Decentralized Access Control for IoT Data Using Blockchain and Trusted Oracles

H. Albreiki, L. Alqassem, K. Salah, M. H. Rehman, D. Svetinovic
Department of Electrical Engineering and Computer Science, Khalifa University, Abu Dhabi, UAE
Email: khaled.salah@ku.ac.ae

***Abstract*—The Internet of Things (IoT) is a network of connected electromechanical devices that have limited computational, networking, and storage capabilities. IoT is now widely used in healthcare, smart cars, smart grids, smart homes, smart manufacturing, and smart cities. IoT devices sense, monitor, and collect data where it can be shared with legitimate users. IoT data can be aggregated, stored and made available by multiple IoT data hosting providers. IoT data storage, management, and access involve multiple stakeholders that many include admins, owners of IoT devices, data repository hosts and providers, normal users, etc. Decentralized control and trusted management of such IoT data become critical, in which the management and access control of data is not centralized, i.e., under the control of a single entity. To date, the available methods for for access control in IoT systems are mainly centralized. In this paper, we propose a decentralized access control system for IoT data using blockchain and trusted oracles. We use features of blockchain and smart contracts to propose a decentralized, scalable, and secure management solution for accessing IoT data. In addition, we use oracles as gateways that interface with the blockchain, IoT data hosts, and remote users to provide decentralized, trusted, and uniform source feeds for IoT data. The paper also presents architectural design, interactions, logic flow, algorithms, implementation details, along with cost, computation, and security evaluation. The full code of the developed smart contracts is made publicly available at GitHub.**

***Index Terms*—IoT, Blockchain, access control, smart contracts, trusted oracles, IoT data**

I. Introduction

The pervasiveness of Internet of Things (IoT) devices and systems has brought comfort and convenience in our daily life activities. Almost all of our daily activities are augmented through smart embedded devices which are scattered around us and are turning our cities, transportation systems, buildings, homes, and bodies into smart environments [1]. For example, in healthcare, IoT devices are used in the form of injectables, wearables, or implantables to collect vital signs data. The attained data can provide insight into the underlying symptoms, enable remote diagnosis and care, and generally allow patients to have control over their ongoing treatment and prognosis. Also, smart sensors deployed at our homes can deduce when we shower, what we eat, and where we sleep. Likewise, driver-less cars can autonomously drive us to our desired destinations. Moreover, smart watering systems can use real-time weather data which is collected through wind, temperature, and humidity sensors and it can forecast to create an optimal watering schedule for our farms and domestic gardens.

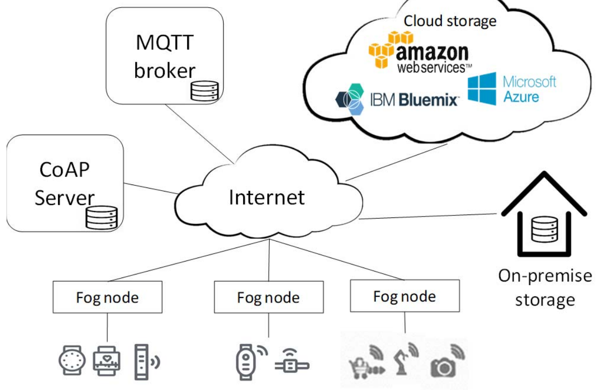

Figure 1. Storage Services for IoT data

The number of IoT devices to be deployed is expected to reach 20.5 billion devices by 2020, and their generated data is expected to exceed 800 Zettabytes [2]. The generated data could be cold (already collected), or hot (live or being streamed); in addition, 90% of the data will be unstructured [2]. This is why IoT technology is considered a big data problem, and there is a need to manage, analyze and control the access to the huge volumes of data. The current ecosystem of the IoT consists of three tiers: (1) low-power IoT devices at the forefront, (2) a potential middle-tier gateway that interconnects IoT devices with the back-end Internet servers, and (3) the back-end where IoT data is stored and analyzed to enrich IoT applications. Figure 1 illustrates different storage services for IoT data. The data can either be stored in the cloud data centers such as Amazon web services, Microsoft Azure and IBM Bluemix, MQTT/CoAP supporting centralized servers, or edge servers in customer premises.

One of the main aspects of IoT devices is sharing the IoT data with other entities. However, such data sharing may escalate the security risks and privacy concerns since malicious users can gain illegal access to the data and perform unauthorized alteration. As a result, access control is becoming an essential research topic in both academia and industry. There are several approaches to perform access control for IoT devices. One possible solution is to integrate such functionality into the IoT device itself. However, IoT devices have limited computing power and memory to store and manage Access Control Lists (ACL) and policies for authorized users–

making hard to scale and maitnain. As a result, new solutions have been proposed to mitigate these limitations. The widely adopted solution is the centralized client-server approach that is built by implementing communication protocols for resource-constrained devices. This solution manages access to resources in a centralized way using servers. The main disadvantage of the centralized solution is the single point of failure that prevents the IoT systems to scale-up. Furthermore, the centralized access control management systems are designed for managing static devices that belong to a specific group of devices during their lifetime. However, the IoT devices usually belong to multiple groups, and their scenarios change over time. In addition, the centralized model is built around the notion of trust where device data is entrusted with a centralized trusted entity. This may cause ethical and privacy problems since the centralized entity can make illegal use of IoT data. For instance, Fitbit, a company that manufactures small wearable devices to help people keep track of their fitness activities, believes that the collected data should be public by default [3]. Hereafter, new access control management approaches are needed.

This paper focuses on using blockchain technology to manage the access control for IoT devices and addresses the critical issues related to scalability and security [4]. A blockchain-based system can potentially store the access control policies in the blockchain through the exploitation of smart contracts that govern, grant, or deny access to IoT data requested by end users. Blockchain is a decentralized distributed and tamper-resistant ledger that does not need a central authority for management. Blockchain also offers robust decentralized security to the system in terms of integrity, authentication, authorization and confidentiality, since it uses strong cryptography algorithms, with good degree of scalability and efficiency as it remove centralized ACL managment. The data collected by IoT devices can be stored using different storage services (as shown in Figure 1), and each of these storage services requires its own solution for access control mechanism.

In this paper, we present a decentralized access control management system that considers different storage services used by IoT devices and the types of the stored data. Our approach supports multiple storage services and uses trusted entities called oracles to attest the external data being reported to the blockchain. In addition, oracles allow a blockchain to interact with the real, outside world [5]. Our proposed solution has several advantages over the centralized approach. For one thing, it increases system scalability as users and devices can be registered and delisted from the system easily without the need to modify the system architecture. Moreover, using trusted oracles for data transfer solves the issues related to the trustless blockchain network. With oracles, the data providers do not have to change their services to be interface with blockchain networks. Moreover, oracles act as gateways and can be easily integrated with private and public blockchain platforms and networks.

Our main contributions in this paper can be summarized as follows:

- We propose a blockchain-based system using Ethereum smart contracts to manage access control policies for IoT data access in a decentralized manner without a trusted third party.
- Our proposed system supports multiple storage services such as cloud storage services, MQTT brokerage services, and the storage services provisioned by CoAP servers. We make use of emerging concepts like trusted oracles to provide seamless interface between blockchain networks, IoT data hosts, and remote users in order to provide decentralized, trusted, and uniform source feeds for IoT data.
- We present and discuss our proposed system architecture and interactions among all participating users, devices, and trusted oracles communicating with Ethereum smart contracts; namely: IoT Data Access Smart Contract (SC), Reputation SC, and Aggregator SC.
- We present algorithms and discuss key functions, interfaces, and logic flow of the employed Ethereum smart contracts. The complete source code of the smart contracts has been made publicly available at Github[1].
- We evaluate our proposed solutions and developed algorithms in terms of cost, computation complexity, and resiliency against popular security attacks.

The remainder of the paper is organized as follows. The related work is given in Section II while Section III presents the discussion on key enabling technologies and the design of our proposed solution. Section IV shows the implementation details followed by detailed evaluation in Section V. Finally, the article is concluded in Section VI.

II. Related Work

Access control systems for IoT devices can be divided into two categories: centralized access control systems and decentralized access control systems.To date, most systems are centralized. In general, centralized systems are single point of failure, hacking, compromise, and privacy evasion. With the advent of blockchain, decentralized access control schemes have started to gain traction. In this section, we review and discuss decentralized schemes reported in the literature.

For the most part, decentralized access control systems are blockchain based. Blockchain enables IoT devices or their owners to formulate their own access control policies of the data they generate [4]. The smart contracts are usually used for implementing access control policies. The policies can either be deployed upon data, subject to the identity of the data controller or specific data; or upon the data controller for multiple data subjects. Another way to use blockchain for access control in IoT is to use blockchain as a database to store all access control policies for each pair of resource and requester in the form of transactions [4]. If an access request is admitted, the access grant transaction can be recorded on the blockchain and broadcasted to all nodes on the underlying

[1]https://github.com/DecentralizedACLIoT/DecentralizedACL_IoT/

peer to peer network. Otherwise, the access request transaction is rejected, and a notification is sent to its sender.

A few smart contract based access control management schemes for IoT data were reported in [6], [7], [8], [9], [10], but with no consideration to different storage services for IoT data. The reviewed systems either make the data exchange part of the blockchain transactions as in [8] or they use a third-party to exchange data directly between two IoT devices as in [6], [7]. The author in [6] presented a blockchain-based access control system for data exchange between IoT devices and compared his system to the state-of-the-art lightweight machine to machine (LwM2M) servers. The results showed that the blockchain-based system is comparatively more scalable when the load is distributed between the participating nodes on the blockchain network. The system consists of six components: (1) *wireless sensor network* (WSN), which represents the network of connected IoT devices that communicate with each other using constrained application protocol (CoAP) and are uniquely identified in blockchain through public keys. The WSN is not part of the blockchain network; however, they interact with blockchain via the management hub (2) *managers*, who registers the devices and define their access control policies, (3) *agent node*, which distributes smart contracts, (4) *smart contracts*, where all system management operations are defined, (5) *blockchain network*, and (6) *management hub*, which is used for data exchange between IoT devices. It acts as an interface between the IoT devices and blockchain network. The managment hub receives requests from IoT devices, checks access permission using smart contracts, and responds to the IoT device by sending the requested data (if the permission is verified). The lack of support in dynamic IoT environments is the major drawback of this system.

Researchers in [7] extended and improved the work done in [6] to cope with various IoT scenarios. They proposed a dynamic decentralized access control system, whereby the system managers can dynamically generate access control policies for authenticated devices. In this system, the IoT devices are classified into three classes: *class 1* represents unregistered devices with restricted access, *class 2* shows the group of registered devices without any access control policy, and *class 3* embodies the registered devices with predefined access control policies. The system has two main components: 1) *management hub*, which has the same role as in [6], and 2) *dynamic policy generator*. Without the dynamic generation of policies, the management hub will reject all devices without associated policy (class 2 devices). Therefore, a dynamic generation of policies is needed. Once the management hub receives queries from class 2 devices, it sends requests to the managers via smart contract to create a policy for an authenticated device and registers it on the blockchain. Once the new policy is registered, the management hub will perform the data exchange between the devices.

Shafagh et al [9] proposed a decentralized access control system for sharing IoT time-series sensor data. The system is divided into two logically separated layers: data plane and control plane. The data are stored off-chain at the edge of the network via a locality-aware distributed storage system and managed by the blockchain. The proposed system is targeting the distributed storage; however, they claimed that current distributed storage systems like IPFS and Filecoin are not suitable for IoT data. Hence, the authors used the distributed hash table (DHT) as an alternative solution. They stored the data in consecutive chunks of predefined lengths. The metadata of chunks are then stored in a DHT whereas actual chunks are then stored on nodes that are closer to the data owner. The authors claimed that their system can support any type of data storage like cloud storage and on-premise storage.

Two IoT systems were implemented in [8]. Both systems consist of two IoT devices (smart refrigerator and smart TV). The first system uses MQTT to send data securely between the two IoT devices. The second system replaces the MQTT with blockchain network and smart contract to store and get data from the blockchain network. The aim of their work is to analyze the security level of the systems by simulating attacks. The evaluation results showed that the blockchain-based solution is more secure. Researchers also used blockchain for authentication only and make access to the IoT devices off-chain. The system in [10], for example, proposed a decentralized authentication mechanism that grants users to have transparent and granular access control over their IoT devices. The system uses blockchain-enabled fog nodes with connectivity to smart contracts to authenticate the users who request access to registered IoT devices. If the user is authorized to access the IoT device, he can have a direct secure connection with the IoT device. Paillisse et. al. [11], on the other hand, took the concept a step further by extending the concept of group-based policy to support multi-administrative domains using blockchain. They used Hyperledger Fabric, a permissioned blockchain, to distribute the access control policies securely and to preserve the independence of each organization. Their aim is to allow users from one organization to access resources (i.e. IoT devices or data storage) which belong to a different organization. Their system architecture consists of three layers: (1) Policy interface: it allows the administrators to perform management operations like specifying users and creating/deleting policies. These operations are implemented in smart contracts. (2) The second layer is blockchain which ensures integrity and accuracy of stored data. (3) The last layer is Network layer where LISP-enabled routers are used to access the blockchain to check if a user is authorized to access a specific resource.

III. Enabling Technologies and Proposed Solution

In this section, we give a brief background on blockchain and other supporting decentralized technologies including smart contracts and trusted oracles which all can be leveraged to support and facilitate a decentralized access control for IoT data. We also describe and discuss our proposed solution detailing key system components and interactions among different actors.

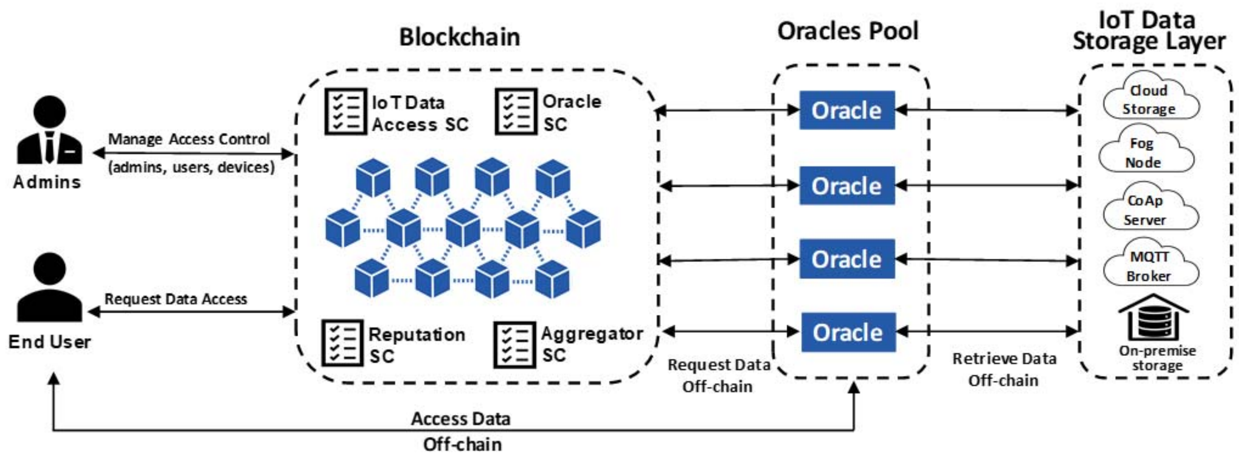

Figure 2. Proposed System Architecture

A. Enabling Technologies

There are three primary enabling technologies utilized in our propose solution.

1) Blockchain: Blockchain is the underlying technology of the bitcoin cryptocurrency. It is a distributed ledger that can hold transactions and records in an immutable, trusted, secure, and decentralized manner, with no use of intermediaries or centralized authorities [12]. These records and transactions are stored in blocks that are validated by thousands of mining nodes. The data in the chained blocks are hashed and a change in the block content will invalidate the whole chain of blocks.

2) Ethereum and Smart Contracts: Ethereum is an open source blockchain platform with a built-in Turing-complete programming language. It offers a decentralized virtual machine called Ethereum Virtual Machine (EVM) to handle smart contracts [13], [14]. A smart contract is basically a computer program that runs the business logic and agreement among participants based on predefined rules and without the need for a trusted third party [11]. Participants interact with the smart contract by calling its functions. The execution outcome of a smart contract functions is validated by all EVM nodes on the Ethereum network [10].

3) Trusted Oracles: By design, smart contracts cannot fetch external information on their own. To overcome this limitation, there is a need for external trusted entities to provide data feeds to smart contracts [15]. Such entities are called oracles, and their main purpose is to provide data to the blockchain. The provided data may be publicly available, such as asset prices or information on world events, or it may be private, such as bank account information or identity verification [15]. A single oracle to report information cannot be trusted. Therefore, multiple oracles are required to report feeds to smart contracts. Smart contracts need to validate and check reported data from multiple oracles to verify the trustworthiness of the reported data [15]. Decentralized mechanisms for reporting, consensus, and reputation are used for the oracles to ensure a high level of trust for the reported data.

B. Proposed Solution

In this section, we present a decentralized solution for accessing IoT data through the use of blockchain smart contracts and trusted oracles. The section gives a system overview, architectural, and main interactions. Figure 2 depicts system architecture with key components and actors. Multiple blockchain smart contacts are used to govern interaction and access to IoT data, and also to provide reputation score and registration of oracles that act as gateways to services and host of IoT data. Users interact with smart contracts to request access to IoT data, and admins responsible for setting the access rules and policy to IoT data. Multiple oracles can be selected by the end-user based on the reputation score to allow for off-chain access to IoT data. Access tokens are issued by smart contracts and given to the oracles and end-users to validate the right of access.

1) System Components: Our proposed system composed of the following components, as depicted in Figure 2:

- *Admins*. The system supports multiple administrators responsible for managing, controlling, and delegating user access to IoT data. The owner of the IoT devices is the actual creator of the *IoT data access* smart contract (SC), and who is also the primary admin. A primary admin can register secondary admins in the system. An admin can add new IoT devices and users to the system and it can delegate permissions to users in order to access IoT devices.
- *End-Users*. The system supports IoT devices (in case of machine to machine communication) and human users (in case of humane to machine communication). The end-user interface is done through DApp or wallets that have built-in blockchain libraries and APIs to provide on-chain interactions with the smart contracts and blockchain net-

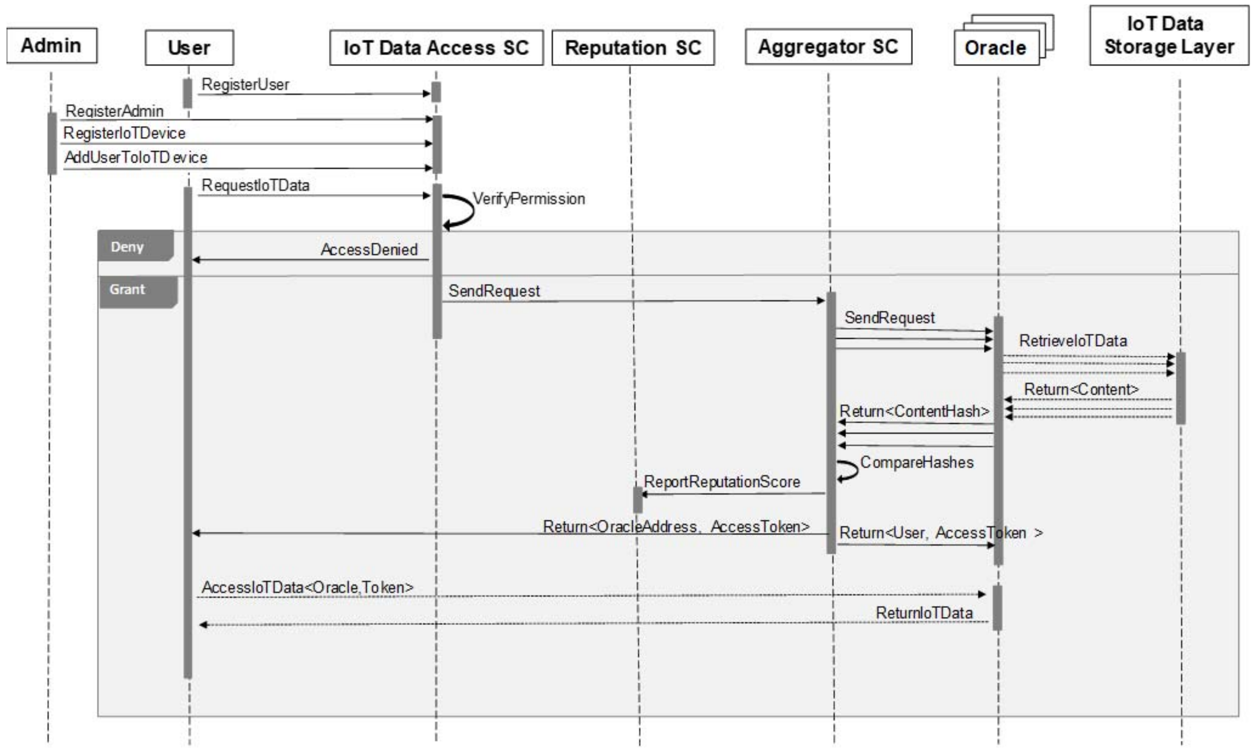

Figure 3. Sequence diagram for proposed system interactions

works, as well as off-chain interactions with the oracles to access IoT data from the IoT storage layer.

- *Blockchain Network*. The blockchain network hosts Ethereum smart contracts that govern and manage all transactions in a decentralized, secure, and trusted manner. We use four different smart contracts (SCs).
- *IoT Data Access SC*. This smart contract manages access control for IoT data and verifies user permissions for accessing IoT data. End-user interacts with this smart contract to send their data access request for particular IoT device. This smart contract sends valid access requests to Aggregator SC which connects the user with the oracle.
- *Reputation SC*. This smart contract computes and records the average reputation scores for all oracles. Reputation scores are sent by the Aggregator SC after each request to access data.
- *Aggregator SC*. This smart contract sends a data request to set of oracles and receives back hashes for the requested data. Then it compares all hashes, reports reputation scores for each involved oracle to Reputation SC, selects highly reputed oracles, generates an access token and send it to both user and selected oracles.
- *Oracle SC*. This SC holds information about registered oracles and its capabilities to support the access to data from the IoT data storage layer for a particular IoT device. The oracle can list also pricing information and availability.
- *Oracles*. These are the data feed sources or online gateways or servers that can access the IoT data and also have interface capability with the blockchain. The system interacts with a pool of oracles from which to retrieve IoT data to prevent an oracle to monopolize the system.
- *IoT Data Storage Layer*. This layer enables hosts to store collected data for a particular IoT device. This layer can include different storage services such as cloud storage services, MQTT brokerage services, and the storage services provisioned by CoAP servers. In the cloud, such services must have features of scalability and elasticity to enable efficient use of cloud resources as workload fluctuates [16], [17].

2) System Interactions and Workflow: The interactions between different components of the proposed system are captured by the sequence diagram illustrated in Figure 3. Some of these interactions take place on-chain and some interactions will be off-chain. Admins are responsible for registration and managing access permissions for end-users to access data from *IoT data storage services* through *IoT Data Access SC*. Users with required permissions send IoT data access request to *IoT Data Access SC* with the required details that include: user Ethereum Address (EA), IoT device EA, number of oracles to retrieve data from, etc. *IoT Data Access SC* verifies user request, and if the user has valid access permissions, the request proceeds as following:

- *IoT Data Access SC* forwards the user request to *Aggregator SC*.
- *Aggregator SC* sends a data request to a pool of oracles.
- *Oracles* find and retrieve requested data from off-chain *IoT data storage services*.
- Each oracle hashes the retrieved data and sends it back to *Aggregator SC*.
- Aggregator SC compares all received data hashes from oracles, finds 51% agreement on the returned hashes, and reports reputation score for each oracle to *Reputation SC*.
- *Reputation SC* updates reputation score for all participating oracles, selects an oracle based on highest average reputation scores, and returns the address of the selected oracle to the *Aggregator SC*.
- *Aggregator SC* generates an access token and sends the token to the selected oracle and the end-user.
- *End-user* uses the generated access token to access IoT Data from the selected off-chain oracle.

IV. Implementation

In this section, we highlight key implementation aspects of three main smart contracts: *IoT Data Access SC*, *Aggregator SC*, and *Reputation SC*. The smart contracts were implemented and tested using Remix that is a browser-based compiler and IDE that offers rich features allowing testing and debugging smart contracts prior to deploying them [18]. The full code of our smart contracts in Solidity language is made available at GitHub. We discuss the main functionalities of these smart contracts and primary related on-chain and off-chain interactions.

A. IoT Data Access SC

This contract manages access to the data of the registered IoT device(s). Each device or group of devices has one smart contract. This depends on the owner of the contract. If the owner, for example, is an organization then all the devices in that organization are managed by one single contract. In addition, these devices can be managed by one or multiple admins. Admins can be added or deleted by other admins. The owner of the IoT device(s) creates a contract through the constructor of the IoT Data Access SC to map the users to their IoT devices and grant them access to their data. The contract has four primary functions with associated events:

1) Add/delete Admins: The admin is the only modifier who can add or delete another admin. However, the owner of the contract cannot be deleted as there must be at least one admin per contract.
2) Add/delete devices: Registered admins can add devices or delete them.
3) Add/delete users. Registered admins can add users and map the user Ethereum Address (EA) to the set of devices the user can access. Admins also decide on the type of access to the IoT data: read, write or execute. Admins can also delete the users to prevent them from accessing the IoT data in the future.

Algorithm 1 Request access to IoT data

```
Input: $EA\_device, numberOfOracles$
Output: $NewToken$
State: $Token[\,], UID$
                ▷ struct Token : UID, EA_User, EA_device, numberOfOracles
Require: $DeviceExists, IsOdd(numberOfOracles), Is(numberOfOracles > 3)$
if user is authenticate then
    $UID \Leftarrow keccak256(EA\_User, EA\_device, block.timestamp, numberOfOracles)$
    $NewToken \Leftarrow (UID, EA\_User, EA\_device, numberOfOracles)$
    emit Authenticated event
    emit newTokenCreated event
    Send newToken to aggregator SC
else
    emit NotAuthenticated event
    no token is created
```

4) Request access to IoT Data. This is the main function that can be called by anyone in the blockchain network to request access to the IoT data. The caller of the function should provide the EA of the IoT device and the number of oracles he would like to retrieve data from. Algorithm 1 illustrates how the function works to check if the user is authorized to access the data of an IoT device. Once the user issued a request, the SC will check if the device exists, and then checks if the user can access its IoT data. In the case the user access is granted, the contract will create a smart token following the proposed solution in [10]. The token contains the following information: (1) Unique Identification (UID), which is generated from hashing the user EA, device EA, block time-stamp, and number of oracles using *keccak256* hashing algorithm, (2) User EA , (3) Device EA , and (4) number of oracles. The created token is then sent to the Aggregator SC that is responsible for the interaction with the oracles and connect the user with the appropriate oracles. A selection of a minimum of 3 oracles as well as odd number of oracles are required in order to generate the token. This is to ensure oracle consensus of more than 51%. Choosing an even number of oracles may lead to a tie. With a tie, the Aggregator SC will be unable to reach a decision about the results reported from the oracles.

B. Aggregator SC

This smart contract coordinates the flow of data requests between the different components of the system: IoT Data Access SC, Oracles, Reputation SC and users. This smart contract keeps track of oracles and map these oracles to the list of IoT devices which they have access to their data storage layer. If user granted access to IoT data, user request is forwarded by IoT Data Access SC to Aggregator SC. The

Algorithm 2 Determine reputation scores to oracles

```
Modifier: $aggregators$
Input: $EA_{reputationSC}, EA_{oracles}, dataHashes$
Output: $EA_{correctOracles}, score_{correctOracles}$
State: $oracles_{data}, hashData_agreement, numOfMatches, counter$
$oracles_{data} \Leftrightarrow dataHashes$

$hashData_{agreement} \Leftrightarrow counter$
$numOfMatches$ = $EA_{oracles}.length/2$

for every data in dataHashes do
    if $hashData_{agreement}[data] \neq 0$ then
        $hashData_{agreement}[data] \leftarrow hashData_{agreement}$ [data]++
    else
        $hashData_{agreement}[data] \leftarrow 1)$
for every counter in $hashData_{agreement}$ do
    if $hashData_{agreement}$ [counter]> $numOfMatches$ then
        $EA_{correctOracles} \leftarrow oracles_{data}$[counter]
        $score_{correctOracles} \leftarrow 100)$
        break loop
send $EA_{correctOracles}$ and $score_{correctOracles}$ to reputation SC
```

Aggregator SC receives the request, finds pool of oracles to fulfill user data request, sends the request to oracles, collects and evaluates oracles' responses, reports oracles' reputation to Reputation SC, and finally generates and sends access token back to the user along with selected oracle address, and sends same token and user address to the selected oracle. The main functions of this smart contract are:

1) addOracle/addDeviceToOracle: Contract owner is the only modifier who can add new oracle and add/assign new IoT devices to oracles.
2) sendDataRequest: This function handles user data requests, by finding the pool of oracles (user defines the number of oracles, and we restrict the number of oracles to a minimum of 3 oracles) that can retrieve the requested IoT data and send the request to these oracles.
3) oracleResponse: This function will be called by oracles to report their responses and hashes of the result for the requested IoT data.
4) reportReputationScore: This function assesses the oracle responses and reports a binary reputation score to the reputation SC based on 51% agreement. If more than 51% of oracles return the same result, then all these oracles will get 100 reputation scores and any other oracle that returns different result will receive 0 reputation score. Algorithm 2 shows how reputation scores are determined.

Algorithm 3 Select a winning oracle

```
Input: $EA_{oracles}$
Output: $EA_{winningOracle}$
State:$oracles$
$Max_{avgscores} \leftarrow 0$
$Current_{avgscores} \leftarrow 0$
for every $EA$ in $EA_{oracles}$ do
    $Current_{avgscores} \leftarrow a.avrg$
    if $Current_{avgscores} >= Max_{avgscores}$ then
        $Max_{avgscores} \leftarrow Current_{avgscores}$
        $EA_{winningOracle} \leftarrow a$
return $EA_{winningOracle}$
```

C. Reputation SC

The main objective of this smart contract is to keep track, compute, and manage reputations for oracles. Aggregator SC reports to this SC reputation scores for oracles. This smart contract computes the overall average reputation score for each oracle. The main functions of this smart contract can be summarized as follows:

1) addAggregator. The owner is the only modifier who can register and add Aggregator SCs which are eligible to report reputation scores about oracles.
2) reportReputationScore: The Aggregator SC is the only modifier which can report oracles' reputation scores. This function will update the average reputation score for each oracle. Algorithm 3 illustrates how this function works. This function includes computation to calculate the average reputation score. The classical way for calculating the average of *n* scores is done by the following:

$$Avg = \frac{\sum_{k=1}^{c} P_k}{c} \rightarrow cost = 3 * c + 5 \quad (1)$$

In Ethereum, the cost of ADD/SUB operation is 3 Gas, and the cost of MUL/DIV operation is 5 Gas. To avoid recalculating the average for each call to the function we used the following equation (2), which is derived from the previous one:

$$Avg = \frac{(c * Avg) + P}{c + 1} \rightarrow cost = (2 * 5) + (2 * 3) = 16 \quad (2)$$

Equation (2) is more cost efficient in terms of Gas consumption, as the computation cost will remain constant. Unlike Equation (2), the cost of Equation (1) will increase 3 Gas per transaction or function call.
3) selectWinningOracle: This function takes as an input list of oracle EA and it returns as an output the address of the oracle with highest average reputation score. Algorithm 4 illustrates how this function works.
4) getOracleReputation: This function returns average reputation scores for a given oracle EA.

V. Evaluation

In this section, we evaluate our proposed solutions and developed algorithms in terms of cost analysis pertaining to Ethereum Gas consumption to carry out smart contract transactions. We also analyze the computational complexities of the different smart contracts. Moreover, we perform security analysis and compare qualitatively our solution with existing similar systems reported in the literature.

A. Cost Analysis

All of the on-chain computations, data storage, and transactions consume Gas which is the price unit in Ethereum. We calculated the execution cost of all functions in the smart contracts. Table I shows the cost of the actual function execution as well as the storage cost. The static cost analysis of functions follows the official documentation [19] of Ethereum blockchain and costs of functions are calculated on the basis of Gas prices offered by Ethereum network on May 15, 2019, as stated by Eth Gas Station [20].

The functions listed in Table I are executed by the smart contracts: *AccessControlSC*, *AggregatorSC*, and *ReputationSC*. The overall cost of the functions is minimal as none of them increased beyond $0.1. The functions for adding and deleting users, devices, admins, oracles, and aggregators cost the least in all smart contracts and no impact was seen on the overall cost of using the system. It was noted that functions such as *oracleResponse()*, *query()*, *reply()*, *selectWinningOracle()*, and *getReputationScore()* contain G_{call} operations in their execution state models, therefore, it incur comparatively more cost which range between $0.0016 and $0.0021. On the other hand, *requestUserToAccessDevice()*, *sendDataRequest()*, and *reportReputationScore()* functions cost the most because it contains both G_{create} and G_{call} in their state execution models. It should be noted that these operations enable to create identifiers from other contracts or call functions with data from other contracts, therefore, inter-contract communication incurs more cost as compared to intra-contract communication and data processing. Generally, IoT users can access the data by paying the default Gas price from their wallets. However, Ethereum defines three optimal price structures for its network users namely 1) safe-low, 2) average and 3) fastest. The safe-low Gas price defines the cheapest possible execution cost for a successful transaction. Alternately, the fastest Gas price defines the best possible maximum value to attract the miners. The average Gas cost represents the average cost of execution per GWei on the Ethereum blockchain. The Gas cost of all functions is presented in Table I. Considering the low execution cost of functions, it was observed that the difference between safe-low and fastest Gas prices remained minimal in all cases. Based on the cost analysis, Figure 4 depicts the overall cost consumption trends of our three smart contracts.

Algorithm 4 Report Reputation Score

```
**Modifier:** $aggregators$
**Input:** $EA_{oracles}, Oracle_{scores}$
**State:** $oracles, aggregators$
**for** every $EA$ in $EA_{oracles}$ **do**
    **if** $EA$ in $oracles$ **then**
        $Avg \leftarrow ((c \times avg) + Oracle_{scores})/(c+1)$
        $c \leftarrow (c+1)$
    **else**
        add $EA$ to $oracles$
        $avg \leftarrow Oracle_{scores}$
        $c \leftarrow 1$
```

B. Complexity Analysis

The computational complexity of all functions is presented in Table I considering the worst-case static analysis. This complexity analysis is more useful for EVM operators because they need to ensure high availability of required resources on the network. It can be also beneficial to end-users and oracle providers in determining the time consumption of smart contract functions. The results show that functions with repetitive operations tend to be more time consuming; however, restricting the 'n' values (which represents the number of devices, users, oracles, and storage services) could significantly improve the transaction time. The 'c' values represent the overall computational cost of non-repetitive operations. Figure 5 shows the overall computational cost of smart contracts while performing on-chain and off-chain operations. Overall, the functions requesting IoT data access and reporting reputation scores consume more time, thereby incurring more Gas directly proportional to their computational time complexity.

C. Qualitative Comparison

When compared with the existing state-of-the-art systems (as exhibited in Table II), our proposed system is more superior in providing support to trusted oracles and maintaining their on-chain reputation. Existing systems enable support to either centralized data storage services [6], [7], [8] or they make use of hash tables [9] for distributed storage across the underlying peer to peer network. On the other hand, our proposed system enables an extra storage layer whereby any data storage device or system can register and provide their services on the blockchain network, allowing for a heterogeneous and decentralized storage services.

D. Security Analysis

In this section, we highlight best practices to develop and write smart contracts so that they can withstand known attacks. We also discuss and analyze the security of our overall solution. As best practices, smart contract developers need to ensure that smart contract code should be free of security vulnerabilities and bugs. We used popular tool Oyente and Securify to check for known vulnerabilities and bugs. For example, in our code, we used **low-level call** methods with external calls which return 'false' flag in case of any exception in the smart contract code or detection of malicious activity in the called smart contracts. We also used **pull-based external calls** to ensure maximum safety in the smart contracts. Finally, we **explicitly** labeled functions and state variables to clearly define their accessibility on the Ethereum blockchain.

A major risk of calling external contracts is their ability to control the execution of smart contract code and perform

Table I
Execution Cost and Computational Complexities of Smart Contract Functions

Function Caller	Function Name	Execution Cost (GWei)	Cost ($) - Cheap	Cost ($) - Avg	Cost ($) - Fastest	Computational Complexity
AcessControlSC	*addDevice*	6	0.00002	0.00002	0.00002	2c
AcessControlSC	*addAdmin*	6	0.00002	0.00002	0.00002	2c
AcessControlSC	*addUserDeviceMapping*	28	0.00007	0.00007	0.00007	10n+c
AcessControlSC	*delAdmin*	16	0.00005	0.00005	0.00005	5n+c
AcessControlSC	*delUser*	5	0.00002	0.00002	0.00002	2c
AcessControlSC	*requestUserToAccessDevice*	32755	0.07633	0.08247	0.09081	15n+10c
AcessControlSC	*addOracle*	6	0.00002	0.00002	0.00002	2c
AcessControlSC	*sendDataRequest*	32728	0.07495	0.08244	0.09475	12n+3c
AcessControlSC	*oracleResponse*	718	0.00165	0.00180	0.00210	8c
AcessControlSC	*reportReputationScore*	32757	0.07502	0.08249	0.09743	31n+4c
AggregatorSC	*query*	706	0.00163	0.00178	0.00208	3c
AggregatorSC	*reply*	703	0.00160	0.00178	0.00214	2c
AggregatorSC	*addAggregator*	6	0.00002	0.00002	0.00002	2c
AggregatorSC	*reportReputationScore*	35	0.00007	0.00009	0.00011	11n+c
AggregatorSC	*selectWinningOracle*	720	0.00165	0.00180	0.00210	7n+2c
ReputationSC	*getOracleReputation*	700	0.00160	0.00176	0.00208	c

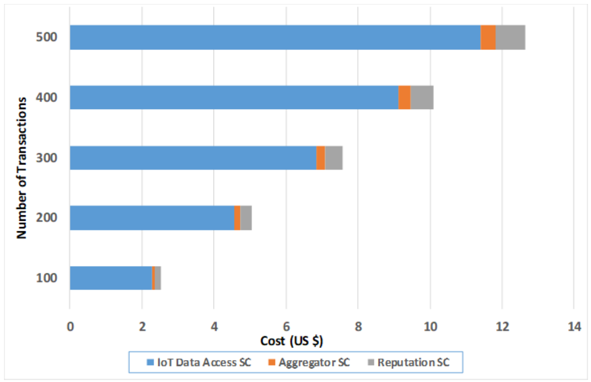

Figure 4. Cost vs. Transaction Volume in implemented Smart Contracts

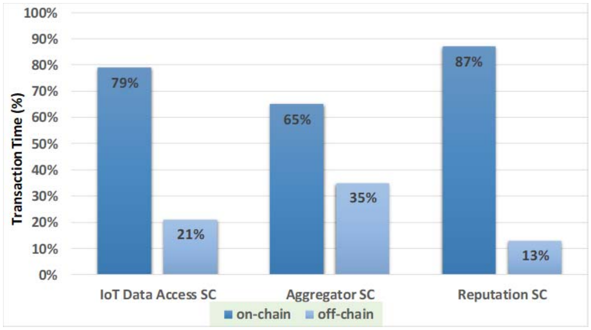

Figure 5. On-chain and Off-chain Computational Time Consumption

Table II
Comparison with state-of-the-art

Features	[6]	[7]	[8]	[9]	our work
Distributed access control	✓	✓	✓	✓	✓
Policy management (static-scenario)	✓	✓	✗	✗	✓
Policy management (Dynamic-scenario)	✗	✓	✗	✗	✓
Support to Homogeneous Storage	✓	✓	✓	✓	✓
Support to Heterogeneous Storage	✗	✗	✗	✓	✓
Trusted Oracles	✗	✗	✗	✗	✓
Secure Communication	N/A	N/A	✓	N/A	✓
Distributed Data Management	✓	✓	✓	✓	✓

unexpected manipulation in transactional data. These types of bugs in a smart contract code could exist in multiple forms. The attackers can execute the malicious code and reenter in the smart contract by recursively calling the value transfer function and performing multiple transactions. Considering the **reentry attack**, we used *send()* function instead of *call.value()* therefore our smart contracts prevent execution of any external code. The **cross-function race conditions** are another form of attack where a potential attacker can replicate the attack from two or more different functions having shared state variables. The severity of attack could be disastrous when cross-function race conditions span over multiple contracts. Therefore, we avoided cross-function and cross-contract race conditions by minimizing external calls and these calls are made only when the functions completely traversed through the internal state models of the functions. Using *mutual exclusion* strategies could be another possible solution which we will explore in our future research works.

The use of blockchain to control the access to the IoT data relieves the use of the expensive key distribution of Public Key Infrastructure (PKI) since EAs are used to identify different nodes in the network. According to [10], EAs are uniquely assigned to any participant or node added to the blockchain network with almost no collision. In addition, the addresses come with asymmetric public key pairs that can be used in the off-chain secure SSL connection between the oracle and the user. Finally, the use of nonce (i.e. block time-stamp) and UID makes the system secure against **Man-In-The-Middle (MITM) attacks** as the attacker won't be able to correctly sign the message even if the attacker changes the user EA and public key with his own.

In the case of verifying that the oracle node is the right node that the user should authenticate itself to, the blockchain name service (BNS) like ENS [21] could be used. BNS works in the same manner as DNS by replacing the long hashed addresses (like EA) to human-readable names in a secure and decentralized way. ENS does not suffer from the security issues DNS system has as it is built on smart contracts on the Ethereum blockchain [21].

Oracles are the source of information in our proposed system. These oracles are third-party services. Oracles need to be trustworthy to ensure the security of the overall system. One of the main challenges with oracles is **centralization**, where oracles represent single scores of failure in the Blockchain network. We addressed this security issue in our proposed solution by introducing a reputation registry for oracles.

Moreover, we performed byte-code level debugging analysis of proposed smart contracts using the semantic framework proposed in [22]. We analyzed the smart contracts in terms of **call integrity** and **atomicity**. Ethereum has the history of DAO bug which resulted in the loss of 60 million US dollars from Ethereum's blockchain network. The DAO bug allowed the **reentry of attackers** on the blockchain who redirected payments to specific EAs. The semantic analysis shows that our smart contracts comply with call integrity property by not allowing any intermediate calls during mining and verification and atomicity by ensuring complete transaction execution and value transfer before processing the next transaction.

VI. Conclusion

In this paper, we have proposed a decentralized access control for IoT data using blockchain and trusted oracles. The proposed solution employed smart contracts to achieve decentralized access control to allow end-users to access remotely stored IoT data. Trusted oracles were used as gateways to provide interoperability among entities hosting the IoT data and the blockchain network. The proposed solution also used smart contracts to provide a decentralized reputation service for oracles. The reputation service can be used by end-users DApps to select oracles. We used a web-based Remix IDE environment to implement and test the different functionalities of the proposed solution. The paper presented algorithms of key smart contract functions, and also discussed best practices for smart contract development, along with cost, computation, and security analysis. As a future work, we are in the process of implementing a fully functional system consisting of IoT data stored in the cloud and building software for the trusted oracles that can access the data and part of the public Ethereum blockchain network. We are also developing front-end DApps for the participants to interact with the Ethereum smart contracts.

Acknowledgment

This publication is based upon work supported by the Khalifa University of Science and Technology under Award No. CIRA-2019-001.

References

[1] M. Rehman, C. Liew, T. Wah, J. Shuja, B. Daghighi *et al.*, "Mining personal data using smartphones and wearable devices: A survey," *Sensors*, vol. 15, no. 2, pp. 4430–4469, 2015.

[2] K. Gyarmathy, "5 iot statistics you need to know in 2019," *[Online]. Available: https://www.vxchnge.com/blog/iot-statistics. [Accessed 25 March 2019]*, 2018.

[3] T. Maddox, "The dark side of wearables: How they're secretly jeopardizing your security and privacy," *[Online]. Available: https://www.techrepublic.com/article/the-dark-side-of-wearables-how-theyre-secretly-jeopardizing-your-security-and-privacy/. [Accessed 24 March 2019], year=2015.*

[4] M. A. Khan and K. Salah, "Iot security: Review, blockchain solutions, and open challenges," *Future Generation Computer Systems*, vol. 82, pp. 395–411, 2018.

[5] L. Willems, "How to create a dapp using truffle, oraclize, ethereum-bridge and webpack," *[Online]. Available: https://medium.com/coinmonks/how-to-create-a-dapp-using-truffle-oraclize-ethereum-bridge-and-webpack-9cb84b8f6bcb. [Accessed 18 March 2019], year=2018.*

[6] O. Novo, "Scalable access management in iot using blockchain: a performance evaluation," *IEEE Internet of Things Journal*, 2018.

[7] D. Hwang, J. Choi, and K.-H. Kim, "Dynamic access control scheme for iot devices using blockchain," in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2018, pp. 713–715.

[8] K. Salah, M. H. U. Rehman, N. Nizamuddin, and A. Al-Fuqaha, "Blockchain for ai: review and open research challenges," *IEEE Access*, vol. 7, pp. 10 127–10 149, 2019.

[9] 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*. ACM, 2017, pp. 45–50.

[10] R. Almadhoun, M. Kadadha, M. Alhemeiri, M. Alshehhi, and K. Salah, "A user authentication scheme of iot devices using blockchain-enabled fog nodes," in *2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)*. IEEE, 2018, pp. 1–8.

[11] J. Paillisse, J. Subira, A. Lopez, A. Rodriguez-Natal, V. Ermagan, F. Maino, and A. Cabellos, "Distributed access control with blockchain," *arXiv preprint arXiv:1901.03568*, 2019.

[12] A. Suliman, Z. Husain, M. Abououf, M. Alblooshi, and K. Salah, "Monetization of iot data using smart contracts," *IET Networks*, vol. 8, no. 1, pp. 32–37, 2018.

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

[14] I.-C. Lin and T.-C. Liao, "A survey of blockchain security issues and challenges." *IJ Network Security*, vol. 19, no. 5, pp. 653–659, 2017.

[15] J. Adler, R. Berryhill, A. Veneris, Z. Poulos, N. Veira, and A. Kastania, "Astraea: A decentralized blockchain oracle," *arXiv preprint arXiv:1808.00528*, 2018.

[16] F. Al-Haidari, M. Sqalli, and K. Salah, "Impact of cpu utilization thresholds and scaling size on autoscaling cloud resources," in *Proceedings of the 2013 IEEE International Conference on Cloud Computing Technology and Science-Volume 02*. IEEE Computer Society, 2013, pp. 256–261.

[17] P. Calyam, S. Rajagopalan, S. Seetharam, A. Selvadhurai, K. Salah, and R. Ramnath, "Vdc-analyst: Design and verification of virtual desktop cloud resource allocations," *Computer Networks*, vol. 68, pp. 110–122, 2014.

[18] Ethereum, "Remix ide," *[Online]. Available: http://remix.ethereum.org. [Accessed 20 February 2019].*, 2019.

[19] G. Wood, "Ethereum yellow paper: a formal specification of ethereum, a programmable blockchain," pp. 45–50, 2017.

[20] Ethereum, "Ethereum gas station, comsumer-oriented metrics for the ethereum gas market," *[Online]. Available: https://ethgasstation.info/ [Accessed 15 May 2019]*, 2019.

[21] T. ENS, "Ethereum name service," *[Online]. Available: https://ens.domains/. [Accessed 20 February 2019].*, 2019.

[22] I. Grishchenko, M. Maffei, and C. Schneidewind, "A semantic framework for the security analysis of ethereum smart contracts," in *International Conference on Principles of Security and Trust*. Springer, 2018, pp. 243–269.