



Blockchain-Based Solution for Proof of Delivery of Physical Assets

Haya R. Hasan^(✉) and Khaled Salah^(✉)

Department of Electrical and Computer Engineering,
Khalifa University of Science, Technology and Research, Abu Dhabi, UAE
{haya.hasan, khaled.salah}@kustar.ac.ae

Abstract. To date, building a highly trustworthy, credible, and decentralized proof of delivery (POD) systems to trace and track physical items is a very challenging task. This paper presents a blockchain based POD solution of shipped physical items that uses smart contracts of Ethereum blockchain network, in which tracking, and tracing activities, logs, and events can be done in a decentralized manner, with high integrity, reliability, and immutability. Our solution incentivizes each participating entity including the seller, transporter, and buyer to act honestly, and it totally eliminates the need for a third party as escrow. Our proposed POD solution ensures accountability, punctuality, integrity and auditability. Moreover, the proposed solution makes use of a Smart Contract Attestation Authority to ensure that the code follows the terms and conditions signed by the participating entities. It also allows the cancellation of the transaction by the seller, buyer and transporter based on the contract state. Furthermore, the buyer can also ask for a refund in certain justifiable cases. The full code, implementation discussion with sequence diagrams, testing and verification details are all included as part of the proposed solution.

Keywords: Proof of delivery · Blockchain · Ethereum · Smart contracts

1 Introduction

With the widespread of technology and the internet, online shopping or trading have become part of people's daily activity. Often at the comfort of their homes, people start searching for a desirable item and wonder if there is an online vendor that can provide the item in a perfect condition to their door step. Meeting the needs of today's world, lots of online stores have launched and provided delivery services and even world-wide shipping. Thus, there is an immense need to have a solution that provides proof of delivery of any physical item such as a piece of clothing, book, home essentials etc. delivered between any two parties.

Proof of Delivery (POD) or 'last mile' of delivery is crucial as it shows that an item has reached its final and required destination. In real world, courier and delivery service companies use trackers and proof of delivery systems to ensure that their customers' needs are met on time and without delays. Not only punctuality is important, but also the delivery of the item as is from the initial source is extremely vital.

Today's proof of delivery systems are typically based on signed papers and documents which are carried with the transporter to the recipient. Other courier services might depend on an electronic hand-held device used to obtain the signature of the recipient with a valid ID. This is cumbersome and does not provide total trust for delivery, whereby there is no true and genuine verification by the courier for the signature and the ID of the recipient which can be fake. Even, the courier cannot be trusted. Furthermore, online retailers depend on a third party for shipment. For instance, Amazon depends on several regional courier companies for their delivery services, such as UPS, FedEx, DHL, Pilot and many others [1, 2]. In addition, today's delivery service is completely centralized, costly, and extremely hard to manage. In general, centralized systems suffer from privacy invasion, being a single point of failure, and mistrust which can lead to corruption and attacks.

Problem Statement. *To date, trusted online trading between two unknown parties is not yet established without a centralized trusted third party. There is an immense need for a proof of delivery and tracking of shipped items with a highly trusted, secure, and decentralized traceability and auditability.*

Blockchain is a new disruptive technology that possesses a lot of features that make it ideal for tracking systems. Hence, making it a perfect candidate for creating proof of delivery solutions. Blockchain is an immutable, distributed ledger that is tamper-proof with ordered logs [3, 4]. Ethereum smart contracts made blockchain programmable, as it allowed the execution of code, making it even more powerful [4]. An ideal proof of delivery system should satisfy the following desirable characteristics:

- **Accountability:** It should be possible to trace back the actions performed to the system to the actual initiating entity.
- **Authorization:** Each party in the system is allowed to only perform certain roles.
- **Auditability:** It should be possible to track all activities performed by the acting entities and hence, to trace back the item's state and route.
- **Integrity:** No one should be able to modify the audits and agreed upon terms and conditions.
- **Punctuality:** The system should be able to time every action and deliver the item on time to the customer.
- **Honesty:** Each of the participating entities (seller, transporter and buyer) should be incentivized to act honest and do their part legitimately.

In this paper, we propose a blockchain solution for the proof of delivery of physical items that solves the issue of trust, tracking, tracing back and proves the item reached its final and legitimate destination. The solution can be extended to include intermediary destinations before the final one and can be easily integrated with a Know Your Customer (KYC) protection to add an extra layer of security. We show in our solution that all parties involved in the contract are all equally trusted with the same authority levels. Their roles allow them to execute certain functions only, hence managing the contract flow using role restriction. The main contributions of the paper can be summarized as follows:

- We propose a blockchain-based proof of delivery solution that utilizes tamper-proof logs for auditability and traceability.

- Our solution utilizes an equal agreed upon collateral to incentivize each of the participating entities to act honest. Our PoD solution ensures integrity of the signed terms and conditions form by using InterPlanetary File System (IPFS) hash in the smart contract. Also, the smart contract code is attested by the smart contract attestation authority to ensure that it follows the signed terms and conditions.
- We demonstrate accountability in our solution by using keys and hashes for verification of the true legitimate receiver. We also show how our solution addresses issues related to customer loyalty and delivery punctuality. Refund and cancellation are also taken care of to preserve the rights of the seller, buyer and transporter.
- We present a complete system architecture, sequence diagrams and the full code with the implementation and testing details.

The remainder of this paper is organized as follows. Section 2 provides the related work. Section 3 presents the system architecture of the proposed solution. Section 4 describes the important aspects of the implementation. Section 5 illustrates the testing and validation of the smart contract code and Sect. 6 concludes the paper.

2 Related Work

This section presents the work that is available in the literature for proposed algorithms and techniques for proof of delivery of physical items. All blockchain solutions try to solve the trust issue between the seller, buyer and transporter while tracking the item's state through the logged events.

For instance, [5] proposes using a secure hash and a key that is given by the seller to the transporter along with the item. Once the transporter arrives to the destination, the buyer would enter the key and the hash of the key is compared with the hash already available in the contract. This method is simple and easy to implement. It uses the contract as an escrow. Therefore, the item's price would only be given to the seller once the hash verification is done. However, it involves trusting the transporter not to manipulate the key. Furthermore, its success depends on all the parties acting honest and trust worthy and this cannot be guaranteed.

On the other hand, localethereum [6], an Ethereum market place, does not use the contract as an escrow. It uses a third party as a funded escrow. The seller and buyer or traders agree upon a trusted third party as escrow that they would trust till the end of the transaction. Also, in case of dispute, the involved parties can allow localethereum to act as an arbitrator. Although using a third party as an escrow costs more and also requires more incentives for trust and honesty, having an arbitrator incase of a dispute makes it more convenient.

Furthermore, BitBay [7], a decentralized marketplace which provides a platform for users to trade and sell, uses a 'double-deposit escrow'. This means that a collateral is deposited by the traders or buyer and seller that is equal to double the item price being traded or sold. BitBay eliminates the need of a trusted third party as an escrow. Hence, it uses the contract itself as an escrow that takes all the deposits until the transaction is completed. BitBay also does not act as a moderator themselves in case of issues unlike localethereum. However, using BitBay the delivery of a physical item would require

trusting a third party for the transportation. There is no guarantee that the transporter would not manipulate the delivered item or create a delay.

An important point is also to be able to integrate the system with a KYC procedure to verify the identity of the parties involved in the contract. For instance, this can be achieved by using a certification body that takes the signed personal information of a user and stores their hash in the blockchain. Later on, when a user would like to register in a service, the service providers would verify the submitted signature hash with the hash value of the personal information acquired from the certification body. If both the hashes match, then the identification is complete [8].

3 Proposed Blockchain Solution

In this section, we propose a solution that utilizes the Ethereum blockchain to create a system that is decentralized, provides trust, and uses immutable logs and events. This system is a solution for selling physical items with a proof of delivery and without the need of an intermediary service such as a broker or an agent. This is done by using smart contracts which facilitate the automation of the process and help in saving the history of all transactions without alterations.

3.1 System Overview and Design

The proposed solution is focused on proof of delivery of physical items between a seller and an interested buyer. A transporter is also part of the contract to ensure the item delivery is carried out and all other parties are notified about the status of the item during delivery. An agreed upon and trusted arbitrator by the seller, buyer and transporter is also part of the contract and can only step in incase of a dispute. Each of the mentioned participants possess an Ethereum address and they all sign an agreement form which has all the terms and conditions of the contract. Furthermore, an important part of our solution, is the attestation of the contract from a certified trusted authority, Smart Contract (SC) Attestation Authority. The attestation guarantees to all participating entities that the smart contract follows the signed agreement terms and conditions. The roles of the Ethereum entities in the smart contract are as follows:

Seller: The seller is the owner of the item to be sold. The seller creates the contract, signs the agreement terms and conditions form, deposits a collateral, and provides keys to both the buyer and transporter.

Buyer: The buyer is the entity that shows interest in the item being sold. The buyer agrees to the terms and conditions, deposits a collateral along with the item price, and requests an item key to be able to take the item from the transporter when delivered.

Transporter: The transporter's main role is to deliver the item as received between the seller and buyer.

Arbitrator: The arbitrator is a trusted third party that is agreed upon by the seller, buyer and transporter who will only get involved in case of a dispute to solve the issues off the chain.

SC Attestation Authority: The smart contract attestation authority ensures that the smart contract complies with the terms and conditions signed by the involved parties in the agreement form.

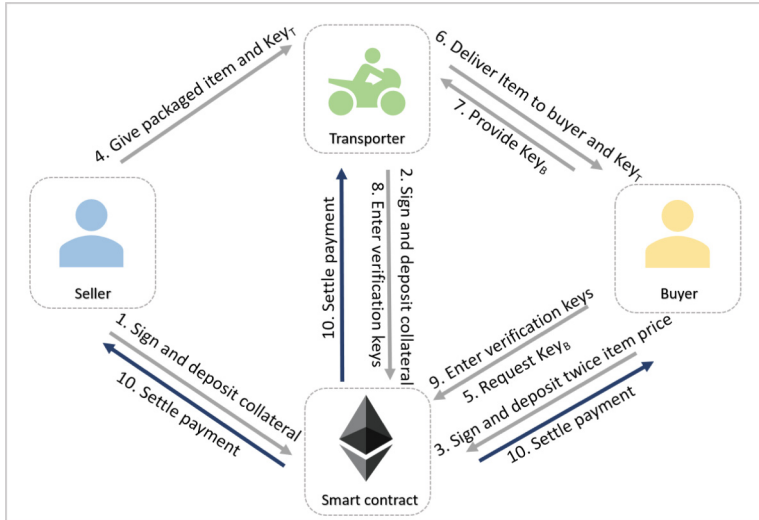


Fig. 1. System architecture of the Ethereum-based solution showing the main participating entities participating in a successful transaction

The smart contract of the proposed solution follows a certain algorithm that flows in a sequence that should be followed by the participating entities to preserve everyone's right. All actions that take place off the chain are accompanied by functions in the contract that trigger logged events and notifications. The contract receives the agreed upon collateral which is in our case twice the item price from the seller, buyer and transporter as they are equally trusted by each other and by the smart contract. The collateral can be of any reasonable value as long as it is equal among all entities. The funds are held by the contract, so the contract acts as escrow until the payment is settled based on the events and verification results. The smart contract contains the following:

- **Modifiers:** Modifiers ensure that transactions and functions are executed by the right legitimate entities and that the payable functions only accept the correct intended payment before proceeding. Modifiers change the function that uses them to allow it to execute based on the result of another code that is first executed inside the modifier. For example, requiring the collateral by each entity to be twice the actual price of the item or requiring a certain function to be executed by only the seller, buyer, transporter or any one of them is specified using the modifiers: `costs()`, `OnlySeller()`, `OnlyBuyer()`, `OnlyTransporter()`, `OnlySeller_Buyer_Transporter()`.
- **Events:** After the execution of a function, an event is used to create notifications and saved logs. Events help in tracing and in notifying all participating parties about the current state of the contract and current activities.

- Variables:** Variables help in saving important values that preserve the state of the contract as it changes along with the functions. Variables used in the contract hold addresses of the participating entities, IPFS hash of the agreement terms and conditions form, item’s details such as price and ID, and current contract state.

Figure 1 shows the system architecture of the proposed blockchain solution that shows the transactions between the seller, buyer and transporter. All the parties sign the terms and conditions agreement form and agree to its content by depositing a collateral that is twice the item price. Then the seller prepares the item and hands it over to the transporter along with a key (Key_T). Every item has two keys that are given by the seller, i.e. a key that is given to the transporter and a key that is handed over to the buyer (Key_B). The transporter delivers the item to the buyer and they both exchange their keys. This ensures that the transporter has reached the intended buyer. Both of the transporter and buyer enter the keys to the contract and verification takes place. The smart contract computes the hashes of the keys entered and if the hashes match, then the payment is settled. The buyer is refunded one item price, the transporter gets back twice the item price that was deposited as collateral with 10% of the item price additional payment for the delivery service. Lastly, the seller gets the rest of the ether deposits which include the rest of the deposited collateral and the payment of the item price by the buyer.

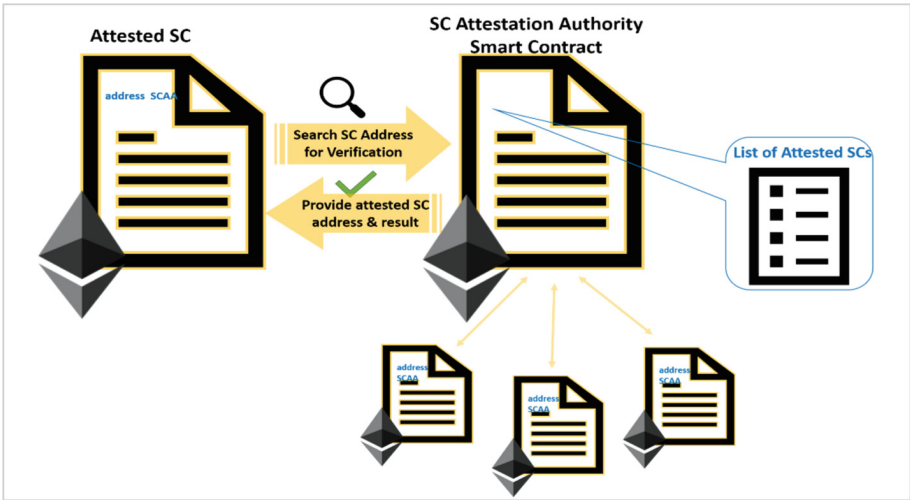


Fig. 2. Attested SC pointing to the Ethereum address of the SC Attestation Authority which also has a list of all the Ethereum addresses of the attested contracts

The contract used in the blockchain solution is an attested contract. Hence, the contract code and flow was verified and attested by the Smart Contract (SC) Attestation Authority. Therefore, the attested SC includes the Ethereum address of the attestation authority that verified the contract code and ensured that it agrees to the terms and

conditions in the agreement form that is signed by all the parties of the contract. Moreover, the SC Attestation Authority has a complete list of the all the Ethereum addresses of the attested contracts. This provides a way for the participating entities to trust the contract content and to be able to verify that it has truly been attested and is now trustable. Figure 2 shows the attested smart contract and the SC Attestation Authority contract along with their relationship.

4 Implementation Details

The smart contract code is written in Solidity using the web-based IDE, Remix. The code focuses on three main entities, the seller, transporter and buyer to acquire the proof of delivery of a physical item. However, there are two other entities that also contribute in making the contract more trustable. The arbitrator steps in incase of a dispute and the SC Attestation Authority attests and verifies that the contract code matches the terms and conditions signed by each of the actively participating entities.

Figure 3, shows the sequence diagram that demonstrates the flow of the code. The code represents a seller who has an item and would like to sell it to an interested buyer. Therefore, a transporter is needed to deliver the packaged item between the seller and the buyer. At the beginning of the contract, the seller, transporter and buyer are all required to sign the attested terms and conditions agreement form. The agreed upon collateral which is twice the item price is automatically deducted from each party that signs the form. The form's IPFS hash is available in the contract so that each of the seller, buyer and transporter will be able to access the same form using the IPFS hash provided in the contract. This will ensure the data integrity of the content of the agreement form is well maintained.

The transporter is provided with a key along with the physical item. The transporter's key (Key_T) is handed over to the transporter while the key is kept unrevealed to the buyer. On the other hand, the buyer also has a key which the transporter is unaware of its content, (Key_B). The keys are exchanged between the transporter and the buyer upon the successful delivery of the item. The hashes of the concatenated strings is computed twice, once by the transporter and another time by the receiver. Verification is then done by the contract to check that the hashes match. If the hashes match, then the transaction is successful and the buyer gets back half of the deposited collateral. The transporter also gets back the deposited collateral in addition to the transportation fees and the rest goes to the seller. The above is demonstrated in Alternative 1, in Fig. 3. Alternative 2 and 3 that are illustrated in Fig. 3, take place if the verification hashes do not match or any of the transporter or receiver fail to enter the verification keys on time. In this case, the deposited collateral gets transferred to the trusted arbitrator that was agreed upon by all the parties at the beginning of the contract.

It is important to note that the contract code is written in a way that allocates a time window for each action to preserve the rights of the other parties involved. For instance, when the packaged item is received by the transporter, the time is stored in the contract and is later checked to verify if the delivery time was exceeded or not. The buyer has the right to request a refund if the time is exceeded. This can be done using the `refund()` function.

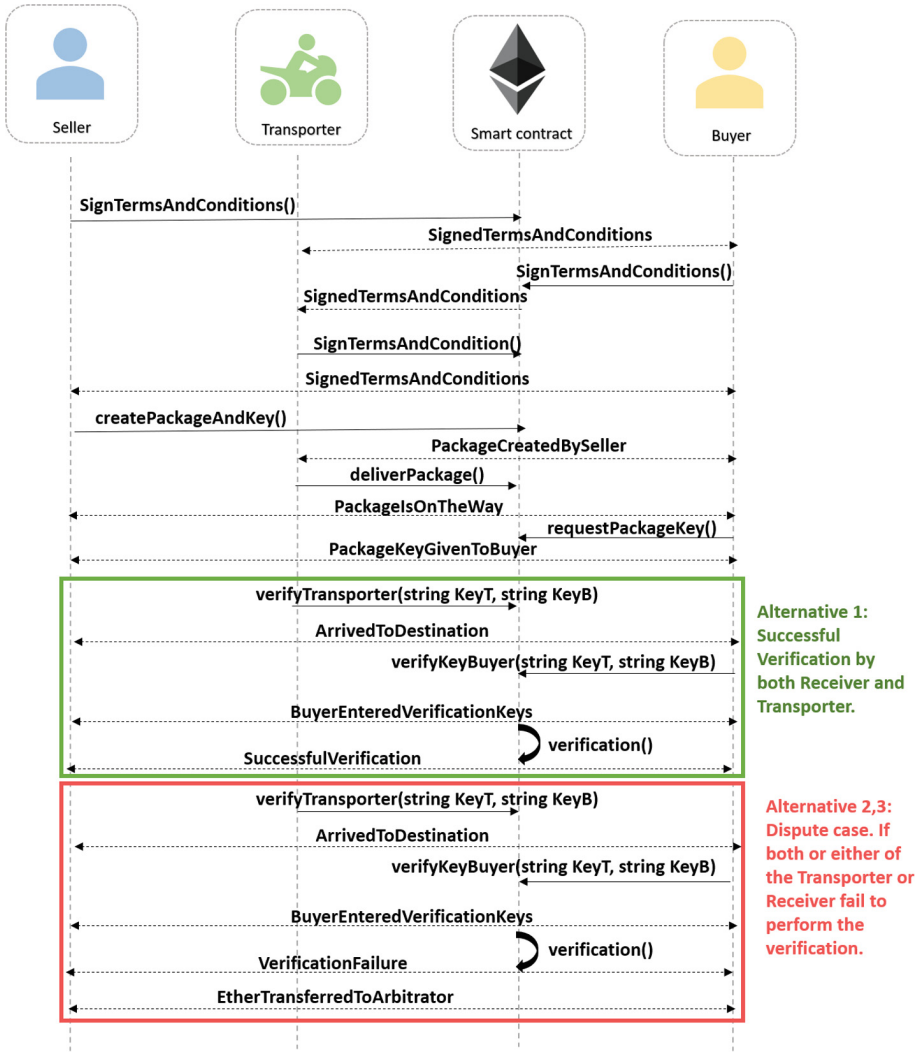


Fig. 3. Sequence diagram of the smart contract code that shows the flow for a successful and an unsuccessful transaction

Furthermore, the buyer also needs to enter the keys for verification after the transporter has called the `verifyTrasporter()` function by a maximum of 15 min. If the buyer exceeds the time window allocated for verifying the keys, the transporter has the right to leave after calling the `BuyerExceededTime()` function. In this function, the time window is checked, and the arbitrator gets involved to solve the issue off the chain. Figure 4, shows the implementation of the verification functions of both the transporter and buyer.


```

function verifyTransporter(string keyT, string keyR) OnlyTransporter{
    require(state == contractState.PackageKeyGivenToBuyer);
    ArrivedToDestination("Transporter Arrived To Destination and entered keys " , msg.sender);
    verificationHash[transporter] = keccak256(keyT, keyR);
    state = contractState.ArrivedToDestination;
    startEntryTransporterKeysBlocktime = block.timestamp;
}
function verifyKeyBuyer(string keyT, string keyR) OnlyBuyer{
    require(state == contractState.ArrivedToDestination);
    BuyerEnteredVerificationKeys("Receiver entered keys, waiting for payment settlement", msg.sender);
    verificationHash[buyer] = keccak256(keyT, keyR);
    state = contractState.buyerKeysEntered;
    verification();
}
function BuyerExceededTime() OnlyTransporter{
    require(block.timestamp > startEntryTransporterKeysBlocktime + buyerVerificationTimeWindow &&
    state == contractState.ArrivedToDestination);
    BuyerExceededVerificationTime("Dispute: Buyer Exceeded Verification Time", msg.sender);
    verification();
}
}

```

Fig. 4. Contract code showing the key verification functions and time window checking (The full code is available at: https://github.com/smartcontract694/POD_PhysicalItems).

Once the buyer enters the keys, the `verification()` function is automatically called as it is an internal function. The function compares the hashes of the transporter and buyer. If the hashes are equal, the verification is successful and the buyer receives half of the deposited collateral, the transporter receives his full collateral in addition to his transportation fees and the seller receives back the rest of the deposits. However, if the hashes are not equal, which could mean that either of the transporter or receiver failed to enter the right keys, all deposits are transferred to the arbitrator and the dispute is solved off the chain. This is demonstrated in Fig. 5 which shows the code of the `verification()` function.

```

function verification() internal{
    require(state == contractState.buyerKeysEntered);
    if(verificationHash[transporter] == verificationHash[buyer]){
        SuccessfulVerification("Payment will shortly be settled , successful verification!");
        buyer.transfer(itemPrice);
        transporter.transfer((2*itemPrice) + ((10*itemPrice)/100)); //receiver gets 10% of item price delivered
        seller.transfer((2*itemPrice)+((90*itemPrice)/100));
        state = contractState.PaymentSettledSuccess;
    }
    else { //trusted entity the Arbitrator resolves the issue
        VerificationFailure("Verification failed , keys do not match. Please solve the dispute off chain. No refunds.");
        state = contractState.DisputeVerificationFailure;
        arbitrator.transfer(this.balance); //all ether with the contract
        state = contractState.EtherWithArbitrator;
        EtherTransferredToArbitrator("Due to dispute all Ether deposits have been transferred to arbitrator ", arbitrator);
        state = contractState.Aborted;
        selfdestruct(msg.sender);
    }
}
}

```

Fig. 5. Contract code showing the verification function (The full code is available at: https://github.com/smartcontract694/POD_PhysicalItems).

Furthermore, each function in the contract requires a certain previous state for it to be successfully executed. This plays a vital role in the ability of a certain entity to cancel the purchase. The buyer can cancel the purchase without penalty if the item has not yet been delivered and a transporter can cancel the delivery before taking the item from the seller. A mapping that maps the address of each of the buyer, seller and transporter along with a boolean is used to control the cancellation. At the beginning all items of the mapping are initialized with true. After the package is created and the key_T is handed to the transporter, the transporter and seller cannot cancel the purchase. Hence, their mapping items are changed to false as illustrated in Fig. 6. Moreover, when the package has been handed over to the transporter, the buyer cannot cancel the purchase and the mapping of the buyer also gets changed to false. Hence, the cancellation function always checks the state to know which stage the item is in and if the cancelling entity has the right to cancel or not using the mapping details.

```

//sender or transporter can cancel the transaction before the package is created with the key.
function createPackageAndKey() OnlySeller returns (string){
    require(state == contractState.MoneyWithdrawn);
    PackageCreatedBySeller("Package created and Key given to transporter by the sender ", msg.sender);
    state = contractState.PackageAndTransporterKeyCreated;
    cancellable[msg.sender] = false;
    cancellable[transporter]=false;
    return "0x378032c1a780b9ab6b0e29afb705ee";
}
//receiver can cancel as long as the package is not with the transporter
function deliverPackage() OnlyTransporter{
    require(state == contractState.PackageAndTransporterKeyCreated);
    startDeliveryBlocktime = block.timestamp;//save the delivery time
    cancellable[buyer] = false;
    PackageIsOnTheWay("The package is being delivered and the key is received by the ", msg.sender);
    state = contractState.ItemOnTheWay;
}

```

Fig. 6. Functions in the contract code that show ‘state’ requirements (The full code is available at: https://github.com/smartcontract694/POD_PhysicalItems).

5 Testing and Validation

The smart contract code has been tested for several important aspects and test cases that are discussed in this section.

5.1 Test Case 1: Payable Collateral Amount

The contract code has one payable function. This function uses a modifier called `costs()` which ensures that the ether deposited is equal to the agreed upon collateral which is twice the item price. Figure 7 shows the logs of a successful deposit withdrawal made by the sender upon signing the terms and conditions agreement form. The figure also shows that the successful transaction took a value of 4 ether since the item price is 2 ether in this contract.

```
[
  {
    "topic": "9cea04aff3f776ebd39f71b9106908dea3a3116faf10e899b8813dc9d376bd67",
    "event": "TermsAndConditionsSignedBy",
    "args": [
      "Terms and Conditiond verified : ",
      "0xca35b7d915458ef540ade6068dfe2f44e8fa733c"
    ]
  },
  {
    "topic": "dbd2e985d8ff3bc981ac1b377adad9122a998155e8573a6425d2dd3c541a9b08",
    "event": "collateralWithdrawnSuccessfully",
    "args": [
      "Double deposit is withdrawn successfully from: ",
      "0xca35b7d915458ef540ade6068dfe2f44e8fa733c"
    ]
  }
]
]
400000000000000000 wei
```

Fig. 7. Log details of a successful deposit transaction by the seller.

5.2 Test Case 2: Role Restriction

To ensure the proper functionality of the code based on the actions of the participating entities, the contract's functions are restricted based on the role. All the functions of the contract have been tested successfully for role restriction. Hence, if the arbitrator for instance whose address is "0x583031d1113ad414f02576bd6afabfb302140225" tries to request the package key of the buyer, the transaction fails as illustrated in Fig. 8.

status	0x0 Transaction mined but execution failed
from	0x583031d1113ad414f02576bd6afabfb302140225
to	POD_PhysicalItems.requestPackageKey() 0x0dcd2f752394c41875e259e00bb44fd505297caf

Fig. 8. Log details of a failed transaction due to role restriction.

5.3 Test Case 3: Matching Verification Keys

A successful test case was tested to ensure that the verification of the hashes works as expected and the payment settlement. As can be seen in Fig. 9, the keys entered provide similar hashes and as a result, the transaction is successful. The buyer gets back a deposit of 2 ether, the transporter gets back 2 ether in addition to 10% of the item price as transportation fees and the seller gets back the rest of the deposited collateral. Figure 10 shows the ether deposits at the end of the successful transaction, with the seller, transporter and buyer having 101.8, 100.2 and 98 ether respectively.

```

{
  "topic": "d58c2147902b2eb7ba2e7297446aad01cc528c23b8b37c0bf79b1a8dda675f80",
  "event": "BuyerEnteredVerificationKeys",
  "args": [
    "Reciever entered keys, waiting for payment settlement",
    "0x4b0897b0513fdc7c541b6d9d7e929c4e5364d2db"
  ]
},
{
  "topic": "5ceb8be1ca60b29392c4eba246b4df328b94b21f7dc3c3925630e318bae1bc31",
  "event": "SuccessfulVerification",
  "args": [
    "Payment will shortly be settled , successful verification!"
  ]
}

```

Fig. 9. Log details of a successful transaction and payment settlement.



Account	0x4b0...4d2db (97.9999999999985227  
Gas limit	0xca3...a733c (101.799999999993549836 ether) 0x147...c160c (100.19999999999854868 ether)
Value	0x4b0...4d2db (97.99999999999852273 ether) 0x583...40225 (100 ether)

Fig. 10. Ether deposits at the end of a successful transaction.

5.4 Test Case 4: Dispute and Arbitrator Role

In the case of the keys entered by the buyer and transporter are not matching, a dispute occurs and the arbitrator steps in. All the ether deposits that are held with the contract are transferred to the arbitrator and the dispute is solved off the chain. As can be seen in Fig. 11, the arbitrator at the end of an unsuccessful transaction has 12 ether, since each of the three parties has deposited 4 ethers when signing the agreement form. Figure 12, shows the events that take place during a dispute and before the contract gets to the ‘aborted’ state.



Account	0x4b0...4d2db (95.9999999999987394  
Gas limit	0xca3...a733c (95.999999999993549836 ether) 0x147...c160c (95.99999999999831534 ether)
Value	0x4b0...4d2db (95.99999999999873941 ether) 0x583...40225 (112 ether)

Fig. 11. Ether deposits at the end of an unsuccessful transaction.

```

{
  "topic": "d58c2147902b2eb7ba2e7297446aad01cc528c23b8b37c0bf79b1a8dda675f80",
  "event": "BuyerEnteredVerificationKeys",
  "args": [
    "Reciever entered keys, waiting for payment settlement",
    "0x4b0897b0513fdc7c541b6d9d7e929c4e5364d2db"
  ]
},
{
  "topic": "4ed0570313882c4b1660d5a3a0fdca6fbeddba1bfca75a125dcd7e403f192442",
  "event": "VerificationFailure",
  "args": [
    "Verification failed , keys do not match. Please solve the dispute off chain. No
refunds."
  ]
},
{
  "topic": "fc0cf60f1f5c19e05c32dfb2dff6c9823cdc87846e26a2e2335a01cb184ab2f3",
  "event": "EtherTransferredToArbitrator",
  "args": [
    "Due to dispute all Ether deposits have been transferred to arbitrator ",
    "0x583031d1113ad414f02576bd6afafbf302140225"
  ]
}
}

```

Fig. 12. Events at the end of an unsuccessful transaction.

6 Conclusion

This paper has presented a blockchain solution which facilitates the trading and tracking of sold items between two parties in a decentralized way. The solution provides a proof of delivery of physical items taking advantage of the security and immutability that blockchain provides. Our proposed solution is generic enough and can be applied to almost all shipped physical items and assets. In this paper, we focused on providing, implementing, and testing the smart contract code and algorithm of the PoD solution that show cases the ability to prove the delivery of an item using an equal deposited collateral by the seller, transporter and buyer. In the paper, we showed and discussed how our solution can provide PoD key features and requirement that include integrity, accountability, authorization, punctuality and honesty. As a future work, we plan to extend our solution to implement aspects related to confidentiality and privacy. Also, work is underway to develop completed DApps with different views for seller, buyer, and transporter.

References

1. "Help & Customer Service", Amazon Shipment Updates via Text Terms and Conditions. https://www.amazon.com/gp/help/customer/display.html/ref=hp_left_v4_sib?ie=UTF8&nodeId=201910790
2. "Help & Customer Service", Shipping Carrier Contacts. https://www.amazon.com/gp/help/customer/display.html/ref=hp_ss_qs_v3_rt_ci?ie=UTF8&nodeId=201117350
3. Toyoda, K., et al.: A novel blockchain-based product ownership management system (POMS) for anti-counterfeits in the post supply chain. *IEEE Access* **PP**(99), 1 (2017)
4. Christidis, K., Devetsikiotis, M.: Blockchains and smart contracts for the internet of things. *IEEE Access* **4**, 2292–2303 (2016)

5. “Two party contracts”, Dapps for Beginners (2018). <https://dappsforbeginners.wordpress.com/tutorials/two-party-contracts/>
6. “How Our Escrow Smart Contract Works”, localethereum.com’s official blog (2018). <https://blog.localethereum.com/how-our-escrow-smart-contract-works/>
7. “Double Deposit Escrow – BitBay”, BitBay (2018). <https://bitbay.market/double-deposit-escrow>. Accessed 28 Mar 2018
8. “Open Source Products”, KYC (Know Your Customer) (2018). <https://guide.blockchain.z.com/en/docs/oss/kyc/>