Trustworthy IoT Data Streaming using Blockchain and IPFS

Article *in* IEEE Access · January 2022

DOI: 10.1109/ACCESS.2022.3149312

CITATIONS
4

READS
213

6 authors, including:

Haya Hasan
Khalifa University
34 PUBLICATIONS **1,085** CITATIONS
SEE PROFILE

Khaled Salah
Khalifa University
357 PUBLICATIONS **10,045** CITATIONS
SEE PROFILE

Raja Jayaraman
Khalifa University
153 PUBLICATIONS **2,565** CITATIONS
SEE PROFILE

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3149312, IEEE Access

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Digital Object Identifier

Trustworthy IoT Data Streaming using Blockchain and IPFS

HAYA R. HASAN[1], KHALED SALAH[1], IBRAR YAQOOB[1], RAJA JAYARAMAN[2], SASA PESIC[3], MOHAMMED OMAR[2]

[1]Department of Electrical Engineering and Computer Science, Khalifa University of Science and Technology, Abu Dhabi, UAE.
[2]Department of Industrial & Systems Engineering, Khalifa University of Science and Technology, Abu Dhabi, UAE.
[3]ASU's Blockchain Research Laboratory, Arizona State University, Tempe, AZ, USA.

ABSTRACT

Today's resource-constrained IoT streaming devices generate large amounts of data which is stored, processed, analyzed for value creation, and accessed using centralized systems, technologies, platforms, and services. Most existing systems leveraged for storing and accessing IoT streaming data fall short in providing transparency, traceability, reliability, trustworthiness, and security features. Also, they are vulnerable to the single point of failure problem due to centralization. In this paper, we propose a blockchain-based solution for resource-constrained IoT streaming devices that allows data chunks to be transferred in a decentralized, transparent, traceable, reliable, secure, and trustful manner. We preserve the privacy and confidentiality of the IoT streamed data through a proxy re-encryption network. We use the decentralized storage of the Interplanetary File System (IPFS) to store and share the IoT streaming data, thereby dealing with the large-size data storage problem. We present system diagrams and eleven algorithms along with their full implementation details. We perform security analysis to show our smart contract code is secure enough against well-known security threats and vulnerabilities. We compare our proposed approach with the existing solutions to show its novelty and effectiveness. We make our smart contract code publicly available on the GitHub repository.

INDEX TERMS IoT; Data Streaming; IPFS; Blockchain; Ethereum; Smart Contracts; Trust; Security; Proxy Re-encryption

I. INTRODUCTION

The Internet of things (IoT) industry is witnessing an evolution with the emergence and development of fast pacing technology. The IoT enables physical devices, automobiles, equipment, and even buildings to be embedded with sensors and to interact with each other and interchange data [1]. Business models are adopting initiatives driven by the IoT industry. The IoT data streams are considered as valuable assets that are processed and managed for their great value and even sold in marketplaces. Querying and filtering IoT streams as well as providing access rights are some of the complex tasks for the power-constrained IoT devices. The massive volumes of data collected create a set of challenges for the limited storage, power, and networking devices. Therefore, the streaming devices acquire data from their surroundings and send this data to edge nodes or gateways for further processing. In order to make value from the data generated by the IoT streaming devices, the devices are typically integrated with the cloud or centralized servers for data analysis. This opens a door towards the new paradigm of integrating the IoT with the cloud, creating the Cloud of things (CoTs) [2], [3].

Although cloud computing has solved the limitations of storage and processing power that the IoT streaming devices lack, centralization is now inevitable [4]. Due to the single point of failure problem, centralization can compromise the security of the access control system. Furthermore, centralization is not the only challenge. Another important aspect that seeks attention is finding a secure and trusted way for data streams. Streams of data pose a challenge by themselves because unlike a regular file, they are continuous, their freshness and usage are time-dependent, and they must be securely accessible by the interested parties while needed. A regular file can be stored securely all together, and accessed when needed. However, streaming relies on continuous bits of data that need to be carefully aligned together to generate the same meaning as when they were originally generated.

We propose a decentralized solution that is secure, transparent, and has very high integrity and immutability to en-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3149312, IEEE Access

force trust for accessing data streams. Our solution is based on the blockchain technology [5]. It leverages the transparency, immutable logs, and data integrity of the distributed ledger to provide a way that allows IoT data streams to be accessed securely by multiple participants [6]. The data provenance and signed transactions offered by the blockchain technology ensure accountability and non-repudiation [7]. Moreover, we integrate our solution with off-chain decentralized storage to facilitate storing the IoT data chunks securely and allowing multiple access to the information when needed. Furthermore, our solution incorporates the use of Ethereum smart contracts that hold the programmable logic of the system [8]. Each function execution generates a transaction stored as part of the tamper-proof logs. We limit the storage on-chain to the data chunk hashes only, as well as the streamed data time-stamped indices and chunk numbers. This was done to ensure that the hash indices and chunk numbers are immutable, easy to track and to avoid the high cost of on-chain storage. Our proposed system exploits the unique characteristics of the decentralized blockchain technology, offers privacy for use cases that require data confidentiality, uses off-chain decentralized storage, and enforces trust amongst the system participants.

A. RELATED WORKS AND CONTRIBUTIONS

The authors in [9] developed a system called Bolt which specializes in efficiently storing and querying data from home-connected devices. Bolt is used for data streaming. It needs to compress and encrypt the data chunks before storing them. It also uses time-based tags for ease of lookups. Bolt is not a blockchain-based system. Although Bolt may have paved the way with its design concept to querying and managing IoT data, it is a centralized system that can work for a small scale use-case.

The authors in [10] used the Ethereum blockchain network to build a platform for sharing weather sensor data in a market place. Their application relies on the sensing as a service business model. They use their own custom token for buying and selling the IoT data. They use two smart contracts and store the information for each IoT weather sensor, such as the owner, time, and frequency. In their implementation, they stored each event in an SQL-relational like database called Maria Database (DB). The Maria DB is an exact copy of the Ethereum logs. The owner of the weather sensor's data decides on the place the data is stored and shares the fixed URL with the weather sensor's data when registering the sensor. The platform is designed for weather sensors, and the weather data can be stored on a centralized server. The use of the DB can be avoided.

The study conducted in [11] focuses on access control and permissions for IoT-based systems on blockchain. Its main goal is to eliminate centralized access management. The authors recognize IoT devices as constrained in nature, hence, the devices cannot communicate directly with the blockchain. A management hub is needed to facilitate communication between a group of IoT sensor devices and the blockchain. All the allowed operations are part of a single smart contract. Since IoT devices do not belong to the blockchain, each of them has a public key generated by the system. The paper presents a proof of concept that focuses on removing IoT devices from the Ethereum blockchain network. Therefore, a lot of work was involved in building a management hub and creating keys for the devices.

The authors in [12] presented a market place for brokered IoT data using blockchain technology. Their model opts for a decentralized market place for the brokered data. They have a producer and a consumer, and the smart contracts act as a negotiation and a settlement smart contract. A reputation is associated with the users. The model offers a market place where offers are denied or agreed upon using the smart contracts. It also focuses on assessing the trust model between the producer and the consumer. In [13], the researchers provided a generic blockchain-based solution for access control and off-chain data sharing. All access logs are stored on the chain. The paper focuses on identity management and access control of data. It also proposes several applications, such as a market place. However, the solution does not accommodate IoT data streams.

The authors in [14] developed a distributed access control and management system for IoT devices using blockchain. They use Bitcoin and rely on virtual chains on top of the blockchain. The blockchain is utilized to provide access control. They chunk the data streams and use key regression for encryption. They propose using stealth addresses to preserve the privacy of access permissions. They focus on data compression, encryption, and storage. It could have been more valuable if the authors had included details of the algorithms with their proposed solution.

Similarly, [15] discussed IoT communication using a consortium blockchain and side chains. A side chain in the paper is referred to as a private blockhain for a group of IoT devices. The smart contract must store all the addresses of the IoT devices in order to ensure they are legitimate and to meet their requests. A requester should also join the consortium blockchain to be able to access the hash of the data that is stored on the InterPlanetary File System (IPFS). They tested their architecture on Ethereum and Monax.

The aforementioned solutions all agree on the importance of managing IoT data, and several of the studies mentioned have proposed the use of blockchain for IoT data access and management. Each proposed system has its own way of implementation. However, not all the solutions accommodate data streams as part of their solution. Our solution is a fully decentralized blockchain-based system for IoT streamed data access management and control. It leverages the blockchain's intrinsic features to enforce trust, transparency, accountability, non-repudiation, authentication, and data integrity. It also provides privacy and confidentiality as per the use case by using a proxy re-encryption network.

The main contributions of this paper can be summarized as follows:

- We propose a blockchain-based solution that ensures

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3149312, IEEE Access

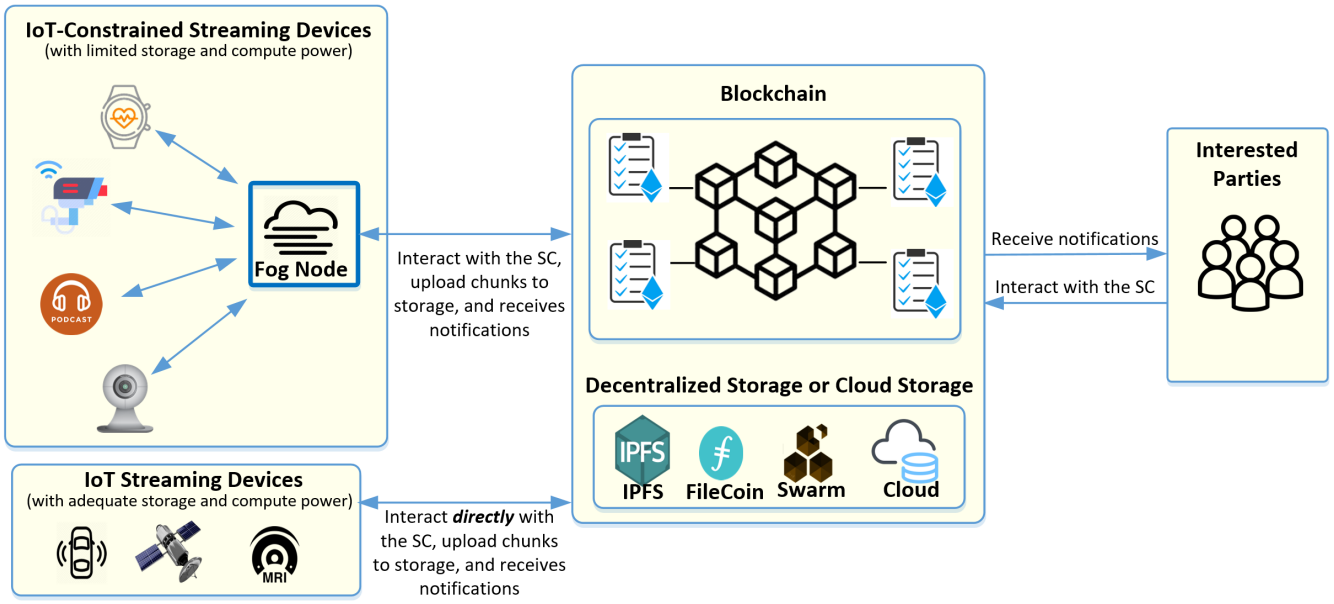

FIGURE 1: System diagram of the IoT streaming devices blockchain-based solution

IoT streamed data access management and acquisition in a manner that is decentralized, transparent, traceable, reliable, auditable, secure, and trustworthy. We integrate our Ethereum blockchain-based solution with the decentralized storage of the IPFS to deal with the large-scale data problem.

- We develop smart contracts that can be used per IoT streaming device for transferring the streamed data chunk hash, chunk number, and timestamped indices on-chain. Also, we write and test a code for bidding in an auction that aims to sell the streaming device when needed.
- We present algorithms for updating the sampling period and IoT device configurations, advertising the streaming device for sale, placing bids by registered participants, processing the payment, and changing the ownership of the streaming device. We provided the smart contract code for these algorithms and made it publicly available on GitHub[1].
- We evaluate the proposed approach to show how our solution can be used in such applications that require privacy and confidentiality of the IoT streamed data. We present the details of encrypting the streamed data using a proxy re-encryption network. Furthermore, we show how our solution allows multiple accesses to the encrypted streamed data chunk file while we only write to the data chunk file once. We analyze our smart contract code against security threats and vulnerabilities using the Oyente tool and compare our solution with the existing solutions.

[1]https://github.com/smartcontract694/IoT/blob/main/Code

The rest of the paper is organized as follows. Section II presents the design details of the proposed blockchain-based solution, followed by the implementation details in Section III including the algorithms. Section IV presents the testing details followed by section V which showcases the security analysis, comparison with the existing solutions, and open challenges. Section VI concludes the paper.

II. SYSTEM DESIGN

This section explains the design of our blockchain-based solution for IoT streaming devices. Figure 1 shows the specific components of our blockchain-based system. In our design, the streaming devices interact with smart contracts. The participants can also listen to events, or interact with the smart contracts. In our solution, both decentralized storage and cloud storage can be used. The rest of this section explains each component of our system in detail.

A. IOT STREAMING DEVICES

IoT streaming devices are the main components that communicate with the blockchain. The streaming devices chunk data streams into smaller parts at regular intervals to form data chunks. The period is set based on the device and the use case. The data chunks are then stored off-chain, and only their tracking information and hashes are stored on-chain. Those devices are classified into two categories. First, limited and constrained IoT devices with limited computing, networking, and storage capabilities. Secondly, highly powered IoT streaming devices with adequate power, networking, and storage capabilities. Figure 1 shows examples of both kinds and their ways of communicating with the smart contracts and blockchain. As can be observed, powerful devices can do it directly, and they do not need any

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3149312, IEEE Access

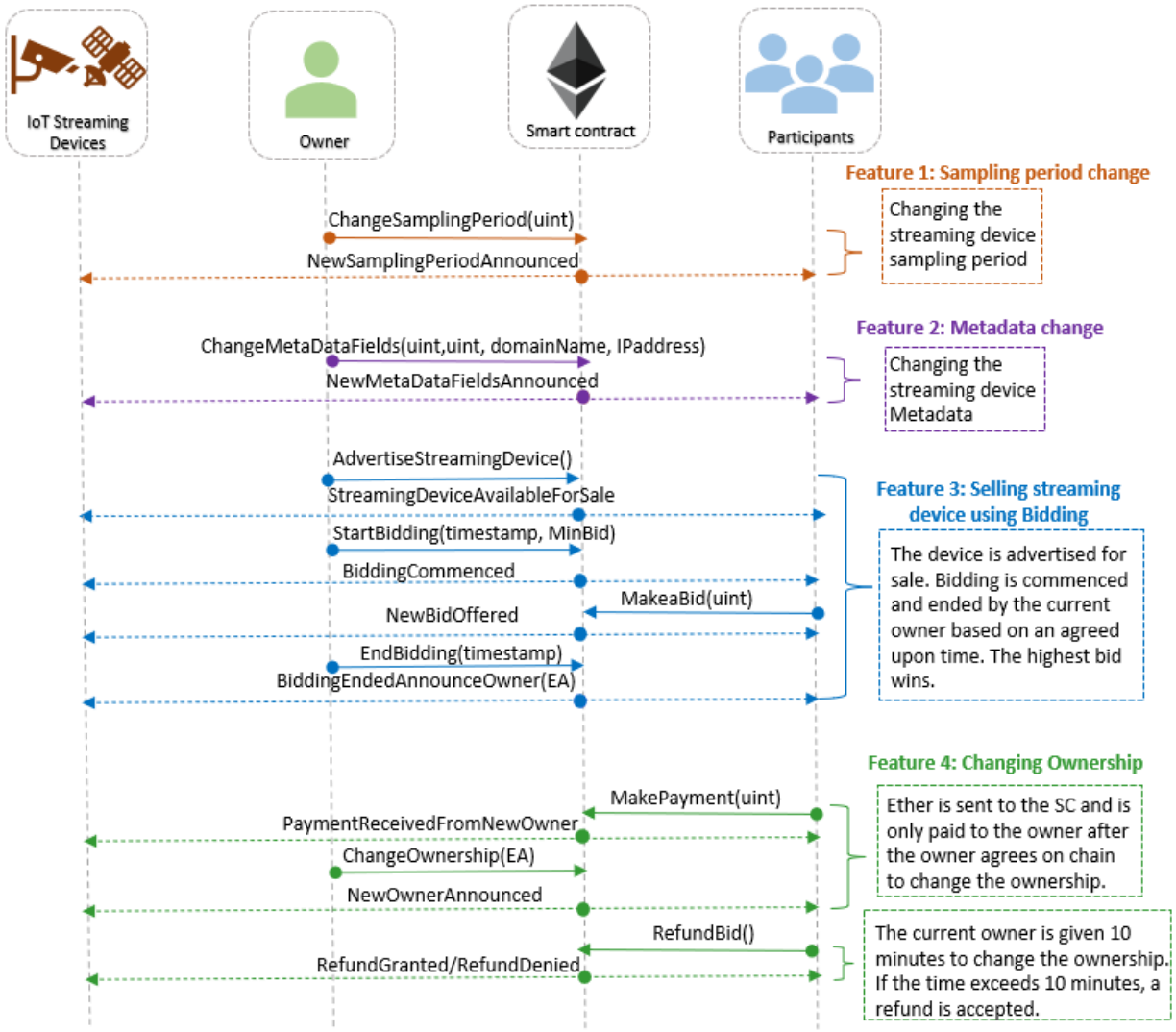

FIGURE 2: IoT Streaming Device Smart Contract sequence diagram

external devices or nodes to facilitate such interaction. They can also communicate with the storage components directly. However, the other constrained IoT devices rely on edge computing to communicate with the blockchain. Using the fog node, they can bridge the gap between their limited capabilities and the programmable logic of the blockchain smart contracts. A limited fog node can also communicate with a blockchain gateway, such as Infura to communicate with the smart contracts. The edge computing fog nodes also transfer the data to off-chain storage. The role of the IoT streaming devices is mainly to keep sending data chunks regularly based on the sampling period and rate. The IoT data streams are divided into chunks and indexed as time-series data. The data chunks are first stored off-chain and their hash is then logged on-chain along with the timestamped index. The hashes can be cross-checked when downloading the data chunks from the off-chain storage and the chunk numbers as well as indices can be used to keep track of the order and ensure the continuity of the data when generated by the device originally. Therefore, this enforces transparency, trust, and integrity of the streamed data.

B. BLOCKCHAIN

Blockchain is a core component of our solution. It has mining nodes and programmable logic. The smart contracts that are created for each streaming device carry the programmable logic and bound all interested parties and participants to fulfill their roles. A public Ethereum blockchain network is used in our solution to capture all the transactions between the different users. Blockchain is a shared ledger that keeps provenance data for tracking data chunk transfers, streamed data hashes, and time-dependent indices. Every function execution is a logged transaction. Using blockchain, each transaction is logged and duplicated across all the nodes in a distributed manner. It also offers accountability as every function executed by the owner, device, or any other participant cannot be denied. Blockchain also provides the ability to sell the IoT streaming device through an on-chain auction

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3149312, IEEE Access

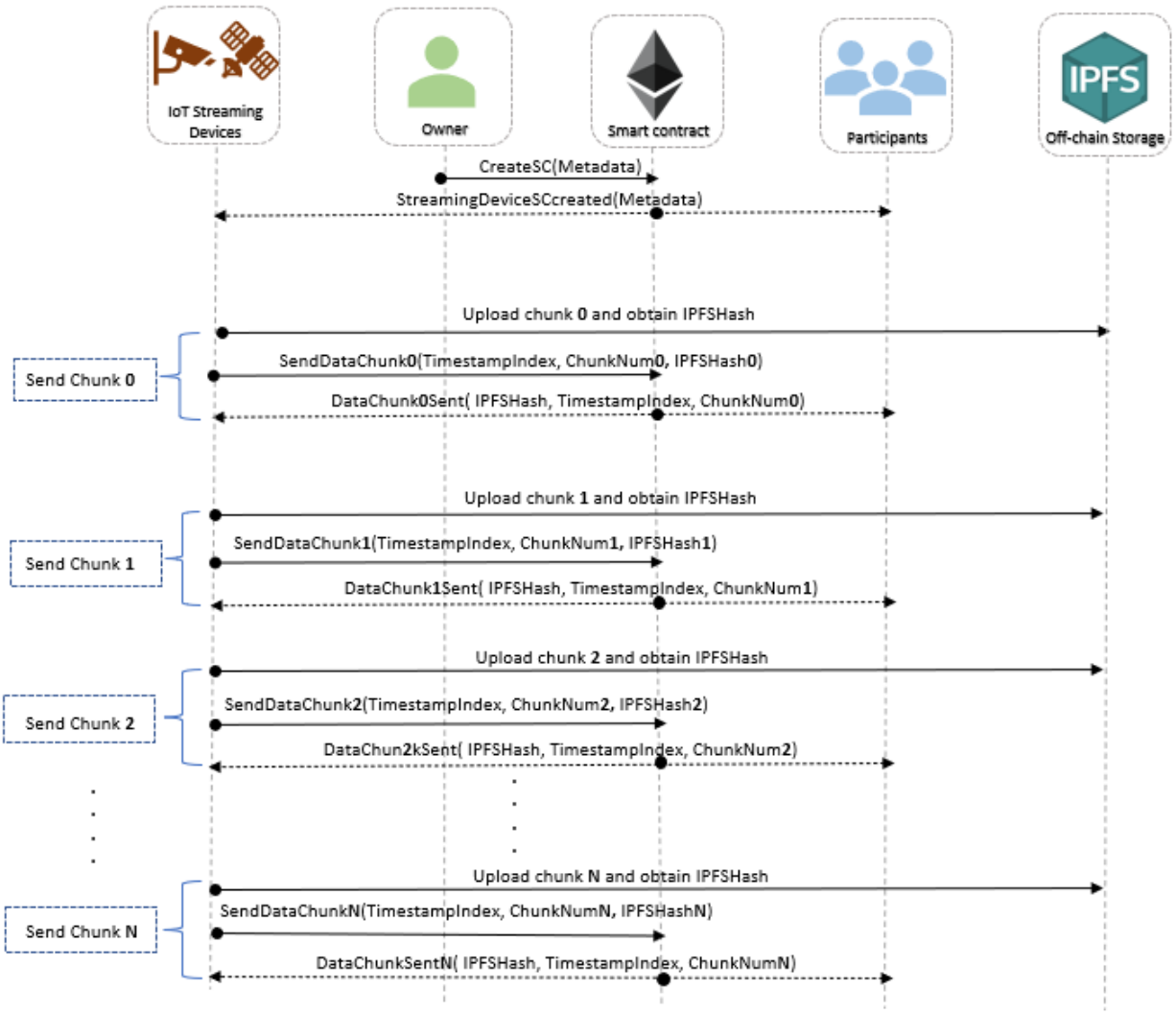

FIGURE 3: Sending the data chunks sequence diagram

that allows bidders to place their bids through the smart contract.

C. PARTICIPANTS

All the participants and users of the Ethereum blockchain have Ethereum addresses (EAs). The IoT streaming device also has its own EA that uniquely identifies it and ensures access roles are maintained among the different functions in the smart contract. The owner of the IoT streaming device creates the smart contract and can register any interested participants. The registered users can bid in an auction that sells the IoT streaming device. When the owner decides to sell the streaming device, an auction takes place, and bidding takes place on-chain. Only registered bidders can place their bids. The winner then transfers the Ether and the ownership of the device is changed accordingly. Participants can also verify the hashes of the streamed data stored off-chain from the on-chain provenance data. The hashes, as well as the timestamps and indices, are all stored in the logs and in events. Moreover, the participants can also verify the order of the streamed data chunks from the unique time-dependent indices available on-chain as well as the chunk numbers.

D. IPFS OR CLOUD STORAGE

Our solution uses decentralized storage, such as the IPFS [16] to store the data chunks sent by the IoT streaming device. The streaming device chunks the data stream into smaller chunks at regular intervals. Each chunk of streamed data is then hashed. The hashes, unique indices, and chunk numbers are stored on-chain without the actual data stream. This is done to avoid the extra costs associated with on-chain storage. Other decentralized storage systems that can be used include swarm [17] and filecoin [18]. A cloud storage option can also be used. However, the host must be trusted and accepted by all parties involved. In our solution, we relied on the IPFS to keep our implementation fully decentralized.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3149312, IEEE Access

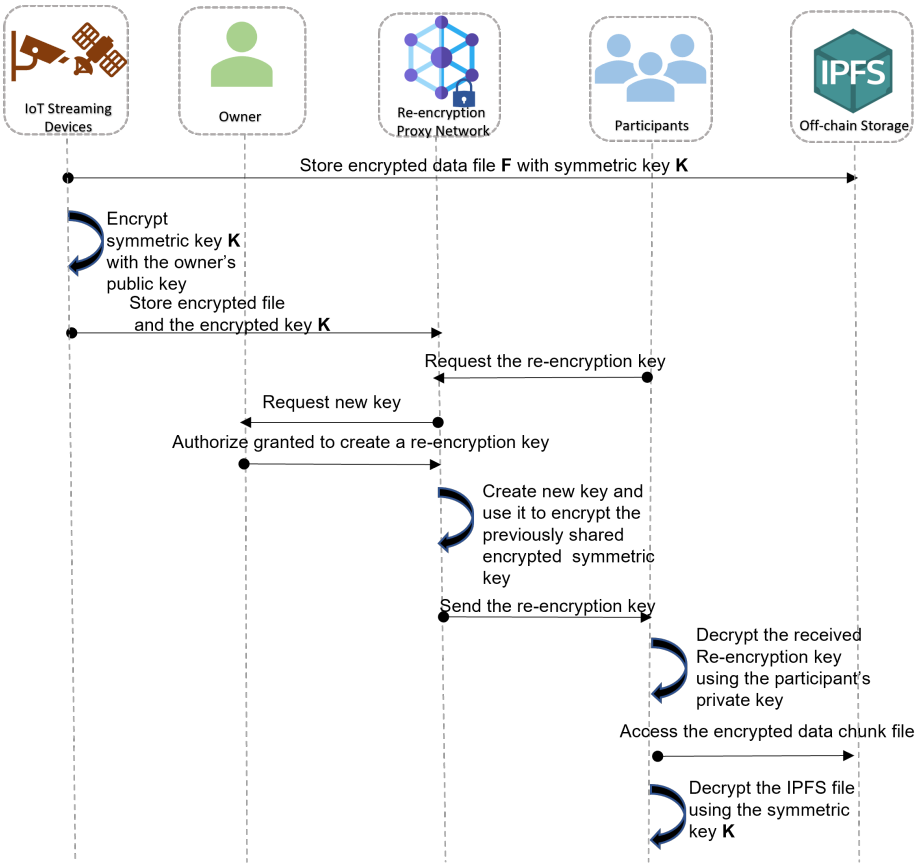

FIGURE 4: Managing data confidentiality using a proxy re-encryption network

E. IOT STREAMING DEVICES SMART CONTRACT

The IoT streaming device smart contract connects all the participating entities. It enables them to communicate and to notify one another about new events and updates. The IoT streaming device owner is the creator of the smart contract. The owner, IoT streaming device, and registered participants all have Ethereum addresses (EAs). The smart contract offers four main features for its users, as can be seen in figure 2. The owner can change the sampling period as well as metadata of the device on the chain through the smart contract. The sampling period varies based on the device type and the application it is used for. Hence, any change physically can be reflected on the chain as well. Moreover, the metadata can also be updated on the chain. The metadata contains information that specifies the geolocation through the latitude and longitude, as well as the domain name and IP address. For each of these changes, an event is emitted to notify all the listeners.

The smart contract also offers the ability to sell the IoT streaming device and change its ownership through an auction. The auction can only be attended by the registered participants. The registration is also done through the smart contract. Each participant has an EA and the EA is mapped to a boolean in the registration list. Registered participants can respond to the ad posted by the owner about the availability of the IoT streaming device for sale. The owner specifies the duration of the auction, the start and end times, as well as the starting price and minimum bid. The registered users can make their bids before the auction ends. At the end of the auction, the owner closes the auction and the payment is received from the winning participant and the expected new owner. The payment is stored in the smart contract where the smart contract acts as an escrow to emphasize trust and transparency between the buyer and seller. Once the ownership is passed to the buyer, who becomes the new owner, the Ether is transferred from the smart contract to the previous owner.

If the payment was received from the winner and the newly expected owner, but the seller did not change the ownership of the device, then the winner can request a refund. A refund is either granted or denied based on the time the request was made. The seller is provided with 10 minutes to change the ownership before the buyer can ask for a refund. This time can also be adjusted by the owner before deploying the smart contract. Providing a way to refund the buyer helps in preserving their rights. The timer of the owner's decentralized application helps in timing the auction in a decentralized way without depending on oracles or any third parties.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3149312, IEEE Access

F. SENDING THE DATA CHUNKS

The IoT data stream is chunked based on a sampling period. The data chunks are transferred by the IoT streaming devices off-chain for storage, and on-chain only their details (chunk number, timestamped index, hash) are transferred through the smart contract. The smart contract has a function that allows the IoT device to send the data chunk details on the chain as captured in figure 3. The figure explains the sequence of events involved in sending the streamed data details after the creation of the smart contract. The data chunk is first sent off the chain to the cloud storage or decentralized IPFS storage. For each data chunk, a hash is generated and the hash is stored on the chain. Furthermore, the communicated data chunk has a unique time-series index. The index used is the timestamp at the time of creation. This ensures that each data chunk has a unique index and a unique hash. On the chain, the timestamp index, hash, and the chunk number are logged. In figure 3, the details for each data chunk are illustrated. Each one is first uploaded into the off-chain storage, then sent on-chain with its details and unique index, number, and hash. An event is then emitted to notify all listeners. Any participating entity or listener can check for the chunk based on the hash or timestamp. Any of the chunk details can be used to filter the events and track the logs. The hash of the file ensures its integrity. After downloading the content from the storage, the data chunk can be hashed again and compared with the hash on the chain.

G. CONFIDENTIALITY USING A PROXY RE-ENCRYPTION NETWORK

The streamed data stored, whether on the cloud or IPFS, can be first encrypted to preserve its confidentiality and privacy. Consequently, this helps in allowing only authorized entities to access the decrypted streamed data. Moreover, this method allows multiple accesses to the same encrypted file by multiple users. Each user needs to be authorized to be granted access to the encrypted data chunk. However, the file containing the streamed data chunk is encrypted only once. We intend to keep the key that encrypts the file that holds one data chunk the same for a period X determined by the owner and the use case. So a user is granted single access for all data chunks in period X. A new access request must be sent to the owner to gain access to other data chunks saved as part of another period. In our solution, as seen in figure 4, we use a re-encryption proxy network [19], [20]. First, the IoT streaming device encrypts the file containing a data chunk, F using a symmetric key, K to get $E_K(F)$. Then the encrypted data chunk is stored off-chain. The streaming device then uses the owner's public key to encrypt the symmetric key, K to get the $E_{pub}(K)$. This is followed by the streaming device storing the encrypted streamed data chunk file $E_K(F)$ and the encrypted key $E_{pub}(K)$ in the re-encryption proxy network.

Whenever a participant requests access to the encrypted file, the request is sent to the re-encryption proxy network. The re-encryption proxy network sends the request to the owner, asking for a new key for the request they have received. If the owner agrees to share the streamed data with the participant, the owner authorizes the re-encryption proxy network to generate a new re-encryption key, N. First, the re-encryption proxy network creates the new key and encrypts with it the encrypted symmetric key to get the re-encryption key $E_N(E_{pub}(K))$. Secondly, the re-encryption proxy network sends the re-encryption key created to the participant. The participant decrypts the received re-encryption key with their private key $D_prv(E_N(E_pub(K))$ to obtain the symmetric key K.The participant can now access the streamed data for a period X from the off-chain storage and then decrypt the file, $D_K(E_K(F))$ using the key K to get the file of the streamed data F. The same key can be used to decrypt all the data chunks stored in a period X, after which a new key must be requested to gain access to the next streamed data chunks. This helps in restricting access per user to a limited period.

III. IMPLEMENTATION DETAILS

This section discusses the algorithms and the execution details of the functions used in the smart contract. The smart contract code is written in Solidity using the Remix IDE [21]. All the participating entities, including the IoT streaming devices, have Ethereum Addresses (EAs). Most of the algorithms depend on a certain smart contract state to execute. Others just depend on the deployment of the smart contract and can be executed at any time, such as updating the sampling period by the owner. All of the functions can only be executed by a certain entity. Hence, modifiers are used to restrict the caller's EAs and to ensure that only authorized entities are making the function calls. The functions can be executed after meeting the algorithm's criteria; otherwise, the smart contract reverts to the previous execution state. In our implementation, we have depended on decentralized storage of the IPFS. However, cloud storage can also be used and the hash of the streamed data chunk file stored on the cloud can be used in the algorithm 11 instead of the IPFS hash where needed.

A. CONSTRUCTOR: CREATING THE SMART CONTRACT

The constructor is an automatically called function when the IoT streaming device owner deploys the smart contract. It is the first function to be executed. Algorithm 1 shows the initial values set when the smart contract is created. The metadata is a $bytes32$ variable. The information that the metadata provides includes the IP address, latitude and longitude, domain name, and description. First, using $bytes32$ for such information helps in avoiding the cost associated with arrays and strings [22]. Secondly, as long as the size of the data stored is arbitrary, it is always a good alternative to use $bytes1$ to $bytes32$ rather than using $strings$. $Strings$ can be used when the information requires dynamically allocated storage.

Moreover, using $bytes32$ allows storing short strings. $Bytes32$ can store 32 bytes of data, which is equivalent

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3149312, IEEE Access

to 256 bits. Mostly, short strings are needed in our code. Therefore, $bytes32$ is more than enough. We can terminate a short string and fit it into a $bytes32$ variable type by using a null character. The null character is 1 byte. Hence, a $bytes32$ variable can store up to 31 bytes, where each byte is a character [23]. However, a 31-byte string may be shorter than 31 characters because some UTF-8 encoded characters take more than 1 byte [23]. The reason why $Strings$ are more expensive compared to bytes is because of the way they store the data. $Bytes$ store raw data, whereas $strings$ store the data in UTF-8 encoding of the real string [24].

Consequently, we have chosen to use bytes32 to ensure that the code execution is cost-efficient and the variable's modification or creation is not expensive. Cost is an important factor in on-chain processing. The constructor also sets the sampling period, state, and initializes the Ethereum address (EA) of the IoT owner. At the end, it emits an event to notify all participating entities that the streaming device smart contract has been created using the initialized variables.

Algorithm 1: Constructor: Creating the smart contract

Input : caller

$Metadata =$
$'0x38302e3231372e32382e35392c32302c31342c6$
$f72672c566964656f53747265616d696e67'$
$samplingPeriod = 7$
$IoTOwner = caller$
$state = created$
Emit an event showing that the Streaming Device smart contract is created using the metadata and sampling period

B. SAMPLING PERIOD UPDATE BY THE OWNER

The sampling period, initialized when the smart contract is created, can be updated by the owner at a later time. Depending on the use case and device hardware settings, the sampling period might change. Therefore, the algorithm 2 can be used to update the sampling period on-chain. This function can only be executed by the smart device owner. The algorithm checks the EA of the caller and validates it with the owner's EA. It also updates the sampling period and then announces the new change to all the listening entities.

C. METADATA UPDATE BY THE OWNER

Similarly, metadata is a field that can be altered if any of its fields change. The meta data includes information on the device's IP address and location, such as latitude and longitude. It also helps in identifying the device using the domain name and a brief description of its use and its specs. The arbitrary information is stored as raw data in the $bytese32$ format as it uses less gas compared to strings. The metadata can only be updated by the device's owner. Algorithm 3 includes the verification of the caller's EA and ensuring it is the owner only. The metadata is then successfully updated, and an event is emitted announcing the new change to the listening participants.

Algorithm 2: Sampling Period Update by the Owner

Input : caller, IoTOwner, state, sp
$IoTOwner$ holds the Ethereum Address of the current device owner
$caller$ holds the Ethereum Address of the function caller
$state$ is a variable that has the contract state
sp holds the new value of the sampling period variable
if $caller == IoTOwner \ \wedge \ state >= created$ **then**
    $samplingPeriod = sp$
    Emit an event announcing the updated value of the $samplingPeriod$
end
else
    Preview an error and return the contract to the previous state.
end

Algorithm 3: Metadata Update by the Owner

Input : caller, IoTOwner, state, md
$IoTOwner$ holds the Ethereum Address of the current device owner
$caller$ holds the Ethereum Address of the function caller
$state$ is a variable that has the contract state
md a bytes32 variable holds the new value of the metadata variable
if $caller == IoTOwner \ \wedge \ state >= created$ **then**
    $metadata = md$
    Emit an event announcing the updated value of the $metadata$
end
else
    Preview an error and return the contract to the previous state.
end

D. ADVERTISING THE STREAMING DEVICE FOR SALE

If the owner decides to make the IoT streaming device available for sale, then the owner can execute a function call that runs the algorithm 4. This function ensures that only the owner has the authority to make the call. It also checks the current state of the smart contract before it emits an event and notifies all the participants of the availability of the device for sale. The state of the smart contract is then updated to $advertisedForSale$ as shown in algorithm 4.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3149312, IEEE Access

Algorithm 4: Advertising the Streaming Device for Sale

Input : caller, IoTOwner, state

$IoTOwner$ holds the Ethereum Address of the current device owner
$caller$ holds the Ethereum Address of the function caller
$state$ is a variable that has the contract state
if $caller == IoTOwner \ \wedge \ state == created$ **then**
    Emit an event announcing the availability of the device for sale
    $state = advertisedForSale$
end
else
    Preview an error and return the contract to the previous state.
end

E. ANNOUNCING THE BEGINNING OF THE AUCTION

The IoT streaming device is sold through a bidding auction. All the registered bidders can take part in the bidding. The registered bidders are saved in a mapping in the smart contract. A mapping between their EAs and a boolean. If the EA is mapped to true, then they are registered; otherwise, false means an unregistered participant and an unqualified bidder. In the algorithm 5, the owner announces the beginning of the auction. This algorithm checks for the state and the caller's EA first. The minimum bid is set by the owner when executing this function, as well as the auction's duration. The bidders are expected to know this information before the auction begins. Consequently, they are announced through this function. The function saves the current block timestamp for later use and updates the state of the smart contract to $biddingInProgress$. At the end of the execution, the smart contract emits an event announcing the beginning of the auction using the current time stamp, minimum bid, and duration.

F. PLACING A BID

The registered bidders can now place their bids through the smart contract. This is done using the algorithm 6. The function takes the bid from the registered bidder. It checks the caller's EA and ensures it is part of the registered bidders' mapping. It also checks that the state of the smart contract is between $advertisedForSale$ and $biddingEnded$. Additionally, it checks that no one is currently bidding. This is important to avoid having multiple bidders at the same instant executing the function. Furthermore, the function also checks that the bidding time has not elapsed. If all the checks are met, then the state is updated to $someoneIsBidding$ accordingly. The bids are only accepted if they are higher than the last bid placed by a previous bidder. Based on the bid amounts, the values of the devices are updated, as well as the EA of the winner. At the end, an event is emitted to announce

Algorithm 5: Announcing the Beginning of the Auction

Input : caller, IoTOwner, state, minBid, duration

$IoTOwner$ holds the Ethereum Address of the current device owner
$caller$ holds the Ethereum Address of the function caller
$state$ is a variable that has the contract state
$minBid$ is a variable that holds the minimum bid a bidder can make
$duration$ is a variable that sets the duration of the auction
if $caller == IoTOwner \ \wedge \ state == advertisedForSale$ **then**
    $currentMaxBid = minBid$
    $biddingWinner = caller$
    $biddingDuration = duration$
    $curremtTimeStamp = block.timestamp$
    $state = biddingInProgress$
    Emit an event announcing beginning of the auction along with the $cuurentTimestamp$, $minBid$ and $duration$
    $state = biddingInProgress$
end
else
    Preview an error and return the contract to the previous state.
end

the current maximum value of the streaming device.

G. ANNOUNCING THE END OF THE AUCTION

The auction ends when the owner's decentralized application triggers the function that executes algorithm 7. In our solution, the auction doesn't depend on third parties or oracles for its beginning or end. It also doesn't depend on timers. This is because the concept of multi-threading and timers doesn't exist in solidity. However, the owner can execute a function call to end the auction. The algorithm checks if the caller is the owner and the current state must be $biddingInProgress$, otherwise the contract returns to the previous execution state. The algorithm then checks if the time has not exceeded the auction time. It then updates the bidding time variable and the state to $biddingEnded$. At the end, the participating entities are notified by an event about the end of the auction.

H. PROCESSING PAYMENT OF THE AUCTION WINNER

The winner of the auction is the last bidder with the highest bid value. The EA of the winner is used to restrict the function call to only them. Only the winner is allowed to pay for the device. Therefore, in the algorithm 8, the caller's EA is verified, and the state must show that the bidding has ended. This algorithm explains the steps of the payable function in the smart contract. In this function, the winner pays using

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3149312, IEEE Access

Algorithm 6: Placing a Bid

Input : caller, state, minBid, duration, bidAmount, RegisteredBidders,currentTimeStamp

$caller$ holds the Ethereum Address of the function caller
$state$ is a variable that has the contract state
$minBid$ is a variable that holds the minimum bid a bidder can make
$duration$ is a variable that sets the duration of the auction
$bidAmount$ is a variable that the participant sets with their bid
$RegisteredBidders$ is a list of registers bidders
$currentTimeStamp$ holds the timestamp when the auction started
$currentMaxBid$ holds the highest bid so far
$biddingWinner$ holds the EA of the highest bidder
if $caller \in RegisteredBidders \ \wedge \ (state > advertisedForSale \ \wedge state < biddingEnded) \ \wedge \ state \neq someoneIsBiddingNow$ **then**
  if $block.timestamp < (currentTimeStamp + duration)$ **then**
    $state = someoneIsBiddingNow$
    if $bidAmount > currentMaxBid$ **then**
      $currentMaxBid = bidAmount$
      $biddingWinner = caller$
      Emit an event stating that a new bid is offered with the $currentMaxBid$
    end
  end
  $state = biddingInProgress$
end
else
  Preview an error and return the contract to the previous state.
end

Ether the bid they have placed previously in the algorithm 6. Therefore, here, the algorithm checks the message value to ensure it matches the expected amount of the bid. After that, the transaction is completed, and the Ether is deducted from the winner. An event is emitted to notify all listeners that the payment has been received by the new owner. The state is also changed to payment received. The Ether is placed in the smart contract itself, so the smart contract acts as an escort to ensure trust is achieved. The payment is only transferred to the previous owner (seller) after transferring ownership to the winner. This protects the rights of both parties.

I. CHANGING THE OWNERSHIP OF THE IOT STREAMING DEVICE

Once the payment is made by the winning bidder, the owner must declare the ownership to the Ethereum address of the winner. Algorithm 9 checks the EA of the owner and the

Algorithm 7: Announcing the End of the Auction

Input : caller, IoTOwner, state, endBiddingTime, currentTimeStamp, duration

$IoTOwner$ holds the Ethereum Address of the current device owner
$caller$ holds the Ethereum Address of the function caller
$state$ is a variable that has the contract state
$currentTimeStamp$ holds the timestamp when the auction started
$endBiddingTime$ is a variable that holds the time the bidding ended at
$duration$ is a variable that sets the duration of the auction
if $caller == IoTOwner \ \wedge \ state == biddingInProgress$ **then**
  if $block.timestamp < (currentTimeStamp + duration)$ **then**
    $endBiddingTime = block.timestamp$
    $state = biddingEnded$
    Notify all participants that the bidding ended by the owner
  end
end
else
  Preview an error and return the contract to the previous state.
end

Algorithm 8: Processing Payment of the Auction Winner

Input : caller, state, courier, biddingWinner

$caller$ holds the Ethereum Address of the function caller
$state$ is a variable that has the contract state
$biddingWinner$ holds the EA of the highest bidder
$currentMaxBid$ holds the highest bid so far
if $caller == biddingWinner \wedge \ state == biddingEnded \wedge \ msg.value == currentMaxBid$ **then**
  Emit an event stating that the payment is received from the new owner $biddingWinner$
  $state = paymentReceivedByWinner$
end
else
  Preview an error and return the contract to the previous state.
end

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3149312, IEEE Access

state to ensure that the payment has already been received before proceeding further. The Ether is then transferred to the previous owner, and the owner variable of the IoT device is now updated with the EA of the winner. At the end, the state of the smart contract is updated to $newOwner$ and an event is emitted to announce the new owner of the device using the owner's EA.

Algorithm 9: Changing the Ownership of the IoT Streaming Device

Input : caller, IoTOwner, state, biddingWinner,currentMaxBid

$IoTOwner$ holds the Ethereum Address of the current device owner
$caller$ holds the Ethereum Address of the function caller
$state$ is a variable that has the contract state
$biddingWinner$ is the EA of the highest bidder
$currentMaxBid$ is a variable that holds the highest value of the device
if $caller == IoTOwner \ \wedge \ state == paymentReceivedByWinner$ **then**
    Transfer $currentMaxBid \rightarrow caller$
    $IoTOwner = biddingWinner$
    Emit an event announcing new owner of the device $biddingWinner$
    $state = newOwner$
end
else
    Preview an error and return the contract to the previous state.
end

J. PROCESSING A PAYMENT REFUND

The previous owner has X minutes to change the ownership of the IoT streaming device after the end of the auction. This can be modified and agreed upon before deploying the smart contract. In our execution and testing, we used 10 minutes as a reasonable time estimate for the owner to change the ownership. The winning bidder has the right to ask for a refund from the smart contract if the owner fails to update the ownership status within the allocated time frame. Algorithm 10 checks that only the winning bidder can ask for a refund. It also ensures that the payment was already received for longer than X minutes in order to proceed with the refund request. If all the conditions are met, the Ether is transferred from the smart contract to the winning bidder and the state of the contract goes back to the initial state, which is $created$. The function also emits an event to notify all listeners that the Ether has been refunded.

K. TRANSFERRING DATA CHUNKS

The IoT streaming device saves the data chunks on the IPFS. Each data chunk has a unique IPFS hash as well as a timestamp. The IPFS hash is stored in the smart contract as $bytes32$. The IPFS hash as a string is 46 bytes long. Strings use UTF-8 encoding. Therefore, some characters use more than 1 byte. Hence, the timestamp is used as an index to uniquely identify and map the data chunks. The time stamp also helps in ordering and maintaining a sequence, along with the chunk number. In algorithm 11, only the IoT device can execute the call to send the streamed data chunk details on-chain. For every data chunk, an event is emitted which logs the data chunk's number, timestamp index, and IPFS hash.

Algorithm 10: Processing a Payment Refund

Input : caller, IoTOwner, state, biddingWinner,endBiddingTime,currentMaxBid

$caller$ holds the Ethereum Address of the function caller
$state$ is a variable that has the contract state
$biddingWinner$ is the EA of the highest bidder
$endBiddingTime$ is a variable that holds the time the bidding ended
$currentMaxBid$ is a variable that holds the highest value of the device
if $caller == biddingWinner \ \wedge \ state == paymentReceivedByWinner$ **then**
    if $block.timestamp >= (endBiddingTime + X \ minutes)$ **then**
        Transfer $biddingWinner \rightarrow currentMaxBid$
        Emit an event stating that the refund is granted
        $state = created$
    end
    else
        Emit an event that the refund is denied
    end
end
else
    Preview an error and return the contract to the previous state.
end

IV. TESTING AND VALIDATION

We develop a smart contract that is written and tested using the Remix IDE [21]. Each function was compiled to check for proper syntax and run with different scenarios to check for the right expected functionality. The functions utilize modifiers to ensure user access is granted based on the user's role. The state of the smart contract is checked before the execution of the algorithm and updated to the next state at the end of the execution. Events are emitted to notify all listeners and participants of updates and notifications. The IoT streaming device can be sold through an auction. The details of the testing along with the results are discussed in the following subsections. To ease the testing process, some variables, such as the EAs are hard-coded in the code rather

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3149312, IEEE Access

Algorithm 11: Transferring Data Chunks

Input : caller, IoTDevice

$IoTDevice$ holds the Ethereum Address of the device
$caller$ holds the Ethereum Address of the function caller
$timestampIndex$ is a variable that is used to uniquely identify a chunk using its timestamp
$chunckNum$ is the chunk number
$IPFShash$ is the IPFS hash of the chunk
if $caller == IoTDevice$ **then**
    Emit an event announcing data chunk details including the $timestampIndex$, $chunckNum$, $IPFShash$
end
else
    Preview an error and return the contract to the previous state.
end

than entered at the execution time.

In our testing scenarios, the IoT device holds $0x583031D1113aD414F02576BD6afaBfb302140225$ as Ethereum addresse (EA), the IoT device owner holds $0xca35b7d915458ef540ade6068dfe2f44e8fa733c$ as EA, and the registered participant and new owner holds $0x14723a09acff6d2a60dcdf7aa4aff308fddc160c$ as EA.

A. IOT STREAMING DEVICE SMART CONTRACT CREATION AND SAMPLING PERIOD UPDATE

The smart contract is first created by the IoT device owner. The owner has the right to create the smart contract, change the sampling period, and change the metadata. Those functions have been tested successfully and the result of any change was shown in the logs as can be seen in figure 5 where the sampling period was changed to 5 seconds after the creation of the smart contract. The event $NewSamplingPeriodAnnounced$ is emitted with the argument 5 as illustrated in the figure.

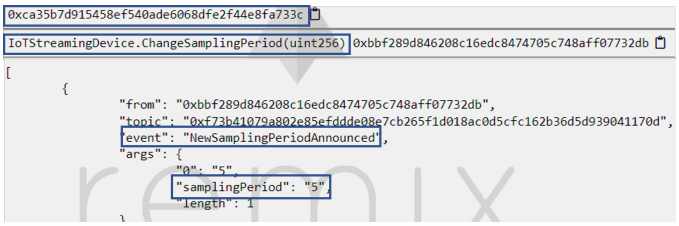

FIGURE 5: Logs showing a successful change of sampling period

B. IOT SALE AND AUCTION COMMENCEMENT

The IoT device can be sold to a new owner. This is done by the current owner by advertising the device for sale. Then the owner starts the auction and sets the minimum bid allowed, and the auction duration. Figure 6 shows a successful auction commencement with the minimum bid set by the owner as 20 Ether and the auction duration is 10 minutes. The event $BiddingCommenced$ emitted indicates the successful execution of the $StartBidding$ which follows the execution of the $AdvertiseStreamingDevice$.

FIGURE 6: Logs of the device owner commencing the auction

C. REGISTERED PARTICIPANTS SUCCESSFULLY MAKING A BID

A registered user can successfully place a bid as long as the auction didn't time out, no other participant is currently bidding, and the bid placed is higher than the minimum and previous bid. In figure 7, the registered participant placed a bid of 50 Ether which was successfully accepted and the event $NewBidOffered$ was emitted, as can be seen in the figure.

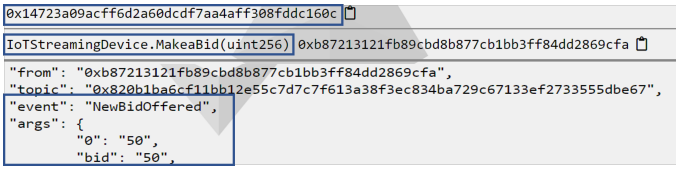

FIGURE 7: Logs showing a participant successfully making a bid

D. THE WINNER MAKING THE PAYMENT

The auction is then successfully ended by the IoT owner. This allows the winning participant to make the payment. Figure 8 shows the winner making the payment successfully with a value of 50 Ether. The event $PaymentReceivedFromNewOwner$ is emitted as a result of correct execution.

FIGURE 8: Logs of the winner paying for the streaming device

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3149312, IEEE Access

E. CHANGE OF OWNERSHIP

The ownership change has been tested successfully, as illustrated in figure 9. After the payment was made, the IoT owner changes the ownership and an event is emitted, $NewOwnerAnnounced$ to announce the new owner of the device. If the new ownership is not claimed within X minutes from the end of the auction, then a refund can take place. We used 10 minutes in our testing.

FIGURE 9: Logs showing the EA of the new owner

F. TRANSFERRING THE DATA CHUNKS

The data chunks can only be transmitted by the IoT device. Therefore, in the testing, any EA other than the IoT device EA will be denied execution. In our testing scenario, the IoT device sent a streamed data chunk with the timestamp index of 1635878813, chunk number 0, and hash 0x64EC88CA00B268E5BA1A35678A1B5316D212F4F366B2477232534A8AECA37F3C . Figure 10 shows the details of the successful execution. The function $SendDataChunk$ is executed successfully and the event $DataChunkSent$ is emitted as expected with the timestamp index, chunk number, and IPFS hash.

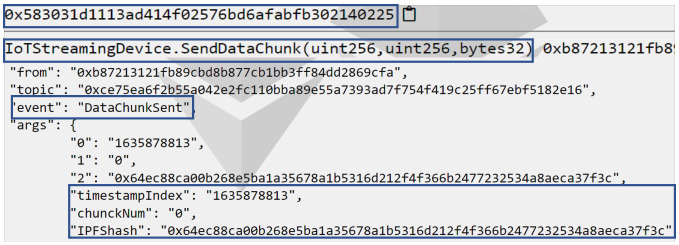

FIGURE 10: Logs showing a successful notification of a data chunk sent by the streaming device on-chain

V. DISCUSSION

In this section, we evaluate the security aspects of our solution and show how it is different from the other existing solutions. Also, we outline the limitations of our solution in terms of open challenges.

A. SECURITY ANALYSIS

Solidity code vulnerabilities are a huge threat and a risk. Cyber attacks can lead to unauthorized access and exploitation of the system and network. Hence, we have used the Oyente tool to scan our code and ensure that it is bug-free [25]. The code is written using the Remix IDE which checks for compiling errors and run-time errors. However, the security tools run a thorough and deeper analysis of the code. Consequently, security analysis tools are considered a more reliable method of analysis. The Oyente tool checks for the Ethereum Virtual Machine (EVM) coverage as well as several other insecurities. Those vulnerabilities include integer underflow, integer overflow, transaction ordering dependency, and timestamp dependency. The tool analyzes the code and generates a report with the EVM coverage and a boolean that states true if any vulnerability is found. Our smart contract code was analyzed using the Oyente tool, and the result generated, is in figure 11. The result shows a high EVM coverage and a false for all possible insecurities. It is worth mentioning that the tool only accepts low compiler versions of solidity. Therefore, the code must be syntax checked to meet the lower compiler version when using the Oyente tool. Also, the tool provides instructions and line numbers to fix any possible weaknesses and rerun the code. Our code was iterated multiple times to ensure it meets the security standards and compiling version of Oyente.

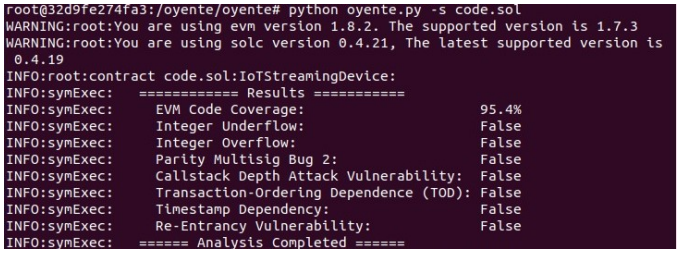

FIGURE 11: Security analysis results of our smart contract code from the Oyente tool

In addition to the security analysis provided by the Oyente tool. We would like to highlight the intrinsic security features of blockchain technology. We have leveraged its characteristics to meet our needs. Blockchain provides integrity, accountability, authorization, non-repudiation, and transparency. Confidentiality can also be acquired when needed by using a private blockchain or through encryption depending on the use case and application.

In our solution, transparency and trust are maintained for all the participating entities, as all events and function executions are logged. The logs are immutable, which eliminates theft, data breaches, and tampering.

Furthermore, the hash uploaded by the IoT streaming device on-chain as well as the streamed data chunk details are all logged in tamper-proof-logs. The integrity of the logs reassures all the listeners that the data provided on-chain can be trusted as it cannot be exploited.

All actions performed on-chain can only be executed by a certain authorized entity. This is maintained by using modifiers that check the EA of the executor. All users are held accountable for their actions on-chain. Consequently, accountability is maintained and enforced through the logs that show the caller's EA. Moreover, non-repudiation is also achieved as all the transactions are signed by the caller's EA.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3149312, IEEE Access

TABLE 1: Comparison with the existing solutions

Approach	Description	Storage	Blockchain Role	Privacy
[10]	Blockchain-based solution for sharing weather sensor data in a marketplace. NTUA token is used.	SQL relational like database called Maria database (DB) for the blockchain events. Weather data can be stored in a centralized server.	Ethereum blockchain for events and access management.	Only users of the DApp must provide a key to access the requested data.
[11]	Decentralized access management system for IoT devices.	A CoAP server is used. Queries can be done through an agent node or stored privately on a network accessible storage.	Access management using an Ethereum blockchain.	A private blockchain is used.
[12]	Decentralized marketplace for brokered IoT data.	No storage is required. Data is exchanged off -chain between producer and consumer.	Ethereum smart contracts are used as a trusted intermediary.	A private blockchain is used.
[13]	Generic blockchain-based solution for access control.	Off-chain storage is suggested.	Identity management and access control of data.	Data is encrypted.
[14]	Distributed access control and management system of IoT data.	Decentralized storage or cloud storage.	Access control and management of data.	key regression is used for encryption.
[15]	Decentralized access model for IoT data using a consortium architecture.	IPFS.	Access control based on stored keys in the smart contract.	The IPFS hashes are encrypted.
Our proposed Solution	Decentralized blockchain-based solution for IoT streaming.	IPFS or Cloud storage.	Ethereum is used for access logs and data provenance, bidding auction, an immutable trusted ledger for sharing a data chunk hash.	Ensures confidentiality using a Proxy Re-encryption Network.

The streamed data sent by the IoT streaming device can also be encrypted to maintain the confidentiality and privacy of the data. In our solution, this is feasible by encryption using a proxy re-encryption network.

B. COMPARISON WITH THE EXISTING SOLUTIONS

Table 1 compares the existing blockchain-based IoT solutions with our proposed system. As can be seen from the table, [10], [12] are solutions that focus on building a marketplace for the IoT data. These solutions are different, and their used storage depends on a database or on exchanging messages off-chain, respectively. On the other hand, [11] is a decentralized access control system for IoT devices that depends on using a CoAP server. Also, all IoT devices are grouped with agent nodes and are provided with keys generated by the system. Our solution allows IoT devices to have EA addresses and does not use any servers. Our solution also does not require a private blockchain for privacy. Furthermore, the solutions provided by [13]–[15] focus on providing access control and management system. They all share the importance of using decentralized off-chain storage. However, [13] is not adaptable to IoT streamed data, unlike [14], [15]. In [15], the authors use side chains and the blockchain. The side chains have smart contracts that store the keys of the IoT devices. All smart contracts used in their solutions store the public keys of the participants, and according to the stored keys, access privileges are provided for an allocated time. Their used approach depends on on-chain storage of keys, which is very different from our approach, where we do not store keys and every participant has an EA. Modifiers in the smart contract provide access roles and rights. Another major difference is that the authors rely on storing encrypted IPFS hashes in their smart contracts. In our solution, we encrypt the data stored on IPFS, but the IPFS hashes are in plain text and available on-chain in the logs. Additionally, the Bitcoin-based solution of [14] provides access control and management for IoT streams, where the data is chunked and transferred off-chain. This is similar to our approach. However, our Ethereum-based solution does not use virtual chains. Moreover, we depend on the intrinsic security features of the blockchain, where each block has a hash pointer. Therefore, we do not create a hash pointer in the stored data chunks, unlike [14]. The authors also rely on key regression for encryption. Our solution works on public plain text data and can also provide a layer of confidentiality based on the use case. This is done through a proxy re-encryption network to only allow access to the encrypted off-chain data based on a regenerated key for each allowed requester. All our detailed algorithms are also provided, as well as the code is made publicly available on GitHub. Therefore, we believe that our solution provides a unique method for streaming IoT devices using the Ethereum blockchain and decentralized storage.

C. OPEN CHALLENGES

Two key challenges in our blockchain-based solution are scalability and cost. Our solution is based on the Ethereum blockchain. Ethereum 1.0 relies on a single consecutive chain of consecutive blocks. It was created this way to ensure that it is highly secure. However, this compromised efficiency and

speed. The network's performance and processing speed are low, thereby causing great delays.

Scalability can be a major challenge, with 30 transactions per second when using the classical blockchain network Ethereum 1.0. However, Ethereum is being upgraded to a more scalable, reliable, and secure version which is Ethereum 2.0. The promising upgrade can eliminate delays and congestion with its ability to process 100 thousand transactions per second [26]. Additionally, the current version of Ethereum relies on proof of work (PoW), where mining nodes compete against each other to solve a highly complex mathematical problem. This makes the validators incentivized to use more energy and power to successfully mine a block. The new Ethereum 2.0, on the other hand, reduces the competition for miners by employing the proof of stake (PoS) model [27]. The blocks in the PoS consensus are randomly distributed among the validators. The new consensus in Ethereum 2.0 is accompanied by sharding to reduce congestion and increase the network throughput. Sharding is a process that spreads a database horizontally, which leads to spreading the load to new chains. The creation of new chains, known as shards, helps in increasing Ethereum's scalability and capacity. Ethereum 2.0 is expected to have 64 new shards [28]

Cost is a second major obstacle when employing the public Ethereim blockchain network. Currently, Ethereum is facing a spike in prices, which leads to very high transaction costs. However, a private blockchain does not rely on the PoW and hence, cost is not a factor to consider. Our solution is adaptable to any private blockchain. A private Ethereum blockchain can be used to ensure that cost does not hinder the feasibility of the solution. Moreover, there are several emerging permissionless Ethereum based blockchain networks like Algorand, Cardano and zkSync that are much cheaper and have a relatively high number of transactions per second. Algorand is working towards 46 thousand transactions per second and is much cheaper than many other Ethereum-based networks [29]. Cardano is built on PoS with a rewarding mechanism [30]. zkSync is a scaling solution for Ethereum with extremely low transaction fees [31].

VI. CONCLUSION

In this paper, we have proposed a blockchain-based solution to ensure IoT streaming data management and its access control in a manner that is decentralized, reliable, transparent, traceable, auditable, secure, and trustworthy. We integrated the Ethereum blockchain with off-chain storage, such as the IPFS, to deal with the large-size data storage issue. Using our solution, users can access the data from the off-chain storage and can easily check its integrity from the immutable logs and on-chain provenance data. We developed smart contracts for each resource-constrained IoT streaming device that can sell the device in an online auction, allow the IoT device to send data chunk hashes on the chain, and store all data chunks off-chain. We ensured confidentiality and privacy by using a proxy re-encryption network to grant users access to encrypted IoT data chunks. Our proposed approach enables users to have multiple accesses to the data chunk files in a certain predetermined period within the same accession request. Also, all the encrypted data chunk files are written to the off-chain storage only once, but multiple reads are possible. We presented eleven algorithms along with their implementation and testing details. We performed security analysis to show that our smart contract code is secure enough against well-known attacks, threats, and vulnerabilities. We compared our solution with the existing solutions to show its novelty and distinctive features.

VII. ACKNOWLEDGEMENT

This publication is based on work supported by the Khalifa University of Science and Technology under Awards No. CIRA-2019-001 and RCII-2019-002–Center for Digital Supply Chain and Operations Management.

REFERENCES

[1] C. Stergiou, K. E. Psannis, B.-G. Kim, and B. Gupta, "Secure integration of iot and cloud computing," Future Generation Computer Systems, vol. 78, pp. 964–975, 2018.
[2] A. A. Alli and M. M. Alam, "The fog cloud of things: A survey on concepts, architecture, standards, tools, and applications," Internet of Things, vol. 9, p. 100177, 2020.
[3] M. M. Mahmoud, J. J. Rodrigues, S. H. Ahmed, S. C. Shah, J. F. Al-Muhtadi, V. V. Korotaev, and V. H. C. De Albuquerque, "Enabling technologies on cloud of things for smart healthcare," IEEE Access, vol. 6, pp. 31 950–31 967, 2018.
[4] A. Banafa, "Iot and blockchain convergence: benefits and challenges," IEEE Internet of Things, 2017.
[5] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," International Journal of Web and Grid Services, vol. 14, no. 4, pp. 352–375, 2018.
[6] R. Zhang, R. Xue, and L. Liu, "Security and privacy on blockchain," ACM Computing Surveys (CSUR), vol. 52, no. 3, pp. 1–34, 2019.
[7] M. A. Khan and K. Salah, "Iot security: Review, blockchain solutions, and open challenges," Future generation computer systems, vol. 82, pp. 395–411, 2018.
[8] L. Ante, "Smart contracts on the blockchain–a bibliometric analysis and review," Telematics and Informatics, vol. 57, p. 101519, 2021.
[9] T. Gupta, R. P. Singh, A. Phanishayee, J. Jung, and R. Mahajan, "Bolt: Data management for connected homes," in 11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14), 2014, pp. 243–256.
[10] G. Papadodimas, G. Palaiokrasas, A. Litke, and T. Varvarigou, "Implementation of smart contracts for blockchain based iot applications," in 2018 9th International Conference on the Network of the Future (NOF). IEEE, 2018, pp. 60–67.
[11] O. Novo, "Blockchain meets iot: An architecture for scalable access management in iot," IEEE Internet of Things Journal, vol. 5, no. 2, pp. 1184–1195, 2018.
[12] S. Bajoudah, C. Dong, and P. Missier, "Toward a decentralized, trust-less marketplace for brokered iot data trading using blockchain," in 2019 IEEE international conference on blockchain (Blockchain). IEEE, 2019, pp. 339–346.
[13] H. Shrobe, D. L. Shrier, and A. Pentland, "Enigma: Decentralized computation platform with guaranteed privacy," 2018.
[14] H. Shafagh, L. Burkhalter, A. Hithnawi, and S. Duquennoy, "Towards blockchain-based auditable storage and sharing of iot data," in Proceedings of the 2017 on cloud computing security workshop, 2017, pp. 45–50.
[15] M. S. Ali, K. Dolui, and F. Antonelli, "Iot data privacy via blockchains and ipfs," in Proceedings of the seventh international conference on the internet of things, 2017, pp. 1–7.
[16] "IPFS is the distributed web," [Accessed on: June 13, 2018]. [Online]. Available: https://ipfs.io/
[17] "Swarm," [Accessed on: Nov 7, 2021]. [Online]. Available: https://www.ethswarm.org/

[18] "Filecoin," [Accessed on: Nov 7, 2021]. [Online]. Available: https://filecoin.io/
[19] H. R. Hasan, K. Salah, R. Jayaraman, J. Arshad, I. Yaqoob, M. Omar, and S. Ellahham, "Blockchain-based solution for covid-19 digital medical passports and immunity certificates," IEEE Access, vol. 8, pp. 222 093–222 108, 2020.
[20] S. Kim and I. Lee, "Iot device security based on proxy re-encryption," Journal of Ambient Intelligence and Humanized Computing, vol. 9, no. 4, pp. 1267–1273, 2018.
[21] "Remix," [Accessed on: July 27, 2020]. [Online]. Available: https://remix.ethereum.org/
[22] G. Zheng, L. Gao, L. Huang, and J. Guan, "Solidity basics," in Ethereum Smart Contract Development in Solidity. Springer, 2021, pp. 49–83.
[23] "bytes and strings in solidity," [Accessed on: Nov 28, 2021]. [Online]. Available: https://medium.com/@cryptopusco/bytes-and-strings-in-solidity-f2cd4e53f388
[24] "Working with strings in solidity," [Accessed on: Nov 28, 2021]. [Online]. Available: https://medium.com/hackernoon/working-with-strings-in-solidity-c4ff6d5f8008
[25] M. Debe, K. Salah, R. Jayaraman, I. Yaqoob, and J. Arshad, "Trustworthy blockchain gateways for resource-constrained clients and iot devices," IEEE Access, vol. 9, pp. 132 875–132 887, 2021.
[26] "What is ethereum 2.0 and why does it matter?" [Accessed on: Dec 7, 2021]. [Online]. Available: https://decrypt.co/resources/what-is-ethereum-2-0
[27] "Ethereum's latest progress toward proof-of-stake," [Accessed on: Dec 12, 2021]. [Online]. Available: https://www.coindesk.com/tech/2021/10/13/ethereums-latest-progress-toward-proof-of-stake/
[28] "Shard chains," [Accessed on: Dec 12, 2021]. [Online]. Available: https://ethereum.org/en/eth2/shard-chains/
[29] "What is algorand and why is it a blockchain to watch?" [Accessed on: Dec 7, 2021]. [Online]. Available: https://forkast.news/what-is-algorand-why-is-it-a-blockchain-to-watch/
[30] "Cardano," [Accessed on: Dec 13, 2021]. [Online]. Available: https://cardano.org/
[31] "Welcome to zksync," [Accessed on: Dec 13, 2021]. [Online]. Available: https://zksync.io/faq/