Decentralized IoT Data Management Using BlockChain and Trusted Execution Environment

Gbadebo Ayoade*, Vishal Karande*, Latifur Khan*, and Kevin Hamlen *

*Department of Computer Science
University of Texas at Dallas, Richardson, Texas 75080
Email: (gbadebo.ayoade,vishal.karande,lkhan, hamlen)@utdallas.edu

Abstract

Due to the centralization of authority in the management of data generated by IoT devices, there is a lack of transparency in how user data is being shared among third party entities. With the popularity of adoption of blockchain technology, which provide decentralized management of assets such as currency as seen in Bitcoin, we propose a decentralized system of data management for IoT devices where all data access permission is enforced using smart contracts and the audit trail of data access is stored in the blockchain. With smart contracts applications, multiple parties can specify rules to govern their interactions which is independently enforced in the blockchain without the need for a centralized system. We provide a framework that store the hash of the data in the blockchain and store the raw data in a secure storage platform using trusted execution environment (TEE). In particular, we consider Intel SGX as a part of TEE that ensure data security and privacy for sensitive part of the application (code and data).

***Keywords*-IoT; Blockchain; SGX; Privacy**

I. Introduction

With the advancement in embedded processors, actuators, sensors and communication systems, everyday devices are retrofitted with capabilities to communicate, compute and complete automated tasks [1], [2]. For instance, many of our everyday appliances have been retrofitted with capabilities to connect to the Internet [3]. Such IoT devices include smart pacemakers, heart rate monitors, smart refrigerators, smart coffee makers, smart television, smart home assistants, and smart door locks. By equipping these devices with computational and communication capabilities, these devices collect and transmit large amount of privacy related data [4]. For example, IoT devices such as smart cameras, smart health monitoring devices [5] such as heart rate monitors,glucose level monitors can reveal privacy information about the users .

Due to the limited processing capabilities of IoT devices [6], [7], IoT devices usually leverage externally controlled third party service providers to perform additional data processing. By transmitting sensitive user data to third party services providers [1], users are forced to trust service providers to enforce data protection and provide data privacy guarantee. Unfortunately, service providers often violate data privacy policies by using data collected from users for unauthorized purposes [8]. This undue advantage by service providers is based on centralized architecture where trust in a third party system as a central authority is required to manage user data. In order to eliminate these imbalance in data access policy enforcement between service providers and users, we propose a system of decentralized data management using decentralized asset management system based on Blockchain [9] and smart contract technology [10].

With the advent of decentralized asset management systems as seen in finance sector which leverage blockchain technology such as seen in Bitcoin [9], electronic fund transfer can occur without the need for centralized electronic fund management system. With this technology, money transfer can occur across international boundaries without the bureaucracy of of centralized authorities. Due to the decentralized nature of blockchain technology, proposed applications [10] in various fields include automated insurance management, supply chain management, decentralized commercial data storage as seen in Filecoin [11]. For instance, Slock It [12] uses blockchain to provide automated device sharing platform for IoT devices such as smart locks.

By leveraging this decentralized architecture, we propose a system that limits the authority of centralized

data management systems. Blockchain technology [9] and smart contracts [10] allow decentralized management of data among *untrusted* parties called miners. Blockchain [9] is a distributed ledger where transaction state integrity is enforced by distributed consensus among decentralized untrusted parties. To enforce the integrity of the blockchain, each current block generated by the miners must contain a hash of the previous block in the blockchain Figure 2, making it difficult to modify the transactions recorded in the blockchain.

Smart contracts [13] are autonomous applications that run within the blockchain. With smart contracts, we provide a system where *rules* that govern interactions among interested parties is enforced autonomously in the blockchain network without a centralized trust. By leveraging this capability, we can equip the users with the capacity to control how their data is accessed and used since smart contract provides them with equal data management privilege. Furthermore, smart contracts executes in isolated virtual machines on the miners infrastructure. By using isolated virtual machines to run these smart contracts, miners cannot modify application outcomes. With smart contract and blockchain, we can provide data access audit system to track data usage among the interested parties leading to proper data access accountability.

All data stored in the blockchain has to be public for the miners to be able to verify transactions [14]. Our proposed system overcomes these challenge by storing the hash of the encrypted data in the blockchain, while the main data is encrypted and stored using trusted computing. By leveraging trusted computing, we can verify the integrity of the system used in our data storage. In our case, we use trusted computing as implemented by Intel SGX architecture.

By leveraging trusted execution environment based on Intel SGX, we provide data protection from unauthorized access from powerful adversaries. SGX offers hardware level protection of user data by enforcing process isolation by executing the programs in secure enclaves and protecting the enclave's memory pages by the CPU hardware. These secure containers called enclaves are protected from operating system, other processes and hypervisor processes [15].

In this work, we make the following contribution.

- We leverage blockchain platform to provide decentralized IoT data access management.
- We leverage smart contracts to provide equal data access management privilege among IoT users and IoT service providers.
- We provide data storage using trusted execution environment(Intel SGX) for secure data storage.
- We provide a full system implementation on real blockchain platform using Ethereum smart contracts.

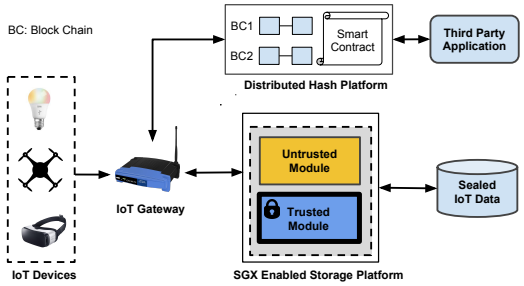

Figure 1: A Simplified Architecture of IoTSmartContract

The rest of the paper is organized as follows. §II provides background on Blockchain, SGX and IoT system. §III discusses the scope, case for blockchain and SGX and challenges and solutions encountered in deploying SGX based system. §IV provides the architecture of our system. In §V, we describe our implementation approach and §VI provides the evaluation of our approach. §VII and §VIII provide discussion and related work respectively. Finally, §IX concludes.

II. Background

A. Overview of Architecture

In this section, we discuss a brief overview of our system components as shown in Figure 1. To provide decentralized management of data generated by IoT devices, we store the hash of the encrypted data generated in the blockchain and then store the data itself in an SGX enabled storage system. As a result, the blockchain manages the data access policy through the smart contract.

To access data, third party users will request permission to access data from the blockchain by utilizing the smart contract API. If request is granted, the hash of the data is returned and used to retrieve data from the SGX platform. Before the SGX platform retrieves that data from secure storage, it will independently recheck the blockchain for access permission before returning the data needed to the third party user. The intuition for these two step check is to ensure all access permission policy and authority is managed by the smart contract executing in the blockchain. The access check does not incur much overhead since access check is a read operation which has a fast execution time on the blockchain as we will later show in our evaluation section. In this section, we provide more background on the components of our IoTSmartContract system.

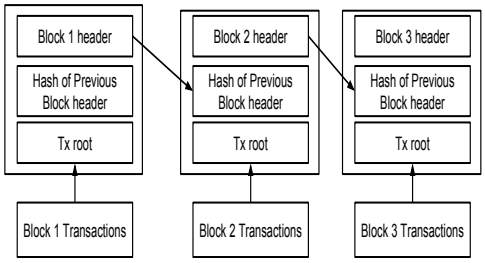

Figure 2: A BlockChain Data Structure

B. Internet of Things

With the increase in computational and communication capabilities and technological advancement in device miniaturization, every day devices are granted capabilities to sense and react to the environment through the use of sensors and actuators. A typical IoT architecture comprises of devices, sensors, actuators, IoT Hubs, IoT Gateway and a cloud service provider.IoT devices are devices with capability to sense and collect data which can be transmitted on a connected network for storage or further processing. IoT devices includes light bulbs, heart rate monitors, smart cameras and many more. With the IoT hub, different devices with disparate communication protocol such zigbee or bluetooth can connect to the IoT network. IoT networks includes IoT gateway which helps to provide data aggregation on the client network. To process the huge amount of data transmitted by the IoT devices, cloud services are used to store and further process the data.

C. BlockChain

Blockchain is a distributed ledger where the state of its transactions is maintained by a distributed consensus among untrusted entities without the need for a centralized trusted third party authority. These decentralized entities are called *miners* [16], [9]. By providing a proof of work, the miners bundle confirmed transaction in blocks by generating a hash of the current block which includes the hash previous block as seen Figure 2. These proof of work generation requires high computational CPU power, therefore protecting the blockchain from adversarial attacks.

The blockchain can store data and perform computations that can be executed by these decentralized entities to determine the state of the blockchain in an autonomous manner. These autonomous computations are called *smart contracts*. By leveraging smart contracts, we provide a system where *decentralized data access policy* control is enforced without relying on third party service providers, therefore ensuring continuous service delivery for IoT system users. We implemented the smart contract using Ethereum blockchain platform.

The *Ethereum* [10] smart contract is an implementation of the smart contract with Turing complete computation. The Ethereum smart contract is deployed in the blockchain and can be executed by the miners to determine the state of the program. By generating blocks, the miners can autonomously ensure that the state of integrity of the contract program.

In order to allow miners to run a deployed smart contract, the contract owner will pay the miners some fee called *Ethereum gas*. The higher the gas paid, the faster the speed of getting the contract to execute and generate confirmations on the blockchain. Because the smart contracts also store data, contract owners will need to provide gas for storage on the blockchain. In our case, we limited the data stored on the blockchain by storing only the the hash of the data and then encrypting the data and storing on another system.

To interact with a smart contract, each smart contract has a unique address in the blockchain. The address can be used to retrieve the contract and then get the ABI (Abstract Binary interface) which provides the API of the contract. By getting the smart contract API, a user can execute the smart contract API to perform some computation.

D. Trusted Execution Environment

Recent advancements in embedded hardware technology to support trusted execution environment (TEE) (e.g., TPM , ARM Trust Zone [17], AMD SVM [18]), Intel SGX [19]) allow service providers to ensure confidentiality and integrity of data and computations by protecting code and data within a secure region of computation.

Intel SGX is a trusted computing architecture introduced in the new Intel Skylake processors. By providing a new set of instructions which extends the X86 and X86_64 architectures, user level applications can provide confidentiality and integrity without the trust of the underlying Operating System. With these instructions, application developers can create a secure and isolated containers called *enclave* to protect security sensitive computations. In particular, the memory content of an enclave is stored inside a hardware protected memory region called as Enclave Page Cache (EPC). By leveraging the Memory Encryption Engine (MEE), all EPC pages are encrypted and any access to them is restricted by the hardware. Therefore, with SGX, applications can protect sensitive and secret data and computations from attacks from high privilege applications like the Operating system, hypervisors and System Management Mode.

III. Overview

A. Scope and Assumptions

The scope of this paper considers decentralization of data access management using blockchain and data privacy protection using Intel SGX. The main challenge is how to establish trust between IoT service providers and the users of IoT services. By leveraging smart contracts, we provide a data access management system where users have equal privilege in controlling how their data is shared or used. With smart contracts, we can specify data access rules that are autonomously enforced by untrusted third party entities on the blockchain network. For our platform, we assume all data is encrypted before transmission and all key exchange is performed using asymmetric cryptographic protocols [20]. In this paper we do not consider replay attacks and denial of service attacks.

B. Threat Model

For our threat model, we consider the IoT data management service providers as untrusted entities since they have full control over user data, which give them undue advantage in how they use data or share user data with other third party entities.

Furthermore, we consider all third party users who request access to data to be untrusted. We assume all non data owner may leak data or use it for unauthorized purposes such as user's email for direct marketing.

We consider adversaries that seek to compromise the data storage cloud services by obtaining root privilege access to low level system resources such as memory, hard drives and Input/Output systems. These attackers employ techniques that compromise highly privilege applications such as Operating System and hyper-visors.

C. The Case of Using Blockchain for IoT data management

Decentralized Trust. As users become more knowledgeable of data privacy leakage and its consequences, users may demand more control over how there data is being used. By leveraging blockchain, IoT vendors and service providers can provide services that users can trust since the data management system is done in publicly verifiable smart contracts program that run in the blockchain.

SmartContract Enforced Accountability. With smart contracts, we can provide autonomous applications that enforce interaction rules among the system users without the need of centralized authority. Smart contracts allow individual entities with varied interest to generate rules

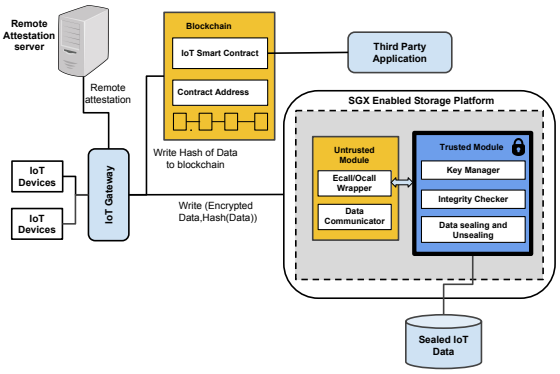

Figure 3: A IoTSmartContract Architecture

that satisfy each participants interest. The rules are then programmed into smart contracts which is then enforced by the miners by independently verifying the state of the contract. For example, in centralized access policy management such Smartthings [1], [21], if a user grants access or revoke access to their data, the user has to trust the third party service provider to comply and enforce his data restriction. With smart contracts, the users have equal privilege on how the policy is enforced since the policy enforcement is done by the miners on the blockchain network.

Audit Trail Enforcement. By leveraging immutability of blockchain ledger [9], we can provide immutable data access history of users' data. Since all entries in the blockchain is cryptographically linked to previous blocks generated on the blockchain, it is difficult for malicious attackers to modify the blockchain entries.

IV. Architecture

As shown in Figure 3, IoTSmartContract consists of three main components which includes the IoT client network, the smart contract and the secure SGX module. The Client IoT network consist of all the IoT devices, the IoT gateway which connects the devices to the external network.

A. Smart Contract Component

As shown in algorithm 1, The Smart contract provides the decentralized access control policy to user data in form of Ethereum smart contract that executes in the blockchain. As a result of the limited data storage and fees required to store data in the smart contract, the smart contract only stores the hash of the data in the blockchain. The main data is encrypted and stored on the SGX module. The smart contract includes the user registration module, device

Algorithm 1: Smart Contract Pseudo-code

```
1: HashMap deviceRegistry(key:ownerAddress,value:List[DeviceIds])
2: HashMap deviceData(key:(ownerAddress,deviceId), value:List[DataHash])
3: HashMap
   DataAccessRegistry(key:(ownerAddresss,thirdpartyAddress,deviceId),value:
   bool isAllowed)
4: function REGISTERDEVICE(ownerAddress,deviceID)
5:     InsertToHashMap(deviceRegistry)
6: end function
7: function WRITEDATA(ownerAddress,deviceID,Data)
8:     if owner == ownerAddress
9:     deviceData[owner,deviceID].List.InsertData(hash(Data))
10: end function
11: function READDATA(ownerAddress,thirdPartyAddress,deviceID)
12:     if DataAccessRegistry(thirdPartyAddress) == true
13:     return deviceData[hash(ownerAddress,deviceID])
14: end function
15: function GRANTACCESS(ownerAddress,thirdPartyAddress,deviceID)
16:     if owner == ownerAddress
17:     DataAccessRegistry[hash(ownerAddress,thirdPartyAddress,deviceID)] =
   true
18: end function
19: function REVOKEACCESS(ownerAddress,thirdPartyAddress,deviceID)
20:     if owner == ownerAddress
21:     DataAccessRegistry[hash(ownerAddress,thirdPartyAddress,deviceID)]
   =false
22: end function
```

registration, read and policy access module for hash data storage.

User Registration. This module leverages the user registration system on Ethereum network. Each user joins the Ethereum network by generating a public private key pair which uniquely identifies the user. The private key can then be used to interact with the smart contract to perform functions such as device registration and data access.

Device Registration. Each authenticated user can register their IoT devices by providing the identifier for the device. In the smart contract, we provide a hash map that maps the devices owned by a user to the owners address on the blockchain as denoted *mapping (address = list of owners deviceids)*

Data Write Access Policy. For a device to write data to the blockchain, the device will provide the owners address and the device id with the data to be written. By using the combination of the owner address and the device id as the key in a hash map, we can uniquely store all data that corresponds to all devices separately as denoted *((owner_address,device id) = list of device data)*. The value of the hash map is a list of hashes of the data written by the device. Before the smart contract allows data to be written to the contract, the smart contract will check if the owner address correspond to the device ID, so as to ensure only a device owner can execute write operation.

Device Data Read Access Policy. For data access, a third party user who needs access to a device data from another user will request for permission to read the data. The requesting user will provide the address of the owner of the device and the device ID of the device. A hash map that contains the device owner and address and the device id as key with the list of the third party users as values is maintained within the smart contract. This is denoted as *((owner,device id,third party user address) = bool access)*. Before access is granted to the data, this hash map is checked to see if a requesting user can access the data by ensuring only registered third party users can access the device data.

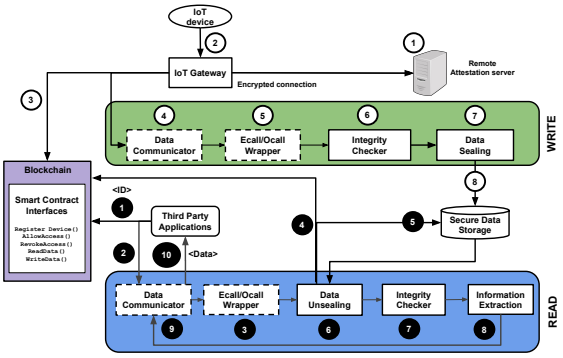

Figure 4: Illustration of the Data Flow in IOTSMARTCONTRACT

B. IOTSMARTCONTRACT Detailed DataFlow

In Figure 4, we show a detailed data flow diagram of IOTSMARTCONTRACT. For a device to write or read data, first, the device communicates with the IoT gateway in **Step** ③ to register itself with the blockchain. For the IoT gateway to trust the SGX platform, it performs remote attestation as shown in **Step** ①. To perform data write, the device communicates with the IoT gateway in **Step** ②. The gateway then retrieves the smart contract address in **Step** ③. The gateway will then encrypt and hash the data. The hashed data will be written to the blockchain using the `writedata` function in the smart contract. The raw encrypted data is then written to the SGX platform in **Step** ④. By using the Ecall/Ocall wrapper, the untrusted module in the SGX application communicates with the trusted module as shown in **Step** ⑤. In **Step** ⑥, the Integrity Checker module calculates the hash-based message authentication code(HMAC) of the data and appends the HMAC of the data before the data is sealed and written to disk in **Step** ⑦ and **Step** ⑧.

For the read operation, the user must register third party users with the smart contract by using the `allowAccess` method. To revoke access, the user calls the `revokeAccess` function. The third party user communicates with the smart contract as shown in **Step** ❶ to obtain the hash of the data generated by the device by supplying the device Id. The smart contract checks if the third party user can access the data from the device using the device Id and the address of the third party user,

if permission is granted, the hash of the data is returned and can be used to access the data from the SGX storage platform. In **Step ❹**, the SGX application rechecks with the smart-contract using `READDATA` API to determine if the third party user can access the data hash identifier supplied by the third party request. If access is allowed, the SGX application retrieves the data from secure storage **Step ❺** . Note that the overhead for read operation from the blockchain is insignificant as we will show in the evaluation section in Table I. The data is then unsealed in **Step ❻** and the Integrity Check **Step ❼** recalculates the HMAC of the data which is then compared with the stored HMAC. If the HMAC is unmodified the data is read and returned the the user as shown in **Step ❾** and **Step ❿**.

V. Implementation

We used five real IoT devices and a mobile phone to evaluate IoTSmartContract. The devices includes Philip Hue Hub with Zigbee light bulb, Samsung Smartthings Hub with Motion/Proximity sensor, Belkin Wemo Switch, Wemo Wall socket and a heart rate monitor mobile application on android.

A. Ethereum Smart Contract

We implemented the IoTSmartContract smart contract component using the Ethereum blockchain. Our implementation consists of 50 lines of code in solidity programming language. The code footprint needs to be concise so as to limit the amount of Ethereum gas needed to run a smart contract transaction. To limit storage space needed to store data in the blockchain, we only store the hash of the data. We ran the smart contract on the Rinkeby Ethereum test network for evaluation.

We implemented the following interfaces `registerDevice`, `allowAccess`, `writeData`, `readData` and `revokeAccess` that enable the IoT devices to interact with the smart contract . By using the geth Ethereum client, we can retrieve the smart contract address in the blockchain and performed operations such register devices, write data, read data, write and read access policy update and revoke access policy.

VI. Evaluation

Table I shows our evaluation result for each smart contract operation in gas used by the miners to complete an operation call. To confirm a transaction, the transaction must be included in a generated block. The data payload size for device 1 is 27 bytes, device 2, 47 bytes, device 3, 132 bytes, device 4 and 5, 127 bytes respectively

Smart Contract Interface	Parameters	Gas Used
`registerDevice`	Device ID	47543
`allowAccess`	Device ID, ThirdParty Address	29517
`writeData`	Device ID, DataHash	51049
`readData`	Device ID, ThirdParty Address	–
`revokeAccess`	Device ID, ThirdParty Address	14792

Table I: Efficiency of Smart Contract Application based on Gas usage

while the hash length is 256 bits. As seen in Table I, `registerDevice` uses 47,543 gas to complete its operation, `allowAccess` required 29,517 gas, `writeData` required 51,049 gas and `revokeAccess` required 14,792 gas. `readData` did not use any gas since reading from a smart contract is done on the local blockchain which does not require any mining.

In Figure 5, we compared the efficiency in gas usage required by miners to complete write operation in the blockchain. We compared two scenarios where the whole data which is encrypted from the devices is written to the blockchain versus writing only the hash of data. By considering 5 device types, we show that device 1 used 59,846 gas for hashed data compared to 159,234 gas required for raw data write which is a reduction of 169%. Device 2 used 53,454 gas for hashed data and 92,926 gas for raw write which gives a reduction of 73%. Device 3 and 4 used 58,974 gas for hashed data while 159,000 gas is required to write raw data which gives a reduction of 138%.

In Figure 6, we show the impact of increasing write workload on the blockchain. By increasing the write workload between 500 write requests to 2000 write requests, we measure the transaction throughput per second on the blockchain. Without hashing, For 500 write workload, the write transaction throughput is 10.56 writes per second. For 1500, it is 9.26 and for 2000 writes, the write throughput is 8.6 writes per second. With hashing, for 500 write workload, the write transaction throughput is 8.8 writes per second. For 1500, it is 7.9 and for 2000 writes, the write throughput is 7.2 writes per second. The write throughput decreases with increasing write workload. The write throughput also decreases with hashing enabled. Even though from Figure 5, the gas used with hashing enabled is constant because the hashing function produces 256 bit data for storage on the blockchain, the write throughput is lower because of the hashing process before writing the data to the blockchain.

A. Sealing and Unsealing Overhead

In Figure 7, we show overhead for sealing and the unsealing operation on the SGX platform. The x-axis represents the block size and the y-axis represents the CPU cost in milliseconds. By using a block size of 1,024 bytes,

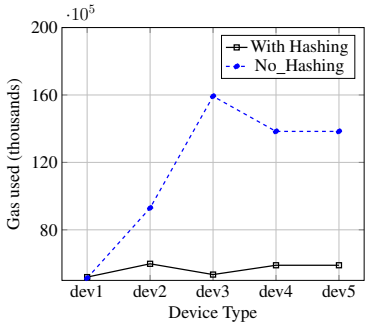

Figure 5: Gas utilization for Write Operation on SmartContract.

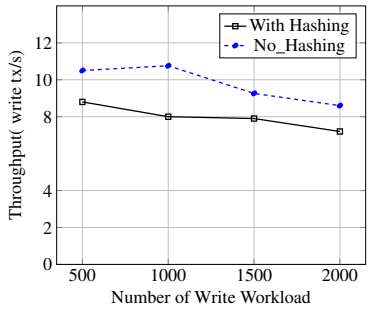

Figure 6: Throughput based on Increasing Write Workload

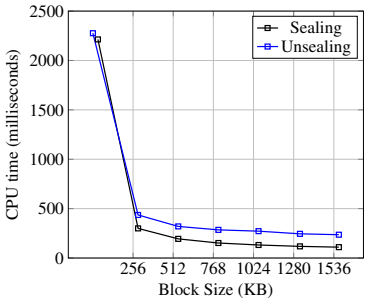

Figure 7: Avg Seal and Unseal time

the average time it takes to seal a single batch record of 2.8 MB is 400 milliseconds compared to 2,000 milliseconds when using 32 bytes block size. With increasing block size, the time to seal and and unseal data reduces. This is as a result of reduction in frequency of number of blocks of data between the enclave and the untrusted module of the application.

VII. Limitations and Future Work

In this work, we leveraged the immutability of the blockchain network to store audit information on how IoT data is stored and read by users. One of the main limitation of using blockchain is the scalability problem. This limitation is not pertinent to our solution, since the data is not always needed immediately and all pending writes and read can be processed and committed to the blockchain at a later time. In addition, this limitation does not apply to read operation as shown in the evaluation. One way to overcome this limitation is to use private blockchains. By using private blockchains, we can eliminate the time used to mine block since all participants in the network is permissioned or known.

VIII. Related Work

In this section, first, we provide discussion on related work on blockchain, and IoT system.

Blockchain. With the increase in adoption of blockchain technology, various researchers have proposed different use cases for the new technology. Zyskind et al. [13] proposed using blockchain to decentralize storage of data. Our work differs from theirs, since we provide a full implementation that leveraged SGX secure computing to store raw data in order to defeat malicious attackers. Dorri et al. [22] proposed using blockchain to manage IoT network. By providing a light weight blockchain consensus system, devices with low processing power can run blockchain independently. Various works exist on how to improve the performance of blockchain technology as seen in [23].

IoT System and Applications. In previous work by Earlence et al. [1], they show how a smart lock can be compromised by attacking the Samsung Smart Application. They demonstrated an attack that requested limited access permission to only perform lock action on a smart lock, but instead gained full control privilege to also perform unlock action. Vijay et al. [24] show how they capture non encrypted network traffic from Wemo device to perform a replay attack on the device.

IX. Conclusion

As adoption of IoT device usage increases, proper data access audit, data usage transparency and data privacy is very critical due to the vast amount of data generated by these devices. This paper introduces IoTSMARTCONTRACT that offers decentralized data access control policy system, data security and data integrity by leveraging recent advances in blockchain technology and trusted computing using Intel SGX. Our approach utilizes the blockchain to manage data access to IoT data in a decentralized way and stores the raw encrypted data in SGX enabled platform by ensuring all data processing and storage is done in the secure enclaves. Our platform utilizes real blockchain platform Ethereum to evaluate our approach. We used real Intel SGX platform to evaluate our secure storage platform.

X. Acknowledgement

This work was supported in part by NSF award #1513704, AFOSR award FA9550-14-1-0173, ONR awards N00014-14-1-0030 and N00014-17-1-2295, an award from Lockheed-Martin. and NSA.

References

[1] E. Fernandes, J. Jung, and A. Prakash, "Security Analysis of Emerging Smart Home Applications," in *Proceedings of the 37th IEEE Symposium on Security and Privacy*, May 2016.

[2] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, and A. Prakash, "ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms," in *Proceedings of the 21st Network and Distributed System Security Symposium (NDSS'17)*, San Diego, CA, February 2017.

[3] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[4] E. Bertino, "Data privacy for iot systems: Concepts, approaches, and research directions," in *Big Data (Big Data), 2016 IEEE International Conference on*. IEEE, 2016, pp. 3645–3647.

[5] P. A. H. Williams and V. McCauley, "Always connected: The security challenges of the healthcare internet of things," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, Dec 2016, pp. 30–35.

[6] N. M. Gonzalez, W. A. Goya, R. de Fatima Pereira, K. Langona, E. A. Silva, T. C. M. de Brito Carvalho, C. C. Miers, J. E. Mngs, and A. Sefidcon, "Fog computing: Data analytics and cloud distributed processing on the network edges," in *2016 35th International Conference of the Chilean Computer Science Society (SCCC)*, Oct 2016, pp. 1–9.

[7] M. M. Masud, L. Khan, and B. Thuraisingham, "A scalable multi-level feature extraction technique to detect malicious executables," *Information Systems Frontiers*, vol. 10, no. 1, pp. 33–45, Mar. 2008. [Online]. Available: http://dx.doi.org/10.1007/s10796-007-9054-3

[8] H. Hu, G.-J. Ahn, and J. Jorgensen, "Detecting and resolving privacy conflicts for collaborative data sharing in online social networks," in *Proceedings of the 27th Annual Computer Security Applications Conference*, ser. ACSAC '11. New York, NY, USA: ACM, 2011, pp. 103–112. [Online]. Available: http://doi.acm.org/10.1145/2076732.2076747

[9] S. Nakamoto, "A peer-to-peer electronic cash system," bitcoin.org, 2009, (Accessed on 08/09/2017).

[10] E. Foundation, "Ethereums white paper." https://github.com/ethereum/wiki/wiki/White-Paper, 2014, (Accessed on 08/09/2017).

[11] Filecoin, "Filecoin: A decentralized storage network." https://filecoin.io/filecoin.pdf, 2017, (Accessed on 08/09/2017).

[12] slock, "Initial coin offering market," https://slock.it/, 2017, (Accessed on 08/09/2017).

[13] G. Zyskind, O. Nathan *et al.*, "Decentralizing privacy: Using blockchain to protect personal data," in *Security and Privacy Workshops (SPW), 2015 IEEE*. IEEE, 2015, pp. 180–184.

[14] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 2016, pp. 839–858.

[15] V. Costan and S. Devadas, "Intel SGX explained," *IACR Cryptology ePrint Archive*, vol. 2016, p. 86, 2016. [Online]. Available: http://eprint.iacr.org/2016/086

[16] M. Crosby, P. Pattanayak, and S. Verma, "Blockchain technology: Beyond bitcoin," 2016.

[17] N. Santos, H. Raj, S. Saroiu, and A. Wolman, "Using arm trustzone to build a trusted language runtime for mobile applications," in *ACM SIGARCH Computer Architecture News*, vol. 42, no. 1. ACM, 2014, pp. 67–80.

[18] L. Van Doorn, "Hardware virtualization trends," in *ACM/Usenix International Conference On Virtual Execution Environments: Proceedings of the 2 nd international conference on Virtual execution environments*, vol. 14, no. 16, 2006, pp. 45–45.

[19] V. Karande, E. Bauman, Z. Lin, and L. Khan, "Sgx-log: Securing system logs with sgx," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '17. New York, NY, USA: ACM, 2017, pp. 19–30. [Online]. Available: http://doi.acm.org/10.1145/3052973.3053034

[20] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978. [Online]. Available: http://doi.acm.org/10.1145/359340.359342

[21] P. Hue, "Philip hue iot portal," http://www2.meethue.com/en-us/, 2017, (Accessed on 08/09/2017).

[22] A. Dorri, S. S. Kanhere, and R. Jurdak, "Towards an optimized blockchain for iot," in *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, ser. IoTDI '17. New York, NY, USA: ACM, 2017, pp. 173–178. [Online]. Available: http://doi.acm.org/10.1145/3054977.3055003

[23] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-ng: A scalable blockchain protocol." in *NSDI*, 2016, pp. 45–59.

[24] V. Sivaraman, D. Chan, D. Earl, and R. Boreli, "Smartphones attacking smart-homes," in *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, ser. WiSec '16. New York, NY, USA: ACM, 2016, pp. 195–200. [Online]. Available: http://doi.acm.org/10.1145/2939918.2939925