Energy Efficient Blockchain Consensus Algorithm using Intel Software Guard Extensions

Conor Shirren

Abstract—With the Growing number of IoT devices in recent years, the need to maintain the integrity and confidentiality of data collected through embedded devices is increasing rapidly. Despite blockchains capability of ensuring trust and confidentially in communication between nodes in a network, the current (most popular) implementations of blockchain technologies are not sufficient for a successful IoT based network. Blockchains most popular implementation, Bitcoin, employs an energy-wasteful consensus protocol which is simply not feasible for implementation on low-powered IoT devices. Proof of Elapsed Time (PoET) is a novel consensus algorithm developed by Intel that could potentially overcome the barriers between IoT and Blockchain. Instead of nodes competing to solve a cryptographic challenge and mine blocks, a random lottery is ensured through the use of a trusted execution environment.

Keywords— IoT, Blockchain, SGX, PoET, PoW, TEE, NodeJS

I. Introduction

The internet of things (IoT) is a technological paradigm which is revolutionizing the way humans and computers talk to each other. The internet of things, at its core, is a large network of inter-connected embedded devices. These embedded devices spam various sectors of industry in today's economy, with application in various network architectures. These applications include Wide Area Networks (e.g. Agriculture, Automotive, and security devices), Local Area Networks (e.g. Healthcare, Entertainment, Appliances, and Automation), and Personal Area Networks (e.g. Toys, Remote-Controls, Wearables, and Safety Devices). According to [1], the number of IoT devices is set to grow to 75 Billion in 2025, up from 0.9 Billion IoT devices in 2009. With this growing infrastructure of low-powered interconnected devices, there is an urgency for a solution to keep the data on these devices secure.

IoT devices generate enormous amounts of data every day, which is stored and analyzed to make decisions. This analysis can range from retrieving your heart rate on a small wearable fitness device, to making large scale business decisions based on extensive amounts of data. In either case, the data being analyzed is important, and more often than not, is also private.

Where there is data, there is a consistent need for security and privacy. IoT devices transmit vast amounts of data across wireless networks for applications in various industries. This wireless transmission of personal information can often be prone to interception. Along with the storage of personal information, embedded IoT devices are often used to control electronic devices like heating systems and automation in homes and businesses. The need for security in embedded IoT devices was highlighted by the *Marai Malware* attack in 2016 [2]. This malware was used to create a 380,000 IoT-based botnet (i.e. a group of compromised computers working in tandem performing malicious activity) used in one of world's most devastating DDoS (Distributed Denial of Service) attacks to date. The malware was used to scan the internet for IoT devices which still used its default root username and password (e.g. the Raspberry Pi's default root username is *'pi'* with password *'raspberry'*). These devices could then be spammed with noise until they could not receive any legitimate traffic. The *Marai Malware* attack affected some of the world's largest companies, including Amazon, Twitter, Spotify and Netflix [2].

In recent years, Blockchain technology has proven itself to be a disruptive technology in the *Fin-tech* industry. It allows for transmission of cryptographically secure data over a distributed network. Bitcoin, Blockchain's most notable application, uses a *Proof of Work* mechanism to ensure consensus between nodes on the distributed network. Although this consensus algorithm is deemed to be robust against misbehaving actors in the network, the algorithm requires nodes to dedicate computation time and energy in order to *solve the puzzle* associated with the PoW algorithm. This mechanism may be adequate for computer's running powerful GPUs (Graphic Processing Units) but would not suffice for many embedded IoT devices which are commonly low powered to ensure a prolonged battery life (i.e. many IoT devices are placed in remote locations and powered by battery).

This paper has examined the use of trusted execution environment *'Intel SGX (Software Guard Extensions)'* in an energy-efficient blockchain consensus algorithm which uses a TEE to run a secure random lottery to provide consensus.

II. Prior Work

A number of research papers were reviewed through the process of this project- one of which can be found at [3], where a blockchain was used to design a network for IoT data, although the blockchain was ran on external node capable of running PoW. The IoT devices communicated to the blockchain through a management hub. The paper [4] also looks at the used of blockchain for IoT in the case of a smart home. The paper discusses the possibility of storing IoT smart home data on a blockchain. As in [3], the blockchain is stored on local computers in the home capable of IoT. This paper looks to bridge the current gap between blockchain and IoT: the ability to use an energy efficient consensus algorithm in order to run a blockchain on remote low powered IoT devices.

III. Implementation

A blockchain network was developed in order to compare the use of PoW and PoET (Proof of Elapsed Time) consensus protocols in the same application. The blockchain network was developed using the NodeJS runtime environment, which was inspired by [5], which allows for the development and testing of a PoW based consensus protocol.

An Intel SGX application was then be developed in order to generate a PoET consensus protocol (written in C/C++) to be integrated with the JavaScript based blockchain. This PoET algorithm was then tested and compared against the PoW algorithm in order to determine a computational profile suitable for embedded IoT devices.

A. Development of PoW based Blockchain using NodeJS

A blockchain network - at its simplest form - is a distributed and decentralized ledger of transactions. These transactions are bundled into blocks, and cryptographically linked together through the use of hash functions.

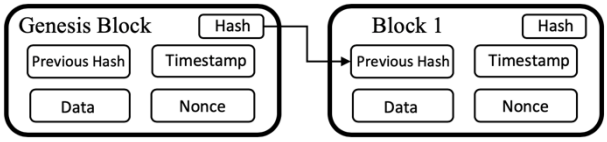

Fig. 1. Structure of Blocks cryptographically linked inside the blockchain network

The basis of this JavaScript based blockchain is a Block Class which can a number of key states: a timestamp, data, a nonce value, a difficulty level (for PoW), a previousHash, and a hash value. These states make up the core of a block on the blockchain.

The next core class of this NodeJS application is the Blockchain Class. This class forms the blockchain by linking the blocks together. This class contains one state – a chain, which is an array of blocks. The blockchain is first instantiated, where its constructor generates a chain array and inserts a default block. The first block on the blockchain, which is known as the genesis block, is hard coded with default values. Each block is then mined onto this block using the specified consensus protocol.

```
Blockchain -
 [ Block {
    difficulty: 0,
    timestamp: 'T-Genesis',
    previousHash: undefined,
    Hash: undefined,
    data: [],
    nonce: 0 } ]
```

Fig. 2. The structure of the Block class in JSON format. This block is the genesis block which is generate via the Block.genesis( ) function

To add blocks to the chain, a node module *'crypto-js/sha256'* is used to generate a SHA256 hash for the contents of the block. The inputs to this hashing algorithm include a timestamp, data, a nonce value, the block difficulty, and the hash of the previous block. Once a hash has been generated for a specific block, it is then added to said block, and push onto the chain array in the blockchain.

```
Block {
  difficulty: 0,
  timestamp: 1566310279311,
  previousHash: undefined,
  Hash: 'dc4201894485aacceb12b8c9fc339f6f82ec5c011e711e58e0bebbd31de6a89e',
  data: 'foo',
  nonce: 0 }
```

Fig. 3. A block generated with sample data to be added to the blockchain. The previous hash is undefined, since the previous block is the genesis block.

Once a block has been added to the chain, it is imperative to ensure the that the chain has not been tampered with. This can be achieved through checking two criterion:

- Compare the current block's 'previousHash' with the hash of the previous block
- Check if the first block in the array is equal to the genesis block.

This specific blockchain application uses a Proof of Work consensus protocol for mining blocks to the network. This means that nodes in the network must concede computational time and energy to solve a puzzle which allows the node to add a 'mined' block to the array for an award (e.g. the award for mining transactions on the Bitcoin network is *bitcoin*). In the Block Class, a *'mineBlock'* method is created which used to add blocks to the network. This method defines the consensus protocol used in the network. The PoW algorithm is governed by a difficulty, which is defined by the network.

The PoW algorithm repeatedly calculates a hash of the contents of the block, where a nonce value is changed between calculations, until the resulting hash value has a leading number of zeros. This leading number of zeros is defined by the difficulty in the network. Once this condition is met, a new block is created containing this hash and is pushed to the chain array. The algorithm used to perform this PoW consensus in JavaScript can be seen below.

```
do {
    nonce++;
    timestamp = Date.now();
    difficulty = 3;
    hash = Block.hash(timestamp,previousHash,data,nonce,difficulty);
    // checking if we have the required no of leading number of zeros
} while(hash.substring(0,difficulty) !== '0'.repeat(difficulty));
```

Fig. 4: The PoW algorithm used in the Block Class of this NodeJS application.

In bitcoin, the network is design so that the network mines a new block to the chain, on average, every ten minutes. This means that the difficulty of the network must adjust over time to account for increasing processing power of nodes. The application being developed in this paper will not include this feature as it is not necessary for testing purposes.

B. *Development of PoET algorithm using Intel's SGX*

Proof of Elapsed Time (PoET) uses the concept of trusted computing to provide Byzantine fault tolerance to a distributed system. In this section, an overview will be provided of the steps carried out to out to design a TEE based consensus algorithm for a blockchain network to improve on the energy consumption downfalls associated with Bitcoin's Proof of Work.

Intel SGX is a new technology [6] that provides CPU implemented trusted execution environments. Although the concept of a TEE is not new, SGX does provide the functionality to instantiate multiple secure enclaves on a single CPU, which will be useful for this paper. Intel's SGX allows an application to *'keep its confidential data inside a secure enclave, only allowing access from within'* [7].

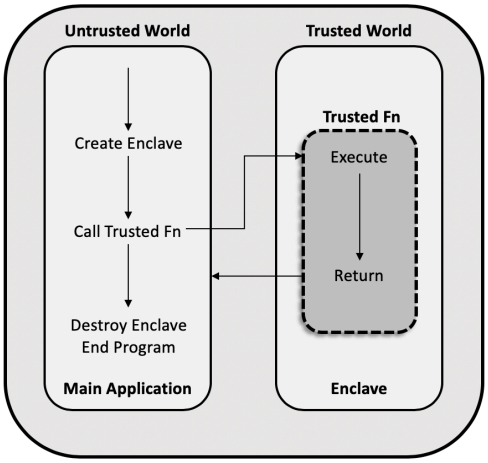

Fig. 5: Intel's Software Guard Extensions. Secure Enclaves are used to ensure the integrity and confidentiality of secrets inside an application

To utilize the full capabilities of Intel's SGX, an Intel NUC (with SGX BIOS support) running Ubuntu 16.04 was used to develop the SGX application. Developing an application in SGX requires the following four step approach:

1. Identify the Application's secrets
2. Identify the providers and consumers of said secrets
3. Determine the Enclave Boundary
4. Tailor the application components for the enclave

The first step in designing the application was to identify which pieces of information needs to be secured by the enclave. After defining the boundary between the secure enclave and the external unsecure world, a number of OCALLs and ECALLs could be formed to allow secure communication between the application and the enclave.

- OCALL – function which is stored inside the enclave and is used to transverse the trusted boundary to the untrusted world.
- ECALL – function located in the untrusted world which can access inside the enclave.

The life cycle of OCALLs and ECALLs can be seen in the figure below. SGX utilizes proxy functions to transverse the trusted boundary in order to ensure the integrity and confidentiality of the secrets stored inside the secure enclave.

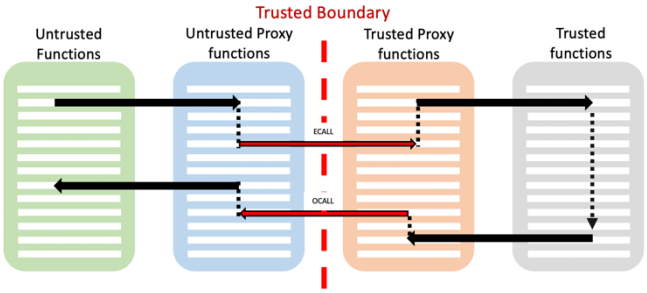

Fig. 6. The SGX OCALL & ECALL lifecycle. These functions are defined in the SGX EDL file. Proxy functions are generated by the EDL file and are used to transverse the secure boundary.

In order to build the PoET algorithm using Intel SGX, the following functionality is required:

1. Create a new enclave
2. Generate a (pseudo) random wait time inside the enclave
3. Wait the allotted time before returning an attestation to the main application
4. Proceed to mine the block into the chain

Since SGX restricts the use of OS related function, some libraries that would commonly be used for random number generators (i.e. rand.h) and wait timers (i.e. the sleep function) are not available for use.

The functions required to develop this application are defined in the EDL file of the SGX application.

```
enclave {

    trusted {
        public void printf_helloworld();
        public void poet(int currentTime);

    };

    untrusted {
        void ocall_print_string([in, string] const char *str);
        void ocall_print_int(int value);
        unsigned int ocall_currentTime();
    };

};
```

Fig. 7. The OCALLs & ECALLs required for the PoET SGX Application

The algorithm used in this PoET SGX protocol can be depicted in the figure below. The main SGX application, which is untrusted, makes an OCALL to the poet function depicted in Fig. 7 above. This poet function is located and executes inside a secure enclave. The poet function receives the current system time as an input argument, which in turn is used to generate a pseudo random number. This random number – which for testing purposes, returns a number form [1,10] – is used as the *'wait time'* for the PoET Algorithm. A *'while'* loop is then used to cross reference the time at which the function was first called to the current system time – a new ECALL must be used for this. Once the generated time has elapsed, control is then passed back to the main application for further functions to take place.

This algorithm is designed and executes on the SGX capable CPU in the Intel NUC. The program runs in SGX Hardware mode in an attempt to provide Byzantine Fault Tolerance [8]. A flowchart depicting the algorithm's mechanism can be seen in Fig. 8 below.

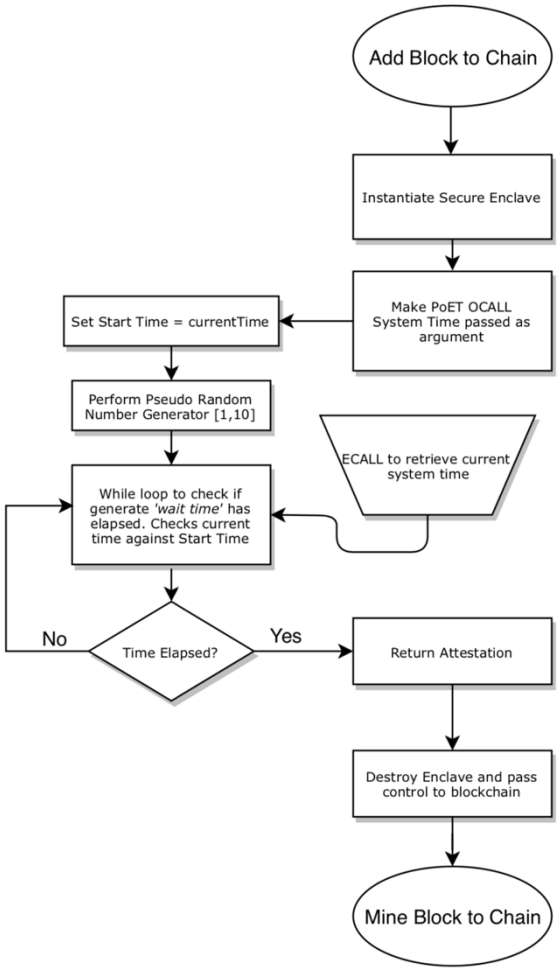

Fig. 8. Flow Diagram for PoET SGX Algorithm

C. *Integrating SGX with NodeJS based Blockchain*

To integrate the SGX application with this NodeJS based Blockchain, the following steps are required:

- Integrating the SGX C++ application into the NodeJS Project Directory
- Integrating the PoET algorithm into the mineBlock method of the Block Class
- Integrating the PoET algorithm into the addBlockToChain method of the Blockchain Class

The SGX C++ application in placed in the root directory of the node project. Utilizing the '*child_process'* node module, the application can be executed through the Linux command line. Since JavaScript is asynchronous in nature, a callback function is created to execute once the SGX application has executed. If any errors occur in execution, an error is thrown by the NodeJS program. The output of the SGX application is then checked as an attestation. Once the PoET timer has finished, it returns the string 'poet' to the console. If the NodeJS reads in this exact string, the block is mined to the chain. This method itself returns a callback function, which allows the block to be pushed to the chain upon completion of the a block being mined.

This SGX application allows the blockchain to perform as designed in Section III Part A without the need for wastefully computation time and energy in attempting to solve a puzzle. The use of Intel's SGX attempts to reduce energy consumption while also attempting to ensure Byzantine Fault Tolerance (BFT).

IV. Testing.

A. *Testing with Jest*

The main testing framework used for this project is Jest. It is a JavaScript testing framework with a focus on simplicity [9]. To test the blockchain that has been developed, a testing jest file will be created for each JavaScript file in the project in order to test the various class and functions in the project. Once the jest node module has been installed in the project, the testing environment can be defined by inserting "test: 'jest –watchAll'" into the project's package.json file. Once this has run, the jest testing environment can be run by executing 'npm run test' in the terminal window of the project.

The Jest testing framework is used to provide a systematic and measurable approach to ensure all classes and methods developed in the project provide the correct functionality.

B. *Energy Efficiency Testing*

In this section, the test scripts designed in order to analyze the computational profile of both the PoW and PoET consensus algorithms developed will be discussed. In order to analyze the energy consumption of both of these algorithms under similar conditions, a set of test were designed in order to investigate the time taken to mine a block to a chain using the PoW protocol at varying levels of difficulty. Once the conditions under which both consensus protocols execute in the same time, the energy consumed in doing so can be observed. To examine this, a test script was designed in order to determine the exact function execution time for each consensus algorithm. This was achieve through JavaScript's 'process.hrtime' method which can be used to return the system time in nanoseconds. This time was recorded before and after a block had been successfully mined to the blockchain in order to determine the mine time. For PoW, this test was performed 30 times at each difficulty level [0,5]. At PoW difficulty level 6 and above it was not possible – due to time constraints – to test.

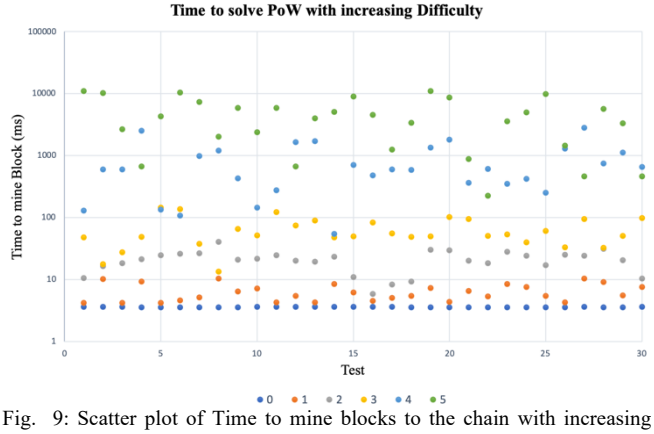

Fig. 9: Scatter plot of Time to mine blocks to the chain with increasing PoW difficulty

Once each level of PoW had been tested, the PoET algorithm was then tested 30 times using the pre-set range [0,10] which had been designed into the SGX application. Statistically, for a truly random number in the range [1,10], the average time for a block to be mined to the chain using PoET should be approximately 5 seconds. Although, due to limitations in the generation of a truly random number, along with limitations with the amount of time that could be spent testing, it was expected that the average time of the PoET mining process will range between 3 and 7 seconds.

V. Analysis

A. Computational Profile

Once these datasets were recorded in Excel, they could be used to obtain a mean and standard deviation of each difficulty level. The values for μ and σ can be seen in Table 1 below.

	PoW L0	PoW L1	PoW L2	PoW L3	PoW L4	PoW L5
μ	3.56899271	6.33916464	21.3043853	63.2587902	818.9168564	4718.21996
σ	0.02159884	2.09953006	7.69371935	33.7813056	725.4574374	3538.59142

Table 1: Standard Deviation and Mean of PoW Timings

It can be observed that as difficulty increases, the standard deviation increases rapidly in relations to its mean value. From this it can be noted that a PoW consensus protocol with a higher level of difficulty yields a 'lottery' like system with an unpredictable chance of success. A plot of the result shown in Table 1 can be seen in Fig. 10 below.

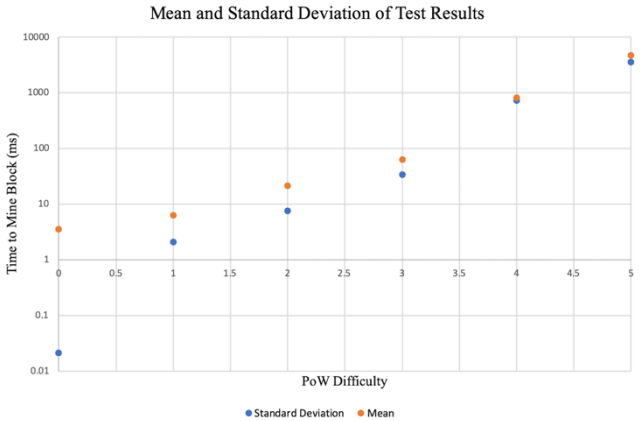

Fig. 10. Plot of mean time taken to mine a block of data to the chain using the PoW based mining operation at 6 varying levels of Difficulty.

By superimposing the data collected for the PoET Timings onto the PoW timings, it can be observed that the PoW level 5 it most closely related to the PoET mining time.

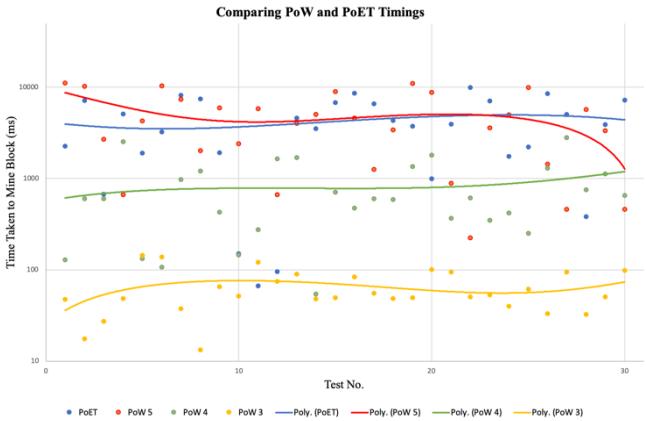

Fig. 11: PoET Timing superimposed over the PoW Timings plot

Based on Fig. 11, the PoW Level 5 will be used to compare against the PoET algorithm in terms of CPU % use during mining. To do this, a Linux command line program called *'htop'* is used to monitor the CPU while each process is running. This program can be seen in Fig. 12 below.

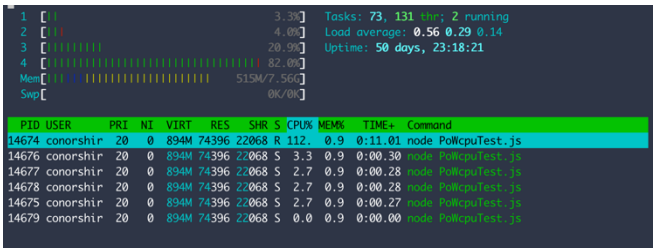

Fig. 12. HTOP monitoring the CPU for the script mining blocks to the blockchain

It can be seen from the image that the PoW mining process requires 112% CPU usage, which really means 1 full core, and 12% of a second core (The Intel NUC has a 4-core processor). This means that the PoW algorithms requires 28% of the processing power from the CPU.

In the case of PoET, the process running the test script was unmeasurable by 'htop', suggesting that it's CPU requirements for performing its functional is minimal – and hence more energy efficient than the aforementioned PoW. It can also be seen from Fig. 10 that the increase in time for different PoW levels increases exponentially, resulting in processing requirements also increasing. In the case of PoET, the processing power for increasing 'wait time' is constant.

Thus, the developed PoET algorithm is energy-efficient in relation to the PoW algorithm also tested.

B. Byzantine Fault Tolerance

Intel's SGX provides an opportunity to build a BFT blockchain through the use of PoET. Although, for this to be the case, the enclave must provide an attestation to allow the nodes on the network to ensure that a secure enclave was used. Intel SGX provides a remote attestation through 'Intel's Attestation Service'.

Although this is an effective method for providing remote attestation, it is heavily reliant on the Intel Attestation Service for providing BFT functionality in a given blockchain. This provides Intel with a governing authority over any given PoET SGX based blockchain, which undermines the motivation for a distributed and decentralized network. If Intel were to remove the IAS, PoET SGX would no longer be BFT.

This project does not use the IAS for remote attestation (only one ID is provided to each SGX capable device, so it is not possible to test IAS on a single machine). Instead, the PoET algorithm in this project checks to ensure a specific string is returned from the enclave to prove the timer finished correctly.

VI. Conclusion

One of the primary problems facing IoT is Security. As IoT devices rapidly increase, the data being gathered, stored and analyzed via these IoT devices is also growing exponentially. This paper looks at investigating the possibility of using the security implementations that blockchain provides, without the computational power requirements that is inherent in Bitcoin's current implementation.

Not all consensus algorithms were investigated in this paper. Bitcoin's own consensus protocol Proof of Work was developed into a simple blockchain application and tested against a Proof of Elapsed Time consensus protocol developed using Intel's SGX.

Due to limited resources – time and physical access to hardware – it is not possible to draw a sweeping conclusion about the ability of Proof of Elapsed Time to work as a solution for IoT current security problem. Although, testing has provided proof that the energy consumption of the PoET is minuscule when compared to its counterpart. This may well pave the way to design dedicated consensus protocol which uses a TEE capable IoT device as its end nodes.

This paper also depicts the necessary steps required to develop both a blockchain application using JavaScript, and an Intel SGX application, along with indicating some major flaws when using Intel SGX for a distributed, decentralized network. Namely, it's requirement for IAS to perform IAS.

This paper has carried out an assessment of PoET and its potential use as a consensus algorithm for IoT based blockchains. It has achieved its goal by implementing both PoW and PoET blockchains, and hence determining that the computational profile of PoET is to be considered energy efficient when compared to that of Proof of Work.

A. Future Work

Further development of this project would build upon the work presented in this paper:

- Make necessary change to current application to allow it to run on a dedicated IoT device which supports TEE (either internally or via an attachable device [10])
- Make enhancements to the current application in order to provide load balancing capability and detection of malicious nodes.
- Design a new remote attestation service which is completely peer to peer and does not require IAS.

References

[1] "• IoT: number of connected devices worldwide 2012-2025 | Statista." [Online]. Available: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/. [Accessed: 20-Aug-2019].

[2] K. Fu *et al.*, "Safety , Security , and Privacy Threats Posed by Accelerating Trends in the Internet of Things," 2017.

[3] O. Novo, "Blockchain Meets IoT: An Architecture for Scalable Access Management in IoT," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 1184–1195, Apr. 2018.

[4] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, "Blockchain for IoT security and privacy: The case study of a smart home," in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, Kona, HI, 2017, pp. 618–623.

[5] K. Khullar, *nodejs implementation of bitcoin. Contribute to kashishkhullar/blockchain-nodejs development by creating an account on GitHub*. 2019.

[6] "Intel® Software Guard Extensions Developer Guide," p. 35.

[7] R. Pires, M. Pasin, P. Felber, and C. Fetzer, "Secure Content-Based Routing Using Intel Software Guard Extensions," *Proc. 17th Int. Middlew. Conf. - Middlew. 16*, pp. 1–10, 2016.

[8] K. Driscoll, B. Hall, H. Sivencrona, and P. Zumsteg, "Byzantine Fault Tolerance, from Theory to Reality," in *Computer Safety, Reliability, and Security*, vol. 2788, S. Anderson, M. Felici, and B. Littlewood, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 235–248.

[9] "Jest · Delightful JavaScript Testing." [Online]. Available: https://jestjs.io/. [Accessed: 22-Aug-2019].

[10] "Enterprise Blockchain Solutions, IIoT," *Filament*. [Online]. Available: https://filament.com/. [Accessed: 26-Aug-2019].