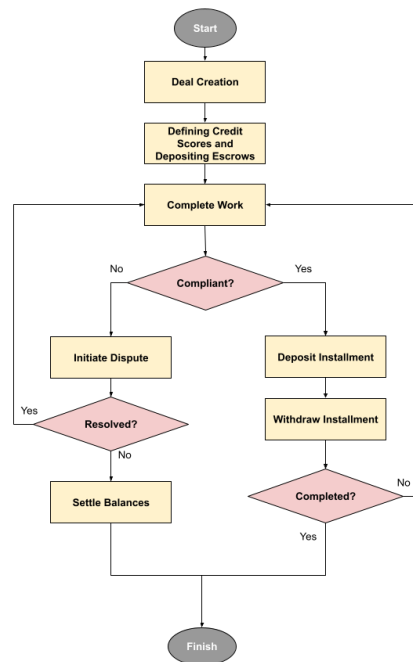


Figure 1. Flow of the payment process.

2.3. The Deals Smart Contract

The system is designed to manage the payments between two parties, one is a buyer buying goods and/or services, a seller selling these goods and services, and an auditor who’s in charge of ensuring the smooth delivery of goods and services and dispute resolution. The system also allows a credit scoring agency to provide the credit scores of the buyer and seller so that escrow amounts are determined in accordance with the risk involved in the deal. Thus, all stakeholders can use the system to handle the flow of goods and services and funds among them, two stakeholders at a time. The system consists of a single smart contract, called Deals. The smart contract offers several functions to the system users that allows them to carry out all actions relevant to the payment handling workflow explained above, which are: create deals, deposit, and withdraw escrows, monitor work, deposit and withdraw installments, initiate and resolve disputes and settle balances. The contract specifications document can be stored on an off-chain file storage service such as interplanetary file system (IPFS) or a cloud storage service. The smart contract was written in the Solidity programming language and can be deployed on the Ethereum blockchain or any blockchain that has an Ethereum virtual machine (EVM). Figure 2 illustrates the relationships between the various system components and its users.

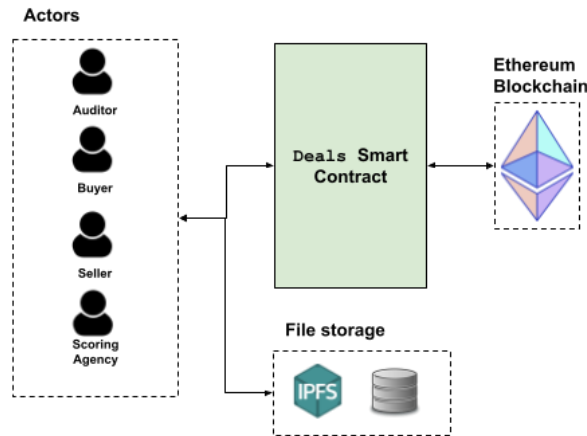


Figure 2. Components of the proposed payment system and their relationship with its actors/users.

2.3.1. Deal Structure

The information and data of every deal managed by the system is stored in a data structure called Deal. All Deal objects are stored in a *mapping* called *deals* and the unique identifiers of the stored deals are stored in an array called *dealIds*. Below are the data members of the Deal data structure and their descriptions:

- **Deal ID:** A unique identifier given to each deal. This identifier is used by the smart contract and the users to refer to the desired deal when executing functions.
- **Specs URI:** A link to the contract specifications document that defines the terms and conditions of the deal between the buyer and seller, which includes, among other terms: a description of the work that needs to be done, the total price, the number of installments and the amount of each installment, as well as the terms and conditions of dispute resolution.
- **Seller:** The seller's account (their address on the blockchain).
- **Buyer:** The buyer's account (their address on the blockchain).
- **Auditor:** The auditor's account (their address on the blockchain).
- **Credit Scoring Agency:** The credit scoring agency's account (their address on the blockchain).
- **Status:** The current status of the deal, which can be one of the statuses in the enum STATUS (Figure 3).
- **Comment:** A comment that is set by the user whenever they execute a function on the smart contract and cause the deal status to change. It is used to share messages among the users.
- **Installments:** An array of all the installments that need to be paid by the buyer to the seller; each element of the array constitutes an installment and specifies its amount in ETH.
- **Current Installment:** The index of the installment the latest installment that is yet to be paid from the Installments array.
- **Balance:** The total amount currently held in the deal by the smart contract, (balance = total deposited escrows + total deposited installments – total withdrawn installments and escrows).
- **Buyer Credit Score:** The credit score of the buyer, which is set by an independent credit scoring agency.
- **Seller Credit Score:** The credit score of the seller, which is set by an independent credit scoring agency.
- **Seller Escrow:** The total amount of escrow required to be deposited by the seller.
- **Buyer Escrow:** The total amount of escrow required to be deposited by the buyer.
- **Seller Escrow Deposited:** True, if the seller deposited their escrow, and false otherwise.
- **Buyer Escrow Deposited:** True, if the buyer deposited their escrow, and false otherwise.

```
enum STATUS {
    NEW,
    CREDIT_SCORES_ADDED, BUYER_ESCROW_DEPOSITED, SELLER_ESCROW_DEPOSITED,
    WORK_FINISHED, WORK_VERIFIED, WORK_CONFIRMED, INSTALLMENT_DEPOSITED,
    DISPUTE_INITIATED, DISPUTE_RESOLVED, DEAL_TERMINATED, BALANCES_SETTLED,
    ALL_INSTALLMENTS_DEPOSITED, DEAL_COMPLETED
}
```

Figure 3. The STATUS *enum* that defines all possible statuses of a deal object.

2.3.2. Events

The smart contract broadcasts a DealStatusChange event every time a function is executed on a deal that causes its status to change. The UpdateStatus function is used by all the other functions in the smart contract to emit the DealStatusChange event. The DealStatusChange event commits the following information to the blockchain:

- **Changed By:** The account address of the user that executed the function.
- **From:** The deal status before the function was executed.
- **To:** The deal status after the function execution.
- **Comment:** The new value of the comment after the status change.
- **Deal Balance:** The balance of the deal whose status was updated.
- **Contract Balance:** The balance of the smart contract, which is the sum of the balances of all deals managed by the smart contract.

2.3.3. Modifiers

The smart contract contains several modifiers that are designed to enforce user access permissions as well as the rules that apply at different stages of the payment process lifecycle as described in the workflow section of this paper. Table 1 lists the available modifiers and what they do when applied to the smart contract functions.

Table 1. A list of the modifiers defined in the smart contract and the behavior changes they make to the functions that they're applied to.

Modifier	Behavior
dealActive(dealId)	Requires the deal to be active to allow a function to proceed with its execution.
dealCompleted(dealId)	Requires the deal to be completed to allow a function to execute.
escrowDeposited(dealId)	Requires that both the buyer's and the seller's deposited to allow function execution.
noDispute(dealId)	Requires that no dispute is active to allow a function to execute.
onlyCreditScoringAgency(dealId)	Requires that the user attempting to execute a function is the credit scoring agency of the deal.
onlyBuyer(dealId)	Requires that the user attempting to execute a function is the buyer of the deal.
onlySeller(dealId)	Requires that the user attempting to execute a function is the seller of the deal.
onlyAuditor(dealId)	Requires that the user attempting to execute a function is the auditor of the deal.
onlyBuyerSeller(dealId)	Requires that the user attempting to execute a function is either the buyer or the seller of the deal.
onlyUser(dealId)	Requires that the user attempting to execute a function is either the buyer, the seller, or the auditor of the deal.
dealExists(dealId)	Requires that a deal with the provided ID exists to allow the function to execute.

2.3.4. Functions/Algorithms

These functions in the smart contract allow its users to modify the deal and add updates to it relevant to the stage of the payment process that it is at. The ability to execute a function is restricted by the rules and behaviors enforced by the modifiers in Table 1 applied to it.

2.3.4.1. Add Deal

This function allows a user to add a new deal and define this information for it: Specs UIR, Buyer, Auditor, Buyer Escrow, Seller Escrow, and Installments. It is assumed that the user that adds the deal is the seller, therefore their address is taken from their transaction information. The function generates a deal ID, creates a Deal object and adds it to the deals mapping. Figure 4 shows the code of the Add Deal function.

```
function addDeal(
    string memory specsURI,
    address buyer, address auditor, address creditScoringAgency,
    uint buyerEscrow, uint sellerEscrow,
    uint[] memory installments) public returns(Deal memory) {

    //create id
    uint dealId = dealIds.length > 0 ? dealIds.length + 1 : 1;
    STATUS status = STATUS.NEW;

    //create deal object
    Deal memory deal = Deal(
        dealId, specsURI,
        msg.sender, buyer, auditor, creditScoringAgency,
        status, "", installments, 0, 0,
        0, 0,
        sellerEscrow, buyerEscrow, false, false);

    //add deal
    deals[dealId] = deal;
    dealIds.push(dealId);

    return deal;
}
```

Figure 4. The Add Deal function code.

2.3.4.2. Set Credit Scores

Allows the credit scoring agency to provide the credit score of both the buyer and the seller. The credit scores must be provided immediately after adding the deal, otherwise, the operation will be reverted. Figure 5 shows the code of the Set Credit Scores function.

```
function setCreditScore(uint dealId, uint buyerCreditScore, uint sellerCreditScore)
    dealExists(dealId) dealActive(dealId) onlyCreditScoringAgency(dealId) public {

    require(deals[dealId].status == STATUS.NEW, "Failed! Credit scores must be set right after creating the deal.");

    deals[dealId].buyerCreditScore = buyerCreditScore;
    deals[dealId].sellerCreditScore = sellerCreditScore;
    updateStatus(dealId, STATUS.CREDIT_SCORES_ADDED, "");
}
```

Figure 5. The Set Credit Score function code.

2.3.4.3. Deposit Escrow

The Deposit Escrow function allows the buyer and the seller to deposit their escrows. Escrows are deposited after credit scores are saved to the deal. The function requires that the full amount of escrow as defined in the deal object to be deposited, otherwise the operation is rejected. Figure 6 shows the code of the Deposit Escrow function.

```

function depositEscrow(uint dealId)
dealExists(dealId) dealActive(dealId) onlyBuyerSeller(dealId) public payable {

require(deals[dealId].status == STATUS.CREDIT_SCORES_ADDED,
"Failed! Cannot deposit escrow before credit scores are provided.");

if((deals[dealId].seller == msg.sender) && (deals[dealId].sellerEscrow == msg.value)) {
deals[dealId].balance = deals[dealId].balance + msg.value;
deals[dealId].sellerEscrowDeposited = true;
contractBalance = contractBalance + msg.value;
updateStatus(dealId, STATUS.SELLER_ESCROW_DEPOSITED, "");
}
else if((deals[dealId].buyer == msg.sender) && (deals[dealId].buyerEscrow == msg.value)) {
deals[dealId].balance = deals[dealId].balance + msg.value;
deals[dealId].buyerEscrowDeposited = true;
contractBalance = contractBalance + msg.value;
updateStatus(dealId, STATUS.BUYER_ESCROW_DEPOSITED, "");
}
else {
revert("Please make sure that you deposit the exact escrow amount.");
}
}
}

```

Figure 6. The Deposit Escrow function code.

2.3.4.4. Complete Work

The Complete Work function allows the seller to announce the completion of a portion of the work as per the agreement with the buyer. It also allows the auditor to announce their verification of the completed work, and the buyer to confirm that work was completed. Figure 7 shows the code of the Complete Work function.

```

function completeWork(uint dealId, string memory comment)
dealActive(dealId) onlyUser(dealId) escrowDeposited(dealId) noDispute(dealId) public {

if(deals[dealId].seller == msg.sender) {
updateStatus(dealId, STATUS.WORK_FINISHED, comment);
}
if(deals[dealId].auditor == msg.sender) {
updateStatus(dealId, STATUS.WORK_VERIFIED, comment);
}
if(deals[dealId].buyer == msg.sender) {
updateStatus(dealId, STATUS.WORK_CONFIRMED, comment);
}
}
}

```

Figure 7. The Complete Work function code.

2.3.4.5. Deposit Installment

The Deposit Installment function allows the buyer to deposit the amount of the current installment, which is referenced by the Current Installment field of the Deal object. The buyer is required to deposit the full amount of the installment, otherwise the operation will be rejected. If this was the last installment deposited by the buyer, the system changes the status of the deal to ALL_INSTALLMENTS_DEPOSITED. Figure 8 shows the code of the Deposit Installment function.

```

function depositEscrow(uint dealId)
dealExists(dealId) dealActive(dealId) onlyBuyerSeller(dealId) public payable {

    require(
        ((deals[dealId].status == STATUS.CREDIT_SCORES_ADDED) && (msg.sender == deals[dealId].buyer))
        ||
        ((deals[dealId].status == STATUS.BUYER_ESCROW_DEPOSITED) && (msg.sender == deals[dealId].seller)),
        "Failed! Buyer must deposit their escrow after credit scores are defined. Seller deposits their escrow after buyer.");

    if((deals[dealId].seller == msg.sender) && (deals[dealId].sellerEscrow == msg.value)) {
        deals[dealId].balance = deals[dealId].balance + msg.value;
        deals[dealId].sellerEscrowDeposited = true;
        contractBalance = contractBalance + msg.value;
        updateStatus(dealId, STATUS.SELLER_ESCROW_DEPOSITED, "");
    }
    else if((deals[dealId].buyer == msg.sender) && (deals[dealId].buyerEscrow == msg.value)) {
        deals[dealId].balance = deals[dealId].balance + msg.value;
        deals[dealId].buyerEscrowDeposited = true;
        contractBalance = contractBalance + msg.value;
        updateStatus(dealId, STATUS.BUYER_ESCROW_DEPOSITED, "");
    }
    else {
        revert("Please make sure that you deposit the exact escrow amount.");
    }
}
}

```

Figure 8. The Deposit Installment function code.

2.3.4.6. Withdraw Installment

Allows the seller to withdraw the installments that are deposited by the buyer. Once all installments are withdrawn, the system marks the deal as completed by changing its status to DEAL_COMPLETED. Figure 9 shows the code of the Withdraw Installment function.

```

function withdrawInstallment(uint dealId, uint amount) onlySeller(dealId) noDispute(dealId) public payable {

    //ensure that only installments can be withdrawn
    uint maxAmount = deals[dealId].balance - deals[dealId].sellerEscrow - deals[dealId].buyerEscrow;

    if(amount <= maxAmount) {
        address payable withdrawTo = payable(msg.sender);
        uint amountToTransfer = amount;
        withdrawTo.transfer(amountToTransfer);
        deals[dealId].balance = deals[dealId].balance - amountToTransfer;
        contractBalance = contractBalance - amountToTransfer;
        updateStatus(dealId, deals[dealId].status, "AMOUNT_WITHDRAWN");
    } else {
        revert("Cannot withdraw more than the maximum allowed amount.");
    }

    uint totalEscrow = deals[dealId].sellerEscrow + deals[dealId].buyerEscrow;

    if((deals[dealId].balance == totalEscrow) &&
        (deals[dealId].currentInstallment == deals[dealId].installments.length)) {
        updateStatus(dealId, STATUS.DEAL_COMPLETED, "");
    }
}
}

```

Figure 9. The Withdraw Installment function code.

2.3.4.7. Withdraw Escrow

Once the deal is completed, this function allows the buyer and the seller to withdraw the escrows that they deposited at the beginning of the deal. Figure 10 shows the code of the Withdraw Escrow function.

```
function withdrawEscrow(uint dealId) dealCompleted(dealId) onlyBuyerSeller(dealId) public payable {
    address payable withdrawTo = payable(msg.sender);

    uint amountToTransfer = msg.sender == deals[dealId].buyer ?
        deals[dealId].buyerEscrow : deals[dealId].sellerEscrow;

    if(amountToTransfer <= deals[dealId].balance) {
        withdrawTo.transfer(amountToTransfer);
        deals[dealId].balance = deals[dealId].balance - amountToTransfer;
        contractBalance = contractBalance - amountToTransfer;
        updateStatus(dealId, deals[dealId].status, "ESCROW_WITHDRAWN");
    } else {
        revert("Transaction failed! Insufficient balance.");
    }
}
```

Figure 10. The Withdraw Escrow function code.

2.3.4.8. Initiate Dispute

The Initiate Dispute function allows the buyer or the seller to halt any further activities on the deal and initiate a dispute with the other party. Figure 11 shows the code of the Initiate Dispute function.

```
function initiateDispute(uint dealId, string memory comment) dealActive(dealId) onlyBuyerSeller(dealId) public {
    updateStatus(dealId, STATUS.DISPUTE_INITIATED, comment);
}
```

Figure 11. The Initiate Dispute function code.

2.3.4.9. Resolve Dispute

Allows the auditor to resolve an active dispute by either allowing the deal to proceed (by marking the status of the deal as DISPUTE_RESOLVED) or terminating the deal (by marking it as DEAL_TERMINATED). If the deal is terminated, all but the Settle Balances function are deactivated. Figure 12 shows the code of the Resolve Dispute function.

```
function resolveDispute(uint dealId, STATUS status, string memory comment)
dealActive(dealId) onlyAuditor(dealId) public {
    require((deals[dealId].status == STATUS.DISPUTE_INITIATED), "No dispute was found.");
    require((status == STATUS.DEAL_TERMINATED) || (status == STATUS.DISPUTE_RESOLVED),
        "Deal can only be resolved or terminated.");

    updateStatus(dealId, status, comment);
}
```

Figure 12. The Resolve Dispute function code.

2.3.4.10. Settle Balances

Allows the auditor to refund the remaining balance to the buyer and the seller. The auditor has to refund the entire balance of the deal, otherwise, the operation is reverted. The balance consists of the escrows deposited by the buyer and the seller as well as any deposited installments that are not withdrawn yet by the seller. Figure 13 shows the code of the Settle Balances function. This function can only be executed if the deal was terminated by the Resolve Dispute function.

```

function settleBalances(uint dealId, uint buyerRefund, uint sellerRefund)
dealActive(dealId) onlyAuditor(dealId) public payable {

    require((deals[dealId].status == STATUS.DEAL_TERMINATED), "Action failed! Deal not terminated.");

    uint totalRefund = buyerRefund + sellerRefund;
    require(deals[dealId].balance == totalRefund, "Failed! You must clear out the entire balance.");

    if(deals[dealId].balance == totalRefund) {

        address payable buyer = payable(deals[dealId].buyer);
        buyer.transfer(buyerRefund);
        deals[dealId].balance = deals[dealId].balance - buyerRefund;
        contractBalance = contractBalance - buyerRefund;

        address payable seller = payable(deals[dealId].seller);
        seller.transfer(sellerRefund);
        deals[dealId].balance = deals[dealId].balance - sellerRefund;
        contractBalance = contractBalance - sellerRefund;

        updateStatus(dealId, STATUS.BALANCES_SETTLED, "");

    }
}

```

Figure 13. The Settle Balances function code.

3. Results and Discussion

An experiment of a hypothetical deal was conducted to test the system functions. The cost of invoking each of the smart contract functions is in Table 2. The Remix IDE was used to write and test the code, and a local Hard Hat Node was used to test the smart contract. Results show that the system enables complete oversight and control of construction supply chain deals and the enforcement of their terms and conditions. The test results demonstrate that all system functions are working as they should in managing and handling payments. Further, test results show that the cost of handling payments on top of the system is quite low, despite the record high price of the Ethereum coin of \$3,260 at the time of the test, which makes the system an efficient, reliable, and cost-effective alternative to handling payments for construction supply chain stakeholders.

Table 2. Smart contract functions and their cost of execution.

Function	Cost in Gas	Cost in USD ¹
Deploy	3,916,767.00	0.64
Add Deal	380,959.00	0.06
Set Credit Score	86,570.00	0.01
Deposit Escrow	60,624.00	0.01
Complete Work	65,623.00	0.01
Deposit Installment	76,084.00	0.01
Withdraw Installment	65,923.00	0.01
Withdraw Escrow	55,208.00	0.01
Initiate Dispute	107,758.00	0.02
Resolve Dispute	57,571.00	0.01
Settle Balances	65,837.00	0.01

¹1 ETH = ~\$3,260, 1 Gas = 20 GWEI (at the time of writing this paper).

3.1. Functional Test Experiment and Results

The deal is as follows: A contractor wants to hire an architect to create the designs for an upcoming construction project that the contractor plans to start. The contractor and the architect agree on a deal with the total price of \$8,900, divided as 20% down payment, then two 30% payments that are associated with agreed upon milestones,

then a last payment of 20% upon the delivery of the final deliverable. The contractor and the architect want to use the proposed system to handle the payment process and ensure smooth execution of the project. The deal, thus, becomes as follows:

- For the purposes of the proposed system, the buyer is the contractor, and the seller is the architect.
- The payments are divided into 4 installments, which are \$1780, \$2670, \$2670, and \$1780.
- An independent credit scoring agency gave a rating of 1377 to the contractor and 721 to the architect. Therefore, the buyer and the seller agreed that the escrow amounts are \$445 for the contractor (buyer), and \$1,335 for the seller (architect).
- The buyer and seller agreed that another contractor trusted by both to be the auditor for the deal.

After agreeing on all the deal specifications, the buyer and seller signed a contract detailing all the terms and conditions. Then, the seller uploaded the contract to a cloud storage service and obtained a link to it. The seller then converted the installments and escrows into ETH (Ethereum currency) according to the exchange price of \$3,260 and created a Deal on the proposed system using the Add Deal function with the following details:

- The link to the contract.
- The account addresses of the buyer, auditor, and the credit scoring agency.
- The credit scores of both the buyer and the seller.
- The buyer and seller escrows, which are 0.14 ETH and 0.41 ETH, respectively.
- The installments, which are (all in ETH): 0.55, 0.82, 0.82, 0.55.

Then, once created, the system returned a Deal object with the exact same information as well as the deal's unique identifier, confirming the creation of the Deal. Figure 14 shows the result of invoking the Add Deal function. Monetary inputs and outputs are provided in the WEI format, which is a fractional representation of ETH where 1 ETH = 1 * 10⁻¹⁸. Figure 15 shows the result of invoking the Get Deals function, which returns the data of a specific deal object. Whenever a function that returns a structure object (struct object, such as the Deal object in our case), the EVM returns a "tuple" that shows the data types of data structure as well as the values corresponding to each data member. Arrays (such as the installments array in our case) are shown as a comma separated series of values. And that is what is shown in Figure 15.

```

[block:5 txIndex:-] from: 0xf39...92266 to: Deals.addDeal(string,address,address,address,uint256,uint256,uint256[]) 0x735...7f31f value: 0 wei
data: 0x839...70000 logs: 0 hash: 0x281...60ea4 Debug ^
status 0x1 Transaction mined and execution succeed
transaction hash 0xdc65f23c05543dc44cae8bc88e7b3c507c92561f060ac91722f1a2f74aff71b
block hash 0x281579a8f263ad5a0ea1f4aebf888c326e8d742b7514e8f889c371f8f60ea4
block number 5
from 0xf39fd6e51aad88f64ce6ab8827279cfff92266
to Deals.addDeal(string,address,address,address,uint256,uint256,uint256[]) 0x73511669fd4de447fed18bb79bafec93ab7f31f
gas 380959 gas
transaction cost 380959 gas
input 0x839...70000
decoded input {
  "string specURI": "https://onedrive.com/53kden0",
  "address buyer": "0x70997970C51812dc3A010C7d01b50e8d17dc79C8",
  "address auditor": "0x3C44cD086a900fa2b585dd299e03d12FA4298BC",
  "address creditScoringAgency": "0x90f798f652c44f70365c7859821f101E93b900",
  "uint256 buyerEscrow": "1400000000000000",
  "uint256 sellerEscrow": "4100000000000000",
  "uint256[] installments": [
    "5500000000000000",
    "8200000000000000",
    "8200000000000000",
    "5500000000000000"
  ]
}
    
```

Figure 14. Results of invoking the Add Deal function.


```

call [call] from: 0xf39Fd6e51aad88F6F4ce6aB8827279cFfFb92266 to: Deals.getDeals(uint256) data: 0x306...00001
from 0xf39Fd6e51aad88F6F4ce6aB8827279cFfFb92266
to Deals.getDeals(uint256) 0x73511669fD4d447FeD188B79bAFeAC93a87F31f
input 0x306...00001
decoded input {
  "uint256 dealId": "1"
}
decoded output {
  "0":
  "tuple(uint256,string,address,address,address,address,uint8,string,uint256[],uint256,uint256,uint256,uint256,uint256,uint256,bool,bool)":
  1,https://onedrive.com/53kdan0,0xf39Fd6e51aad88F6F4ce6aB8827279cFfFb92266,0x70997970C51812dc3A010C7d01b50e0d17dc79C8,0x3C44cdD86a
  900Fa2b585dd299e03d12FA42938C,0x90F79bF6E82c4f870365E785982E1f101E930906,0,550000000000000000,820000000000000000,820000000000000000
  0,550000000000000000,0,0,0,0,410000000000000000,140000000000000000,false,false"
}
logs []
    
```

Figure 15. Results of invoking the Get Deal function.

The credit scoring agency then added the credit scores of both the buyer and seller. The buyer then deposited their escrow amount of 0.14 ETH followed by depositing 0.41 ETH by the seller for their escrow amount. The buyer then deposited the down payment of 0.55 ETH. Figure 16 shows the result of invoking the Deposit Installment. The DealStatusChange event shows that the status was changed by the buyer address, the old and new statuses as a number, the total balance of the deal and the smart contract after the successful invocation of the function. The balances are shown in WEI. The total deal balance amount is 110000000000000000 WEI, which is 1.1 ETH (which equals the 0.14 buyer escrow, the 0.41 seller escrow, and the 0.55 first installment). Since there’s no other deals managed by the smart contract at the time of the experiment, the contract balance is the same as the deal balance.

```

logs [
  {
    "from": "0x2279b7a0a67db372996a5fab50d91eaa73d2ebe6",
    "topic": "0xc36e28cb15c475884135730b5cef436b79fff334ed415f85ee7baf92f01ef2d7",
    "event": "DealStatusChange",
    "args": {
      "0": "0x70997970C51812dc3A010C7d01b50e0d17dc79C8",
      "1": 3,
      "2": 7,
      "3": "",
      "4": "110000000000000000",
      "5": "110000000000000000",
      "changedBy": "0x70997970C51812dc3A010C7d01b50e0d17dc79C8",
      "from": 3,
      "to": 7,
      "comment": "",
      "dealBalance": "110000000000000000",
      "contractBalance": "110000000000000000"
    }
  }
]
    
```

Figure 16. Results of invoking the Deposit Installment function for the down payment.

After paying three of the four installments, the buyer initiated a dispute to complain about a delay caused by the seller in delivering the last deliverable. Figure 17 shows the result of initiating the dispute using the Initiate Dispute function. The auditor investigated the issue and reached an agreement with the seller and the buyer on an updated delivery date. The auditor then invoked the Resolve Dispute function and passed the DISPUTE_RESOLVED status with the comment “Buyer and seller agreed to have the last deliverable ready in 15 days”, and that’s what is shown as the result of invoking the Resolve Dispute function in Figure 18. The deal then proceeded until the last installment was paid by the buyer and withdrawn by the seller. The buyer then withdrew their escrow followed by the seller’s withdrawal of their escrow. Figure 19 shows the result of invoking Withdraw Installment where the seller withdrew the last installment. In that figure, there are logs of two DealStatusChange events; the first event is announcing the withdrawal of the amount while the second is switching the status of the deal from ALL_INSTALLMENTS_DEPOSITED to DEAL_COMPLETED. Figure 20 shows the result invoking the Withdraw Escrow by the seller, which was the last amount to be withdrawn from the Deal, which is why the Deal balance after the withdrawal was 0.

```

decoded input      {
                    "uint256 dealId": "1",
                    "string comment": "The last deliverable has been delayed by 23 days."
                    }
decoded output    -
logs              [
                    {
                        "from": "0x2279b7a0a67db372996a5fab50d91eaa73d2ebe6",
                        "topic": "0xc36e28cb15c475884135730b5cef436b79fff334ed415f85ee7baf92f01ef2d7",
                        "event": "DealStatusChange",
                        "args": {
                            "0": "0x70997970c51812dc3A010C7d01b50e0d17dc79c8",
                            "1": 7,
                            "2": 8,
                            "3": "The last deliverable has been delayed by 23 days.",
                            "4": "2740000000000000000",
                            "5": "2740000000000000000",
                            "changedBy": "0x70997970c51812dc3A010C7d01b50e0d17dc79c8",
                            "from": 7,
                            "to": 8,
                            "comment": "The last deliverable has been delayed by 23 days.",
                            "dealBalance": "2740000000000000000",
                            "contractBalance": "2740000000000000000"
                        }
                    }
                ]
    
```

Figure 17. Results of invoking the Initiate Dispute function.

```

decoded input      {
                    "uint256 dealId": "1",
                    "uint8 status": 9,
                    "string comment": "Seller and buyer agreed to extend the deadline by 15 days."
                    }
decoded output    -
logs              [
                    {
                        "from": "0x2279b7a0a67db372996a5fab50d91eaa73d2ebe6",
                        "topic": "0xc36e28cb15c475884135730b5cef436b79fff334ed415f85ee7baf92f01ef2d7",
                        "event": "DealStatusChange",
                        "args": {
                            "0": "0x3C44CdD86a900fa2b585dd299e03d12FA4293BC",
                            "1": 8,
                            "2": 9,
                            "3": "Seller and buyer agreed to extend the deadline by 15 days.",
                            "4": "2740000000000000000",
                            "5": "2740000000000000000",
                            "changedBy": "0x3C44CdD86a900fa2b585dd299e03d12FA4293BC",
                            "from": 8,
                            "to": 9,
                            "comment": "Seller and buyer agreed to extend the deadline by 15 days.",
                            "dealBalance": "2740000000000000000",
                            "contractBalance": "2740000000000000000"
                        }
                    }
                ]
    
```

Figure 18. Results of invoking the Resolve Dispute function.

3.2. Management Implications

According to Raj et al. (2022), there are three payment modalities in construction supply chains: cash on delivery, advance payment, and credit payment. Cash on delivery is when the buyer pays the price of the goods once they arrive in their location. Advance payment means that the buyer pays a down payment prior to initiating the service, then pays the rest throughout the service provision period. Credit payment means that the seller sells the goods and services to the seller by extending a line of credit, and then the buyer settles the balance later. The proposed system enables construction supply chain stakeholders to handle payments with all three modalities.

Further, the proposed system creates an environment of mutual accountability where it's in the interest of both the buyer and the seller to be committed to their obligations to each other as per the terms and conditions of the deal. This environment is supported by the transparency, decentralization, immutability, and efficiency that blockchain technology offers. That environment is also supported by the full transparency about the history of deals and financial standing of both parties, and by the risk of losing the escrow money that both parties deposit in the system as a guarantee of smooth execution of the deal. Thus, the system helps construction supply chain stakeholders overcome the many problems and issues with payment handling that are found in the literature (Nanayakkara et al., 2021), (Ramachandra and Rotimi, 2015), (Xie et al., 2019), (Swai et al., 2020), including going over budget, rework and errors, late payments, improper withholding of payments, lack of trust, and disputes over quality of work.

```

logs
[
  {
    "from": "0x2279b7a0a67db372996a5fab50d91eaa73d2ebe6",
    "topic": "0xc36e28cb15c475884135730b5cef436b79fff334ed415f85ee7baf92f01ef2d7",
    "event": "DealStatusChange",
    "args": {
      "0": "0xf39fd6e51aad88f6f4ce6a88827279cfff92266",
      "1": 12,
      "2": 12,
      "3": "AMOUNT_WITHDRAWN",
      "4": "5500000000000000",
      "5": "5500000000000000",
      "changedBy": "0xf39fd6e51aad88f6f4ce6a88827279cfff92266",
      "from": 12,
      "to": 12,
      "comment": "AMOUNT_WITHDRAWN",
      "dealBalance": "5500000000000000",
      "contractBalance": "5500000000000000"
    }
  },
  {
    "from": "0x2279b7a0a67db372996a5fab50d91eaa73d2ebe6",
    "topic": "0xc36e28cb15c475884135730b5cef436b79fff334ed415f85ee7baf92f01ef2d7",
    "event": "DealStatusChange",
    "args": {
      "0": "0xf39fd6e51aad88f6f4ce6a88827279cfff92266",
      "1": 12,
      "2": 13,
      "3": "",
      "4": "5500000000000000",
      "5": "5500000000000000",
      "changedBy": "0xf39fd6e51aad88f6f4ce6a88827279cfff92266",
      "from": 12,
      "to": 13,
      "comment": "",
      "dealBalance": "5500000000000000",
      "contractBalance": "5500000000000000"
    }
  }
]

```

Figure 19. Results of invoking the Withdraw Installment function.

```

logs
[
  {
    "from": "0x2279b7a0a67db372996a5fab50d91eaa73d2ebe6",
    "topic": "0xc36e28cb15c475884135730b5cef436b79fff334ed415f85ee7baf92f01ef2d7",
    "event": "DealStatusChange",
    "args": {
      "0": "0x70997970c51812dc3A010C7d01b50e0d17dc79C8",
      "1": 13,
      "2": 13,
      "3": "ESCROW_WITHDRAWN",
      "4": "0",
      "5": "0",
      "changedBy": "0x70997970c51812dc3A010C7d01b50e0d17dc79C8",
      "from": 13,
      "to": 13,
      "comment": "ESCROW_WITHDRAWN",
      "dealBalance": "0",
      "contractBalance": "0"
    }
  }
]

```

Figure 20. Results of invoking the Withdraw Escrow function by the buyer.

3.3. Comparison

In this section, we compare our solution with several other solutions from literature, namely, the works proposed by Raj et al. (2022), Tsai (2023), and Wu et al. (2023). Table 3 shows the comparison between our study and these studies.

3.4. Limitations and Future Work

As with any study, ours has some limitations. Firstly, the smart contract in its current design does not support the handling of paying a commission to the auditor to do their job. So, there's always a risk that the auditor may delay the payment process if they don't play their role in a timely manner. By introducing a feature where a commission is paid to the auditor by the smart contract each time, they complete a task, or an entire fee that is paid to the auditor once the deal completes, the auditor will have a motivation to complete their work so they could collect their fees. Secondly, the proposed system does not support the ability to make changes to the deal while in progress; should there be any changes that need to be made to the deal, the buyer and seller have to terminate it and create a new one with the desired changes. Introducing the ability to make changes to the deal makes the process more efficient and less time-consuming. Thirdly, the credit scoring part of the payment

process is handled off-chain. A feature can be introduced into the system where information about the buyer’s and seller’s financial standing can be shared automatically with another smart contract that represents the credit scoring agency, and that smart contract can then calculate the credit score for each of the buyer and seller in real-time and share it with the system; the system then pays the credit scoring smart contract a fee for that service. Lastly, a lot more testing is still needed to test the functionality of the system, specifically the edge cases that involve money transfers. This is to ensure that there’s always a reliable way that money can be taken out of the smart contract and doesn’t get stuck in it forever; also, to ensure that there are no security loopholes that could enable potential breaches and theft. Future research can focus on introducing the improvements suggested in this section, so the system is more capable of handling payments in a reliable, efficient, and secure manner. Further, a next step based on our study would be to pilot the system where a buyer and a seller in some construction projects manage their payments on the system to examine the system’s effectiveness in a real-world scenario and gather feedback of what improvements need to be made.

Table 3. A comparison between the system proposed in our study with other studies in literature.

Criteria	Our Solution	(Raj et al., 2022)	(Tsai, 2023)	(Wu et al., 2023)
Sector	Construction	Sector-neutral	Sector-neutral	Construction
Actors	Buyer, seller, auditor, credit scoring agency	Buyer, seller, 3PL ¹	SMEs (seller), financial institutions, core enterprises (buyer)	Client, quality inspector, main contractor, progress inspector, quantity supervisor, 3 rd party auditor
Work Tracking	Yes	Yes	Yes	Yes
Installment Tracking	Yes	Yes	Yes	No
Payment Guarantees	Escrow, credit risk, auditor	3PL	Financial institution	Consensus mechanism
Dispute Handling	Yes	No	No	No

¹3rd party logistics provider.

4. Conclusion

In this paper, we presented a blockchain-based system for handling payments in the construction supply chain. It works by allowing a buyer and a seller to create a deal, define its terms and conditions, track the agreed upon goods and services delivery until completion, and manage the payment of installments that are associated with these goods and services. The system ensures a transparent, secure, reliable, and safe environment for the buyer and the seller by requiring both parties to declare their financial standing and have it reviewed and scored by a separate credit scoring agency. Further, deals are ensured by having both parties deposit an escrow amount that serves as a guarantee that can be used to compensate one party if the other fails to comply with their obligations. The system enables initiating and solving disputes transparently on the blockchain. The system introduced the role of an auditor, which is a 3rd party in charge of verifying completed work and resolving disputes. The system functions were tested with an experiment and their costs were recorded. Test results showed that the system can enable an efficient, reliable and cost-effective environment for handling payments. Future work can focus on introducing the ability to make changes to an ongoing/incomplete deal, automating the credit scoring operation, and introducing a feature for paying the auditor a fee in exchange for their services. The system also requires further testing, ideally in a real-world scenario with a buyer and seller from a construction project to test its effectiveness and get more feedback for improvements.

Ethics Permissions

This paper does not require ethics committee approval.

Author Contributions

SH found the gap, developed the solution, tested the results, and wrote the manuscript. EÖ reviewed the manuscript and scientific proofing.

Conflict of Interest

The authors declare that there's no affiliation or involvement in an organization or entity with a financial or non-financial interest in the subject matter or materials discussed in this paper.

References

- Alnıpak, S., and Toraman, Y. (2024). Analysing the intention to use blockchain technology in payment transactions of Turkish maritime industry. *Quality and Quantity*, 58(3), 2103–2123. <https://doi.org/10.1007/s11135-023-01735-3>
- Altman, E. I., and Sabato, G. (2007). Modelling Credit Risk for SMEs: Evidence from the U.S. Market. *Abacus*, 43(3), 332–357. <https://doi.org/10.1111/j.1467-6281.2007.00234.x>
- Antipin, D. A., and Trufanova, S. V. (2021). Project financing as a tool to enhance the role of commercial banks in the construction industry. *IOP Conference Series: Earth and Environmental Science*, 751(1), 012130. <https://doi.org/10.1088/1755-1315/751/1/012130>
- Archa, Alangot, B., and Achuthan, K. (2018). Trace and Track: Enhanced Pharma Supply Chain Infrastructure to Prevent Fraud. *Ubiquitous Communications and Network Computing*, 189–195. https://doi.org/10.1007/978-3-319-73423-1_17
- Das, M., Luo, H., and Cheng, J. C. P. (2020). Securing interim payments in construction projects through a blockchain-based framework. *Automation in Construction*, 118, 103284. <https://doi.org/10.1016/j.autcon.2020.103284>
- Hamledari, H., and Fischer, M. (2021). Role of Blockchain-Enabled Smart Contracts in Automating Construction Progress Payments. *Journal of Legal Affairs and Dispute Resolution in Engineering and Construction*, 13(1). [https://doi.org/10.1061/\(ASCE\)LA.1943-4170.0000442](https://doi.org/10.1061/(ASCE)LA.1943-4170.0000442)
- Han, Y., and Fang, X. (2024). Systematic review of adopting blockchain in supply chain management: bibliometric analysis and theme discussion. *International Journal of Production Research*, 62(3), 991–1016. <https://doi.org/10.1080/00207543.2023.2236241>
- Ioannou, I., and Demirel, G. (2022). Blockchain and supply chain finance: a critical literature review at the intersection of operations, finance and law. *Journal of Banking and Financial Technology*, 6(1), 83–107. <https://doi.org/10.1007/s42786-022-00040-1>
- Kaid, D., and Eljazzar, M. M. (2018). Applying Blockchain to Automate Installments Payment between Supply Chain Parties. *2018 14th International Computer Engineering Conference (ICENCO)*, 231–235. <https://doi.org/10.1109/ICENCO.2018.8636131>
- Kumar, A., Abhishek, K., Rukunuddin Ghalib, M., Nerurkar, P., Bhirud, S., Alnumay, W., Ananda Kumar, S., Chatterjee, P., and Ghosh, U. (2021). Securing logistics system and supply chain using Blockchain. *Applied Stochastic Models in Business and Industry*, 37(3), 413–428. <https://doi.org/10.1002/asmb.2592>
- Levis, D., Fontana, F., and Ughetto, E. (2021). A look into the future of blockchain technology. *PLOS ONE*, 16(11), e0258995. <https://doi.org/10.1371/journal.pone.0258995>
- Lu, W., Li, X., Xue, F., Zhao, R., Wu, L., and Yeh, A. G. O. (2021). Exploring smart construction objects as blockchain oracles in construction supply chain management. *Automation in Construction*, 129, 103816. <https://doi.org/10.1016/j.autcon.2021.103816>
- Lu, W., Wu, L., Zhao, R., Li, X., and Xue, F. (2021). Blockchain Technology for Governmental Supervision of Construction Work: Learning from Digital Currency Electronic Payment Systems. *Journal of Construction Engineering and Management*, 147(10). [https://doi.org/10.1061/\(ASCE\)CO.1943-7862.0002148](https://doi.org/10.1061/(ASCE)CO.1943-7862.0002148)
- Motawa, I., and Kaka, A. (2009). Modelling payment mechanisms for supply chain in construction. *Engineering, Construction and Architectural Management*, 16(4), 325–336. <https://doi.org/10.1108/09699980910970824>

- Nanayakkara, S., Perera, S., Senaratne, S., Weerasuriya, G. T., and Bandara, H. M. N. D. (2021). Blockchain and Smart Contracts: A Solution for Payment Issues in Construction Supply Chains. *Informatics*, 8(2), 36. <https://doi.org/10.3390/informatics8020036>
- Omar, I. A., Jayaraman, R., Debe, M. S., Salah, K., Yaqoob, I., and Omar, M. (2021). Automating Procurement Contracts in the Healthcare Supply Chain Using Blockchain Smart Contracts. *IEEE Access*, 9, 37397–37409. <https://doi.org/10.1109/ACCESS.2021.3062471>
- Raj, P. V. R. P., Jauhar, S. K., Ramkumar, M., and Pratap, S. (2022). Procurement, traceability and advance cash credit payment transactions in supply chain using blockchain smart contracts. *Computers and Industrial Engineering*, 167, 108038. <https://doi.org/10.1016/j.cie.2022.108038>
- Ramachandra, T., and Rotimi, J. O. B. (2015). Causes of Payment Problems in the New Zealand Construction Industry. *Construction Economics and Building*, 15(1), 43–55. <https://doi.org/10.5130/AJCEB.v15i1.4214>
- Rogerson, M., and Parry, G. C. (2020). Blockchain: case studies in food supply chain visibility. *Supply Chain Management: An International Journal*, 25(5), 601–614. <https://doi.org/10.1108/SCM-08-2019-0300>
- Shemov, G., Garcia de Soto, B., and Alkhzaimi, H. (2020). Blockchain applied to the construction supply chain: A case study with threat model. *Frontiers of Engineering Management*, 7(4), 564–577. <https://doi.org/10.1007/s42524-020-0129-x>
- Sigalov, K., Ye, X., König, M., Hagedorn, P., Blum, F., Severin, B., Hettmer, M., Hückinghaus, P., Wölkerling, J., and Groß, D. (2021). Automated Payment and Contract Management in the Construction Industry by Integrating Building Information Modeling and Blockchain-Based Smart Contracts. *Applied Sciences*, 11(16), 7653. <https://doi.org/10.3390/app11167653>
- Studer, W., and De Brito Mello, L. (2021). Core Elements Underlying Supply Chain Management in the Construction Industry: A Systematic Literature Review. *Buildings*, 11(12), 569. <https://doi.org/10.3390/buildings11120569>
- Swai, L. P., Arewa, A. O., and Ugulu, R. A. (2020). Unfair Payment Issues in Construction: Re-Thinking Alternative Payment Method For Tier-1 Contractors to Subcontractors. *International Conference on Civil Infrastructure and Construction (CIC 2020)*, 84–93. <https://doi.org/10.29117/cic.2020.0012>
- Tsai, C.-H. (2023). Supply chain financing scheme based on blockchain technology from a business application perspective. *Annals of Operations Research*, 320(1), 441–472. <https://doi.org/10.1007/s10479-022-05033-3>
- Wu, L., Lu, W., and Chen, C. (2023). Resolving power imbalances in construction payment using blockchain smart contracts. *Engineering, Construction and Architectural Management*. <https://doi.org/10.1108/ECAM-03-2023-0194>
- Xie, Zheng, Zhang, and Li. (2019). Effects of Payment Delays at Two Links in Payment Chains on the Progress of Construction Projects: System Dynamic Modeling and Simulation. *Sustainability*, 11(15), 4115. <https://doi.org/10.3390/su11154115>
- Xue, X., Wang, Y., Shen, Q., and Yu, X. (2007). Coordination mechanisms for construction supply chain management in the Internet environment. *International Journal of Project Management*, 25(2), 150–157. <https://doi.org/10.1016/j.ijproman.2006.09.006>
- Zhang, X., Liu, T., Rahman, A., and Zhou, L. (2023). Blockchain Applications for Construction Contract Management: A Systematic Literature Review. *Journal of Construction Engineering and Management*, 149(1). [https://doi.org/10.1061/\(ASCE\)CO.1943-7862.0002428](https://doi.org/10.1061/(ASCE)CO.1943-7862.0002428)
- Zhou, H., Milani Fard, A., and Makanju, A. (2022). The State of Ethereum Smart Contracts Security: Vulnerabilities, Countermeasures, and Tool Support. *Journal of Cybersecurity and Privacy*, 2(2), 358–378. <https://doi.org/10.3390/jcp2020019>