# Proof of Delivery of Digital Assets Using Blockchain and Smart Contracts

**Article** *in* IEEE Access · October 2018

DOI: 10.1109/ACCESS.2018.2876971

**2 authors:**

Haya Hasan
Khalifa University
**27** PUBLICATIONS **527** CITATIONS
SEE PROFILE

Khaled Salah
Khalifa University
**316** PUBLICATIONS **5,872** CITATIONS
SEE PROFILE

# Proof of Delivery of Digital Assets using Blockchain and Smart Contracts

Haya R. Hasan, and Khaled Salah

***Abstract*—There is an immense need of a Proof of Delivery (PoD) of todays digital media and content, especially those that are subject to payment. Current PoD systems are mostly centralized and heavily dependent on a Trusted Third Party (TTP) especially for payment. Such existing PoD systems often lack security, transparency and visibility, and are not highly credible, as the TTP can be subject to failure, manipulation, corruption, compromise and hacking. Blockchain is used to create a decentralized solution. Utilizing blockchain's immutable and tamper-proof logs, accountability and auditability can be easily achieved. Ethereum which makes blockchain a programmable distributed ledger is used in our implemented solution to create a PoD solution for the digital media. The solution uses a smart contract to allow customers to request the content and be uniquely identified using tokens derived from their Ethereum Addresses (EA). The solution involves the owner of the digital media, the file server and the customers. All participating entities are incentivized to act honestly. Our solution includes off-chain secure download activity involving the file server and the customers. A security analysis of our proposed system has been provided. The full code of the Smart Contract has been made publicly available on Github.**

***Index Terms*—PoD, Ethereum, Blockchain, Digital Content, Double collateral, Trust.**

## I. Introduction

With the ease and the advancement of communication now a days, anyone can be a creator. Creating music, videos, blogs, eBooks, photographs and articles is not limited to experts of the field only. Consequently, emerging digital market places are now found that deliver digital products to consumers who are interested in the content delivered by other digital providers. According to statista, the U.S. revenue from digital media alone is 43.2 billion dollars [1]. In order to meet the increase in demand, digital market places offer a space for both providers and consumers to connect with each other.

However, it is crucial to have a proof of delivery (PoD) of the digital content to ensure that the requested digital media has reached successfully to its customer. PoD also assures both parties involved that the digital content reached untampered and as requested. This helps in reserving the rights of both the parties and establish trust for any future events.

Currently there are several digital market places that offer a space for both the digital creator as well as the consumers to connect. Although such platforms offer huge visibility they take a huge portion of the sales. Hence, leaving the creator with a very small profit. There are also other digital services that are more flexible in terms of the branding and the revenue share but the main issue in all the current solutions is that they are centralized.

Centralized system suffer from being a single point of failure. Moreover, they lack transparency and the authority is not distributed equally. Hence, they are more prone for corruption. Furthermore, such systems also rely on a trusted third party (TTP) for payment. Thus, making them unreliable and not trustworthy [2].

On the other hand, a decentralized improved PoD system is possible using Blockchain. This attractive disruptive technology is known for being immutable and tamper-proof with a decentralized distributed ledger [3]. It has contributed in several solutions, not only in finance [4] but also in health care [5], intrusion detection [6], food industry [7] and supply chain management [3], [8]. Its security features enable transparency, traceability and auditability [9], [10]. In addition to its event system and tamper-proof ordered logs, using Ethereum allows blockchain to execute lines of code, making it programmable [11].Trust plays an important role in a PoD system and is achieved using blockchain as all transactions are transparent and do not require the involvement of a TTP.

In this paper, we propose a blockchain based solution for the proof of delivery of digital assets including videos, photos, eBooks, articles, any digital media etc. The main focus of our solution is to eliminate the need of a trusted third party and to eradicate the trust issue of current PoD systems. The main contributions of this paper are as follows:

- We implement a PoD system for digital media that fulfills the customer needs from requesting the digital content at the beginning till the payment is automatically settled at the end.
- We create a system that incentivizes honesty and reserves the customer's rights in the case of dispute by making a refund and allows cancellation by the customer.
- We use InterPlanetary File System (IPFS) hash [12] in the smart contract to keep the terms and conditions that the enrolling parties must fulfill clear.
- We use unique tokens and utilize the Ethereum Address (EA) of the participating entities to sign the communicated messages off-chain, eliminating the need of a certificate authority and public keys.
- We present the full implementation smart contract code[1] as well as testing details.

H. Hasan is a Masters student at Khalifa Univerisity Department of Electrical and Computer Engineering, UAE, Abu Dhabi, email: haya.hasan@ku.ac.ae

K. Salah is a professor at the Electrical and Computer Engineering Department, Khalifa University, UAE, Abu Dhabi, email:khaled.salah@ku.ac.ae

[1]https://github.com/smartcontract694/PoD_Digital/blob/master/.gitignore/.gitignore

The remainder of this paper is organized as follows. Section II provides the related work. Section III presents the proposed blockchain solution. Section IV has the implementation and testing details. Section V discusses the security analysis of the implemented solution and Section VI concludes the paper.

## II. Related Work

In this section, we will present the existing blockchain based solutions for digital market places. We will be analyzing their PoD systems if any exist.

An online blockchain based digital market place is WEMARK [13]. WEMARK is a solution for photographs explicitly, where photographers can reserve their rights and publish their work to be sold. Interested customers can buy the photos and pay using the Wemark Tokens (WMK) [13]. This blockchain based solution reserves the rights of the content creators. However, it does not propose any PoD scheme. Moreover, it is only limited to photographs and both the content creators as well as customers should trust Wemark as a platform.

AXEL on the other hand, is another blockchain-based digital market place that is more diverse and is also capable of streaming videos and other media files [14]. In spite of it being a system that gives freedom to the content creator to set the royalty share and audience, it does not have any PoD system.

Moreover, All Public Art (APA) is another blockchain digital market place for arts [15]. This startup aims on communicating artists together and reducing intermediary fees. It is specific for artists and their art work. APA is quite limited in its scope of digital media and is mainly targeting reserving the ownership of the artwork. Its solution does not tackle PoD of the artwork sold.

All of the current existing blockchain-based digital market places do not have any PoD techniques or algorithms. They mainly target providing a platform for the creators of the digital media to communicate and sell their items without compromising their share or revenue. The found solutions are extremely limited and do not offer a complete solution for all kinds of digital media. Our proposed blockchain-based PoD solution is transparent, offers automatic payment, refund in case of disputes, eliminates the need of a TTP and is adaptable to any kind of digital media.

## III. Proposed Blockchain Solution

In this section, we present our Ethereum blockchain solution to prove the delivery of different types of digital assets such as digital books, photos, documents as well as media streaming such as videos or music content. The solution utilizes the security features of the blockchain technology by using its transparency and traceability features. This eases the process of ensuring the delivery of the digital content.

### A. System Overview

The proposed blockchain solution focuses on the proof of delivery of digital assets between two parties. All the participating entities possess ethereum addresses and contribute in the communication on the chain with the smart contract. Moreover, an agreement form is signed by all the participating entities at the beginning. Hence, an InterPlanetary File System (IPFS) hash of the terms and conditions agreement is part of the contract created. If all entities agree, the transaction starts and the agreed upon collateral is withdrawn from file server and the customer. The roles of the participants can be summarized as follows:

- **Owner:** The owner refers to the entity that actually owns the digital content and would like to sell it to interested customers.
- **File Server:** The file server is where the digital content is stored and would be streamed or downloaded by the customers.
- **Customer(s):** Multiple customers can have access to the digital content by requesting it through the smart contract.
- **Arbitrator:** The arbitrator is a trusted entity by the owner, file server and customer. The arbitrator would resolve disputes if they occur by trying to download the same item requested by a customer using the customer's token. Based on the download result of the arbitrator a refund may take place.
- **Smart Contract Attestation Authority (SCAA):** SCAA attests the contract to ensure that the code satisfies the agreed upon terms and conditions between all the participating entities. If a contract is attested by the SCAA, the contract address would be part of the SCAAs contract and the SCAAs address would be included in the contract.

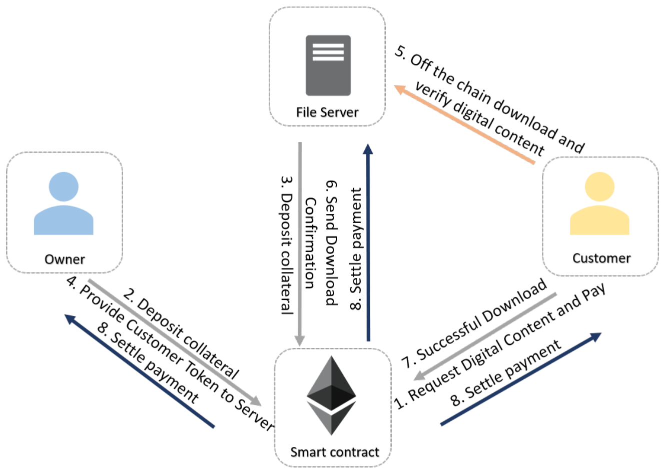


Fig. 1: Main interactions between the different actors and the attested smart contract

If a customer agrees on the terms and conditions then they can request the digital content and this would lead to paying for its price as well as a collateral which is kept as a deposit to incentivize honesty. An equal collateral would then be deposited by the file server. The contract then automatically provides a unique token to the customer. This token would then be used to access the content securely from the server off-chain. Each token has a validity time and the customer should access the content within this duration to avoid a transaction failure. Once the customer downloads the content, the file server would send a confirmation to the contract. The customer

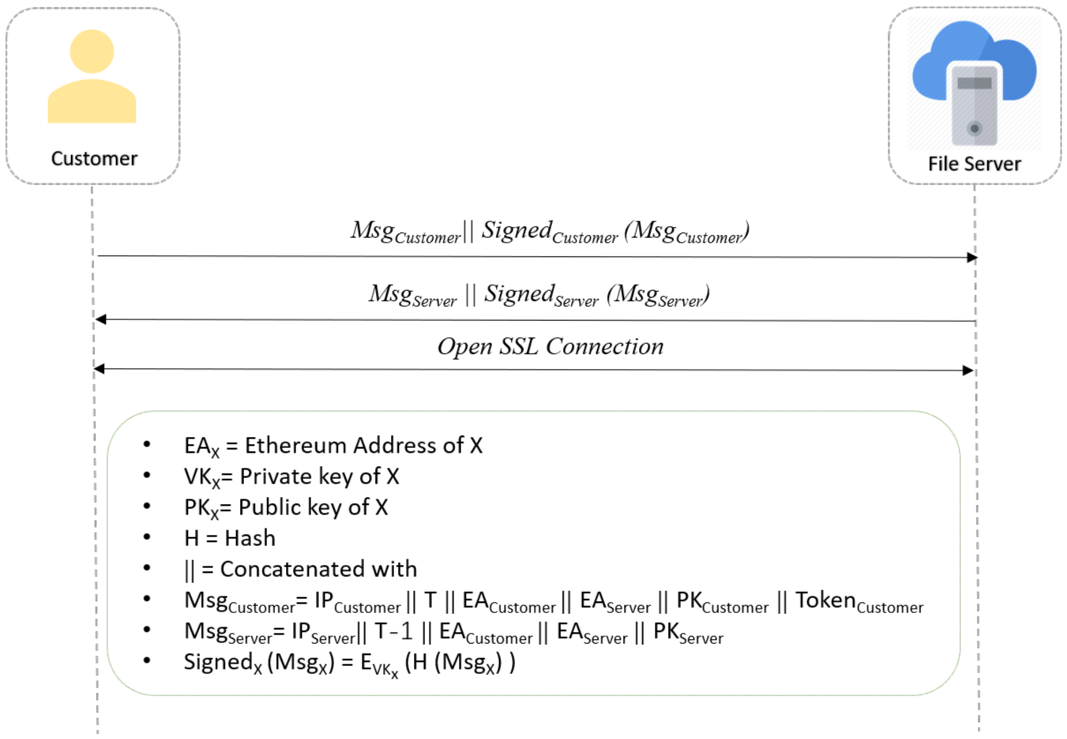

Customer
File Server
$Msg_{Customer}$ || $Signed_{Customer}$ ($Msg_{Customer}$)
$Msg_{Server}$ || $Signed_{Server}$ ($Msg_{Server}$)
Open SSL Connection
• $EA_X$ = Ethereum Address of X
• $VK_X$= Private key of X
• $PK_X$= Public key of X
• H = Hash
• || = Concatenated with
• $Msg_{Customer}$= $IP_{Customer}$ || T || $EA_{Customer}$ || $EA_{Server}$ || $PK_{Customer}$ || $Token_{Customer}$
• $Msg_{Server}$= $IP_{Server}$|| T-1 || $EA_{Customer}$ || $EA_{Server}$ || $PK_{Server}$
• $Signed_X$ ($Msg_X$) = $E_{VK_X}$ (H ($Msg_X$) )

Fig. 2: Off-chain communication between the customer and file server to securely download the digital content.

then sends a successful download confirmation to end the transaction and settle the payment. Everyone's collateral would then be returned and the owner as well as file server are paid their share. Figure 1 shows the interaction of the participating entities with each other and with the smart contract.

Blockchain technology provides transparency and the ability to log everything on the public ledger. Therefore, the Ethereum smart contract creates events and logs that help in tracing and ease the proof of delivery. In order to achieve the needed functionality, the smart contract contains the following:

- **Methods:** Methods are functions in the smart contract. Using functions tasks can be done by the participating entities. There are functions that allow customers to request a document, the arbitrator to resolve disputes, the file server to pay the collateral etc.
- **Modifiers:** Modifiers help to restrict the functionality of a function based on a certain requirement. For instance, a payable function should receive a certain amount of ether to execute, this was used to restrict the deposited collateral to a certain amount. Other modifiers were also used to create a role restriction.
- **Events:** Events are important to notify all the participating entities about updates regarding the transactions taking place. After the execution of any function call, an event takes place. This also eases tracing back in the case of disputes.
- **Variables:** Variables are required to store the information that represents the state of the contract and that might change as the functions get executed.

## B. *Off-Chain Communication*

Off-chain communication is essential between the customers and the file server. The digital media is not stored on the chain as it would be extremely expensive. Hence, a secure download off-chain is the solution. The process starts with the customer requesting the digital content through the smart contract. After the collateral is payed by the customer and the file server, the smart contract automatically generates a unique token for the customer through the function $GenerateToken()$. The token is a hash created using the $keccak256$ built in function of Solidity. The hash is made using the following components, the customer's Ethereum Address ($EA$), Number of Successful Sales ($NoSS$), Number of Customers ($NoC$), Block Time stamp ($BT$), and Token Validity ($\Delta$). The combination of those components ensures that the token is unique for each customer. Therefore, the token is generated as an event to all participating entity as: $Token_{Customer} = keccak256(EA_{Customer}, NoSS, NoC, BT, \Delta)$.

The generated token once created is used by the customer for authentication when communicating with the file server to access the digital content. The message sent between the customer and file server is concatenated with the signature. In addition to the customer's token, the communicated message also includes the customer's Internet Protocol ($IP$) address, Time stamp ($T$), EA of the customer and the file server as well as the public key ($PK_{Customer}$). Hence, the message is sent as: $Msg_{Customer} = IP_{Customer} \mid\mid T \mid\mid EA_{Customer} \mid\mid EA_{Server} \mid\mid PK_{Customer} \mid\mid Token_{Customer}$. This message is sent by the customer to the file server concatenated with its signature. The signature is made using the customer's private key

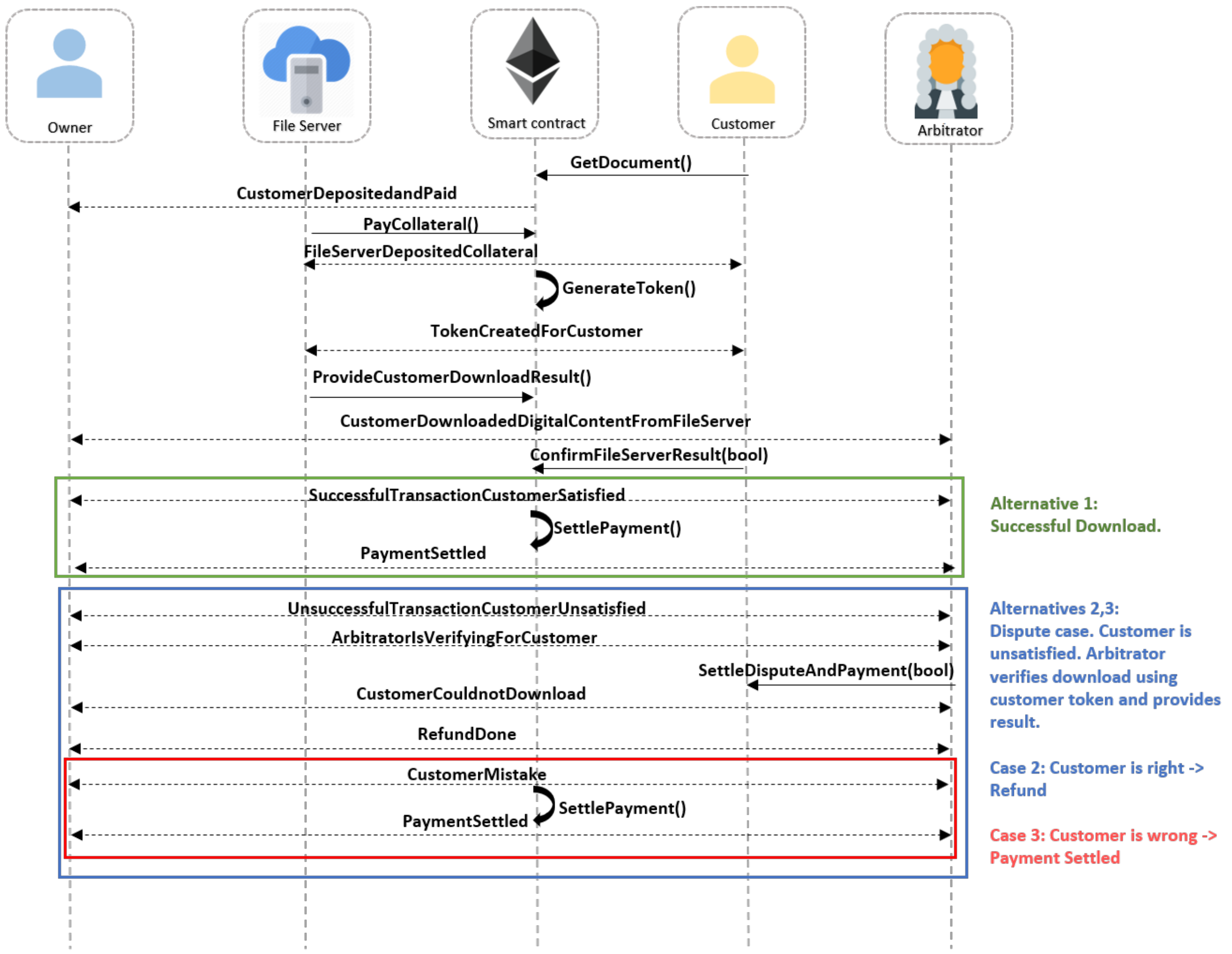

Owner
File Server
Smart contract
Customer
Arbitrator
GetDocument()
CustomerDepositedandPaid
PayCollateral()
FileServerDepositedCollateral
GenerateToken()
TokenCreatedForCustomer
ProvideCustomerDownloadResult()
CustomerDownloadedDigitalContentFromFileServer
ConfirmFileServerResult(bool)
SuccessfulTransactionCustomerSatisfied
SettlePayment()
PaymentSettled
Alternative 1: Successful Download.
UnsuccessfulTransactionCustomerUnsatisfied
ArbitratorIsVerifyingForCustomer
SettleDisputeAndPayment(bool)
CustomerCouldnotDownload
RefundDone
CustomerMistake
SettlePayment()
PaymentSettled
Alternatives 2,3: Dispute case. Customer is unsatisfied. Arbitrator verifies download using customer token and provides result.
Case 2: Customer is right -> Refund
Case 3: Customer is wrong -> Payment Settled

Fig. 3: Sequence diagram showing the function calls and events that take place in three different scenarios based on successful and unsuccessful transactions.

$(VK_{Customer})$ where $Signed_{Customer}(Msg_{Customer}) = E_{VK_{Customer}}(H(Msg_{Customer}))$. The file server upon receiving the message from the customer will verify the signature, the hash and the received message components with the information received through the smart contract.

The file server after authenticating the customer will reply back with a message that contains the server's IP address, time stamp, server's public key as well as the EA of the server and customer. The purpose of this message is to authenticate the file server to the customer, $Msg_{Server} = IP_{Server} \mid\mid T-1 \mid\mid EA_{Customer} \mid\mid EA_{Server} \mid\mid PK_{Server}$. This message is sent along with its signature as well, where the server would sign the message using its private key. Figure 2 presents the off-chain communication between the customer and the file server. After the authentication is completed successfully, an open Secure Sockets Layer (SSL) connection is used for data exchanges and to transmit the digital content from the file server to the customer.

## IV. Implementation and Testing

The contract is created and tested using the Remix IDE. Remix provides an environment for writing smart contracts in Solidity and for testing as well as debugging the contracts. This section focuses on the implementation and testing details.

### A. Implementation Details

The customer first requests the digital content which could be a file, a book, an image, a video or music which can be streamed or downloaded etc. A contract receiving a request by the customer indicates that the customer agreed to the terms and conditions of the contract and has already deposited the double deposit collateral. The file server would then deposit the same amount as collateral. This helps in incentivizing every entity to act honest and to ensure that all participating entities are equal in their authority levels. The contract then automatically creates a unique token for the customer that has a validity depending on the type of the digital content requested. The token is visible to everyone listening in the network. The server would then enable the customer off-chain to securely download the content using the created token.

The file server would then confirm the download by executing a function in the smart contract. This confirmation notifies everyone that the customer has downloaded the content from the file server. The customer would then also execute a function with the download confirmation result. If the customer is satisfied, the payment is settled and the collaterals are refunded to everyone. The owner and the file server would share the item price. However, if the customer is unsatisfied, the arbitrator takes the customer's token and uses it to download the same file. The arbitrator's result would determine if the customer is entitled for a refund or not.

Furthermore, the system also allows the customer to ask for a refund as long as the file server has not yet deposited its collateral. This also helps in returning the funds in case the file server took more than the expected time to respond and to deposit the collateral. The full code[1] is also made available for all the details.

Figure 3 presents the sequence diagram that captures the function calls and events which take place in three different scenarios. The sequence diagram shows the full sequence of events, commencing with the collateral deposit and ending with the payment settlement. Alternative 1 in Figure 3 represents a successful transaction where both the file server and the customer agree that the download was successful. On the other hand, in both of the other alternatives, the file server states that the customer downloaded the content but the customer claims that they did not. Therefore, alternatives 2 and 3 shown in 3 show the dispute cases where the arbitrator steps in. In alternative 2, the arbitrator was also unable to download the content from the server using the customer's token. Hence, a refund takes place. However, in alternative 3, the customer's claim is wrong, therefore, the payment is settled just like in a successful transaction as seen in alternative 1 payment settlement. The rest of this section discusses the details of the important algorithms used in the code.

*1) Requesting the Digital Content:* Algorithm (1) shows the algorithm requesting the digital content by the customer. The customer first reviews the terms and conditions using the IPFS hash available in the smart contract. The customer then makes a request to get the digital content by depositing a collateral as well as the item price.

This process is then followed by the file server depositing an equal collateral. The contract then generates a unique customer token. The token is used off-chain by the customer to securely download the digital content after the server has provided an authorization to the customer.

*2) Successful Download and Payment Settlement:* Once the customer downloads the digital content from the server, the file server executes a function in the contract to notify everyone that the customer has downloaded the content. Algorithm (2) shows the details of a successful download which leads to a payment settlement only if the customer agrees and is satisfied. Consequently, the customer has to reply with a response that confirms their satisfaction. If the customer is satisfied, the payment is settled and the transaction is completed successfully. The file server and the customer would all get back their deposited collateral. Moreover, the owner and file server would get their share of the payment.

*3) Unsuccessful Download and Dispute:* If the customer's download confirmation is negative and they do not agree with the file server, this leads to the customer being unsatisfied.

**Algorithm 1:** Requesting Digital Content

```
Input : $E$, collateral, item price, contract state,
        customer state
1  $E$ is the set of all static ethereum addresses saved in this
     contract.
2  Restrict access to any customer $e \notin E$
3  if $msg.value = item\ price + collateral$ then
4      if $contract\ state = waiting\ for\ customer$ then
5          Withdraw the msg.value ether from $e$.
6          Change state of the customer to next state.
7          Create a notification about the successful
             withdrawal.
8      end
9      else
10         Revert contract state and show an error.
11     end
12 end
13 else
14     Revert contract state and show an error.
15 end
```

Hence, only in this case the arbitrator is requested to step in. Algorithm (3) shows the full details of how the dispute is handled when the customer is unsatisfied. The arbitrator would try to download from the file server the same digital content using the same customer token. If the arbitrator was able to download the content, then the customer's claim is falsified and the payment is settled just like in a successful transaction. All collaterals are returned and the owner as well as the file server are payed according to their share. However, if the arbitrator also faces a problem and is unable to download the content, the customer is refunded and all collaterals are returned.

## B. Testing and Validation

This section describes the details of testing the smart contract code. Four main functionalities are tested which include a successful collateral deposit after a customer requests a document, unique token generation, a successful payment settlement and a refund based on dispute handling by the arbitrator. The testing also involves multiple customers to show that the code is designed to handle multiple requests by different customers from different Ethereum addresses. The Ethereum addresses of the owner, file server and arbitrator are "0xca35b7d915458ef540ade6068dfe2f44e8fa733c", "0x14723a09acff6d2a60dcdf7aa4aff308fddc160c" and "0xdd870fa1b7c4700f2bd7f44238821c26f7392148" respectively. All functions can only be executed by certain entities with specific Ethereum addresses unless they are made for customers with any Ethereum address. Therefore, all restricted functions with modifiers have been tested successfully and they would revert back to the old state if an unauthorized caller tries to execute them.

*1) Document Request and Collateral Deposits:* A customer requests a document using a payable function. Therefore, the collateral which in this case was twice the item price

**Algorithm 2:** Successful Download and Payment Settlement

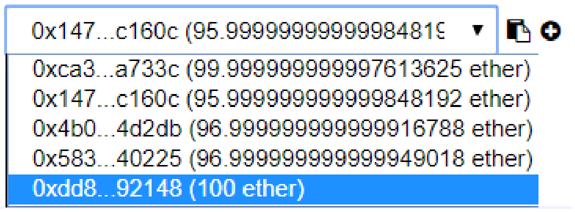
```
Input : $E$, caller, file server address, contract state,
        customer state, customer address
$E$ is the set of all ethereum addresses with collateral
  deposits.
$FS \leftarrow file\ server\ address.$
$CA \leftarrow customer\ address.$
if $caller = FS \wedge (state \in CA) =$
  $provided\ hash\ and\ token$ then
    Change state of $CA$ to Download confirmed by
      Server.
    Create a notification with server download result.
    Wait for confirmation from $CA$.
    if $CA$ confirmation result = true then
        foreach $e \in E$ do
            Return back the collateral.
        end
        Pay the owner and file server their share.
    end
    else
        Execute the Settle Dispute Algorithm (Algorithm
          3).
    end
end
else
    Revert contract state and show an error.
end
```

**Algorithm 3:** Unsuccessful Download and Dispute Handling

```
Input : $E$, arbitrator, caller, customer address
$CA \leftarrow customer\ address.$
$X \leftarrow arbitrator\ address.$
$E$ set of all Ethereum addresses with collateral deposits.
if $caller = CA \wedge (state \in CA) = Unsatisfied$ then
    Create a notification to $X$.
    $X$ uses token $\in CA$ to download digital content.
    $x \leftarrow X\ download\ result.$
    if $x$ = true then
        Create a notification that $CA$ is right.
        Refund Customer $CA$.
    end
    else
        Create a notification that $CA$ is wrong.
        foreach $e \in E$ do
            Refund the collateral.
        end
        Pay the owner and server their share.
    end
    Change state of the customer to Transaction
      Completed.
end
else
    Revert contract state and show an error.
end
```

will be withdrawn from the customer's account along with the payment. In this testing scenario, there are two customers who requested the digital content with Ethereum addresses "0x4b0897b0513fdc7c541b6d9d7e929c4e5364d2db" and "0x583031d1113ad414f02576bd6afabfb302140225". The item price here is 1 Ether, hence, the collateral is 2 Ether. After the customers have successfully made their payment along with their request, the file server also payed the collateral of 2 Ether and a token was generated for each customer.

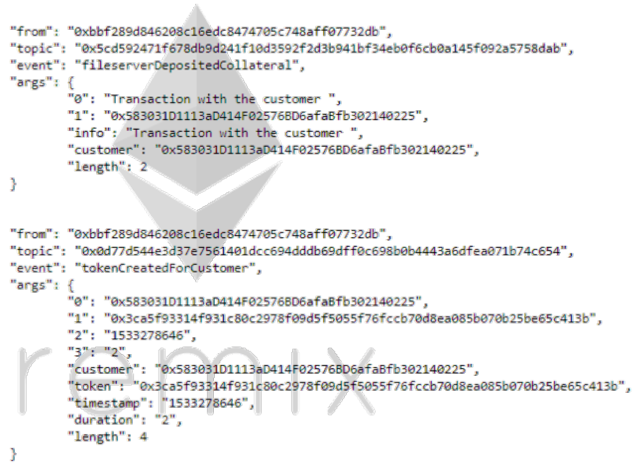
Figure 4 shows the balance of the customers as almost 97 Ether. This indicates that they have successfully requested the digital content. Also, the file server's balance is almost 96 Ether which indicates that it has deposited a collateral for two customers successfully.



```
0x147...c160c (95.9999999999998481$
0xca3...a733c (99.99999999997613625 ether)
0x147...c160c (95.99999999999848192 ether)
0x4b0...4d2db (96.99999999999916788 ether)
0x583...40225 (96.99999999999949018 ether)
0xdd8...92148 (100 ether)
```

Fig. 4: Ether balance of all the participating entities in the smart contract

*2) Token Generation:* A unique token is automatically generated by the smart contract for each customer after the file server pays the collateral. This is done using an internal function that is called by the smart contract. The token is generated using the keccak256 hash of the customer address, number of customers, number of successful sales, validity and block time stamp. Figure 5 shows the event logs that have the unique token generated for the customer with Ethereum address "0x583031d1113ad414f02576bd6afabfb302140225".

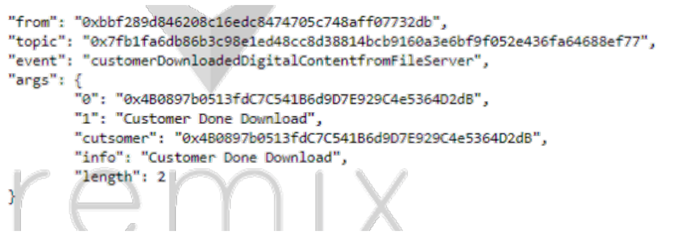


```
"from": "0xbbf289d846208c16edc8474705c748aff07732db",
"topic": "0x5cd592471f678db9d241f10d3592f2d3b941bf34eb0f6cb0a145f092a5758dab",
"event": "fileserverDepositedCollateral",
"args": {
        "0": "Transaction with the customer ",
        "1": "0x583031D1113aD414F02576BD6afaBfb302140225",
        "info": "Transaction with the customer ",
        "customer": "0x583031D1113aD414F02576BD6afaBfb302140225",
        "length": 2
}


"from": "0xbbf289d846208c16edc8474705c748aff07732db",
"topic": "0x0d77d544e3d37e7561401dcc694dddb69dff0c698b0b4443a6dfea071b74c654",
"event": "tokenCreatedForCustomer",
"args": {
        "0": "0x583031D1113aD414F02576BD6afaBfb302140225",
        "1": "0x3ca5f93314f931c80c2978f09d5f5055f76fccb70d8ea085b070b25be65c413b",
        "2": "1533278646",
        "3": "2",
        "customer": "0x583031D1113aD414F02576BD6afaBfb302140225",
        "token": "0x3ca5f93314f931c80c2978f09d5f5055f76fccb70d8ea085b070b25be65c413b",
        "timestamp": "1533278646",
        "duration": "2",
        "length": 4
}
```

Fig. 5: Logs showing a unique token generated by the smart contract

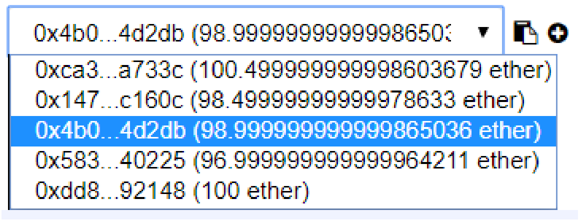
*3) Successful Transaction and Payment Settlement:* A successful transaction means that the file server indicated that the customer has downloaded the content and the customer has agreed and is satisfied. Figure 6 show the logs after an event was triggered as the server indicated that customer with Ethereum address ”0x4b0897b0513fdc7c541b6d9d7e929c4e5364d2db” has downloaded the content from the file server.



```
"from": "0xbbf289d846208c16edc8474705c748aff07732db",
"topic": "0x7fb1fa6db86b3c98e1ed48cc8d38814bcb9160a3e6bf9f052e436fa64688ef77",
"event": "customerDownloadedDigitalContentfromFileServer",
"args": {
        "0": "0x4B0897b0513fdC7C541B6d9D7E929C4e5364D2dB",
        "1": "Customer Done Download",
        "cutsomer": "0x4B0897b0513fdC7C541B6d9D7E929C4e5364D2dB",
        "info": "Customer Done Download",
        "length": 2
}
```

Fig. 6: Logs showing that the customer has downloaded the content from the file server

Then the customer would provide their confirmation. A successful transaction means the customer is satisfied and agrees with the file server that the download was successful. Consequently, the payment is automatically settled and the collateral is returned to the customer and the file server as shown in Figure 7. Moreover, the owner and the file server share the payed price equally. Figure 7 presents the Ether balance of the customer with Ethereum address ”0x4b0897b0513fdc7c541b6d9d7e929c4e5364d2db” increased by 2 Ether as they received back their collateral. The owner received 0.5 Ether and the file server received 2.5 Ether.



```
0x4b0...4d2db (98.9999999999998650
0xca3...a733c (100.499999999998603679 ether)
0x147...c160c (98.49999999999978633 ether)
0x4b0...4d2db (98.99999999999865036 ether)
0x583...40225 (96.99999999999964211 ether)
0xdd8...92148 (100 ether)
```

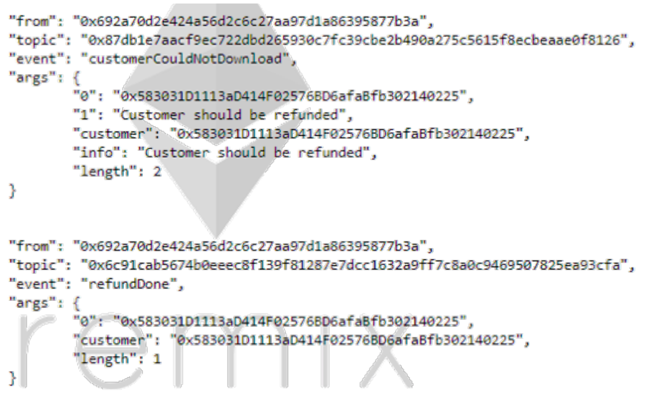
Fig. 7: Ether balance after payment settlement

*4) Unsuccessful Transaction and Dispute Handling:* If the customer disagrees with the file server indicating that the customer has downloaded the content successfully, this leads to a dispute and to the arbitrator stepping in. The arbitrator would use the customer’s token off-chain to download the same content from the file server. If the arbitrator is able to download the content, the payment is settled as mentioned previously just like in a successful transaction. Otherwise, a refund occurs. Figure 8 show the logs after the arbitrator used the token of the unsatisfied customer with the Ethereum address ”0x583031d1113ad414f02576bd6afabfb302140225”. The arbitrator could not also download the content. Hence, a refund occurs automatically after the arbitrator provides the result.

Figure 9 provide the Ether balance of the customer with address ”0x583031d1113ad414f02576bd6afabfb302140225” refunded back 3 Ether. Also, the file server is refunded back the collateral. This shows a successful refund after the dispute was handled by the arbitrator. The arbitrator is only involved in the case of dispute.

## V. Security Analysis

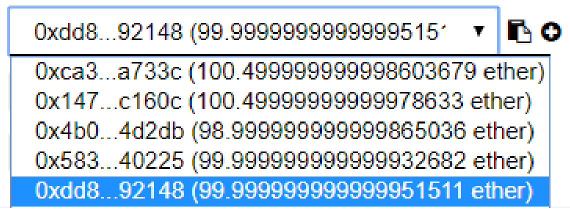
This section provides security analysis of the proposed system. The section also provides vulnerability analysis for the smart contract code.



```
"from": "0x692a70d2e424a56d2c6c27aa97d1a86395877b3a",
"topic": "0x87db1e7aacf9ec722dbd265930c7fc39cbe2b490a275c5615f8ecbeaae0f8126",
"event": "customerCouldNotDownload",
"args": {
        "0": "0x583031D1113aD414F02576BD6afaBfb302140225",
        "1": "Customer should be refunded",
        "customer": "0x583031D1113aD414F02576BD6afaBfb302140225",
        "info": "Customer should be refunded",
        "length": 2
}

"from": "0x692a70d2e424a56d2c6c27aa97d1a86395877b3a",
"topic": "0x6c91cab5674b0eeec8f139f81287e7dcc1632a9ff7c8a0c9469507825ea93cfa",
"event": "refundDone",
"args": {
        "0": "0x583031D1113aD414F02576BD6afaBfb302140225",
        "customer": "0x583031D1113aD414F02576BD6afaBfb302140225",
        "length": 1
}
```

Fig. 8: Logs that show the arbitrator result after trying to download the digital content off-chain

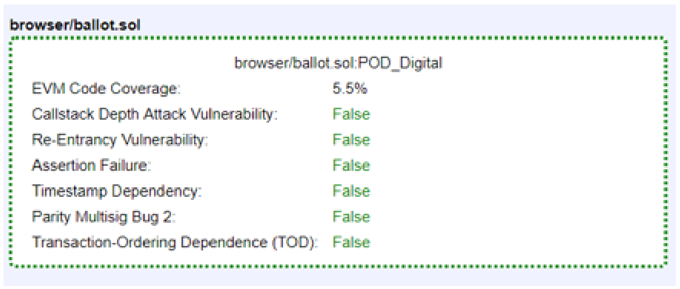


```
0xdd8...92148 (99.99999999999995151
0xca3...a733c (100.499999999998603679 ether)
0x147...c160c (100.49999999999978633 ether)
0x4b0...4d2db (98.99999999999865036 ether)
0x583...40225 (99.99999999999932682 ether)
0xdd8...92148 (99.99999999999951511 ether)
```

Fig. 9: Ether balance of participants after dispute handling that resulted in a refund

### A. Vulnerability Analysis

Oyente is an open source tool that is used for smart contract code analysis against known bugs and vulnerabilities. It is necessary to eliminate security holes that allow an adversary to manipulate with the execution of a smart contract. Oyente works on the byte code and not on the high level language such as Solidity [16]. The tool provides to the user an analysis report that verifies the availability of certain important security bugs such as Transaction Ordering Dependence (TOD), timestamp-dependence, mishandled exceptions or re-entrancy detection [16]. Figure 10 shows the Oyente vulnerability analysis report on our smart contract code. None of the vulnerabilities that the tool checks for is ”True”. The results show that the code does not suffer from security bugs, hence, the result shows ”False” in green.



browser/ballot.sol

browser/ballot.sol:POD_Digital

| | |
|---|---|
| EVM Code Coverage: | 5.5% |
| Callstack Depth Attack Vulnerability: | False |
| Re-Entrancy Vulnerability: | False |
| Assertion Failure: | False |
| Timestamp Dependency: | False |
| Parity Multisig Bug 2: | False |
| Transaction-Ordering Dependence (TOD): | False |

Fig. 10: Security vulnerability report by Oyente tool

### B. Security Features

This section describes the important security features of the proposed solution such as confidentiality, integrity, non-repudiation,authentication and authorization security requirements. Those requirements ensure that the system is protected against man in the middle attacks as well as replay attacks.

- **Confidentiality:** Confidentiality ensures that the communication made between any two parties cannot be interpreted by anyone else. Therefore, unauthorized access to the data cannot take place. Confidentiality can be achieved through encryption. It is necessary to keep the off-chain communication between the customers and the file server confidential to ensure that only the authorized customers have access to the digital content. Therefore, in our solution, message encryption and decryption is achieved through a secure SSL session after a successful handshake that ensures the authentication of the customer and file sever. Using blockchain in our solution exempts us from the burden of using the Public Key Infrastructure (PKI) for encryption. The current PKI system is centralized and therefore, lacks transparency and depends on trusting certificate authorities in the distribution of keys [17]. The unique Ethereum Address that each participating entity owns comes with asymmetric public key pairs which are used for encrypting the communicated messages through the SSL session.
- **Integrity:** All exchanged messages between the participating entities on the Ethereum blockchain are all tamper proof and cannot be altered. Moreover, the off-chain communication between the customers and the file server is secured with time stamps and a unique customer token that is already communicated on the chain. Therefore, the communication is secured against replay and Man In The Middle (MITM) attacks.
- **Non-repudiation:** The communication on the Ethereum blockchain and the events that take place are all part of the logs, where the initiator of the calls is recorded and cannot be manipulated with. Hence, no one can deny their own actions as it is already all recorded in tamper proof logs. Moreover, concerning the off-chain communication, if an intruder tries to mimic a customer's EA or token, they will be discovered as they cannot know the right private key to sign the message.
- **Authentication and Authorization:** All functions of the smart contract can only be accessed by certain authorized entities. If the initiator of the call is identified as unauthorized an error occurs and all states are reverted. Furthermore, off-chain communication between the customers and the file server first depends on a successful handshake for authentication, which yields to a secure SSL connection to download the content.

## VI. Conclusion

In this paper, we have presented a blockchain-based solution for the PoD of digital assets. Our solution provides tamper proof logs that simplify tracing events and tasks. The solution uses a unique token for each customer to access the content from the server. Off-chain communication is encrypted by using the key pairs of the EA. The solution eliminates the need of a TTP, uses double collateral as an honesty incentive, and utilizes the contract as an escrow. We can remark that our proposed solution is completely decentralized, for the most part, with automated payment. Arbitrator is only involved in the case of a dispute. We showed that our solution is secure against popular attacks such as replay and MITM attacks. And we also showed that our smart contract code is free of known vulnerabilities and bugs.

## References

[1] Digital media revenue in selected countries worldwide in 2018 (in billion u.s. dollars). [Online]. Available: https://www.statista.com/statistics/459335/digital-media-revenue-countries-digital-market-outlook/

[2] H. Hasan and K. Salah, "Blockchain-based Solution for Proof of Delivery of Physical Assets," *International Conference on Blockchain (ICBC), Seattle, USA*, June 2018.

[3] K. Toyoda, P. T. Mathiopoulos, I. Sasase, and T. Ohtsuki, "A novel blockchain-based product ownership management system (poms) for anti-counterfeits in the post supply chain," *IEEE Access*, vol. 5, pp. 17 465–17 477, 2017.

[4] P. Treleaven, R. G. Brown, and D. Yang, "Blockchain technology in finance," *Computer*, no. 9, pp. 14–17, 2017.

[5] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "Medrec: Using blockchain for medical data access and permission management," in *Open and Big Data (OBD), International Conference on*. IEEE, 2016, pp. 25–30.

[6] W. Meng, E. W. Tischhauser, Q. Wang, Y. Wang, and J. Han, "When intrusion detection meets blockchain technology: a review," *Ieee Access*, vol. 6, pp. 10 179–10 188, 2018.

[7] F. Tian, "A supply chain traceability system for food safety based on haccp, blockchain & internet of things," in *Service Systems and Service Management (ICSSSM), 2017 International Conference on*. IEEE, 2017, pp. 1–6.

[8] R. AlTawy, M. ElSheikh, A. M. Youssef, and G. Gong, "Lelantos: A blockchain-based anonymous physical delivery system," Cryptology ePrint Archive, Report 2017/465, 2017. http://eprint. iacr. org/2017/465, Tech. Rep.

[9] S. Singh and N. Singh, "Blockchain: Future of financial and cyber security," in *2016 2nd International Conference on Contemporary Computing and Informatics (IC3I)*, Dec 2016, pp. 463–467.

[10] K. Biswas and V. Muthukkumarasamy, "Securing smart cities using blockchain technology," in *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, Dec 2016, pp. 1392–1393.

[11] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[12] Ipfs is the distributed web. [Online]. Available: https://ipfs.io/

[13] License photos directly from top photographers. [Online]. Available: https://www.wemark.com/

[14] Axel. [Online]. Available: https://axeltoken.com/downloads/AXEL-White-Paper-En.pdf

[15] Making history in the global art market. [Online]. Available: https://allpublicart.io/

[16] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 254–269. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978309

[17] M. Al-Bassam, "Scpki: A smart contract-based pki and identity system," in *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, ser. BCC '17. New York, NY, USA: ACM, 2017, pp. 35–40. [Online]. Available: http://doi.acm.org/10.1145/3055518.3055530