Security Protocol for Internet of Things (IoT): Blockchain-based Implementation and Analysis

Kanwalinderjit Gagneja, Riley Kiefer
Department of Computer Science
Florida Polytechnic University
United States
{kgagneja, rkiefer}@floridapoly.edu

***Abstract*—The combination of blockchain technologies and Internet of Things devices on a network reduces overall network usage by centralizing communication that is broadcasted to all devices via a server. A blockchain-based key management protocol lessens the need for a centralized key authority while retaining similar security compared to its centralized counterpart.**

Keywords*—*blockchain, IoT, key management, hash, Java

I. Introduction

Internet of Things (IoT) devices, like a smart-speaker or smart-thermostat, are typically low-processing power and low-memory computers that interact with each other on a network. Blockchain was first introduced by S. Nakamoto as a secure transaction ledger for a new cryptocurrency known as Bitcoin. A blockchain is a sequence of blocks that contain transactions, or messages in the case of IoTs. Blocks are added to the blockchain through a mining process which takes a significant amount of processing power to compute a valid block that is deemed acceptable by the network. Blockchain technology is created in a way that limits the ability to alter a blockchain and its contents by sheer processing power. The only way to take control of the blockchain is through a theoretical 51% attack, where an attacker must have at least 51% of the computational power on a blockchain in order to alter block contents and validate it faster than all other combined users on the network.

This paper is organized as follows. Section II describes others' research work in terms of IoT devices and blockchain. Section III discusses a blockchain framework implementation. Section IV establishes a key management protocol using the blockchain framework of the previous section. Section V refers to the blockchain parameter updates for dynamic IoT environments. Section VI briefly discusses the possibility of secure data transmission using the blockchain by storing temporary public keys for mutual session key generation. Section VII analyzes the proposed equations, the blockchain, and the difficulty parameter used throughout the paper. Section VIII analyzes the different hashing algorithms from the SHA-3 library for blockchain purposes. Section IX provides a summary and a few closing remarks as a conclusion to this paper.

II. Literature Review

Current research on blockchain technology specialized in the Internet of Things (IoT) devices seeks to tackle several research areas, including hash analysis, cloud computing analysis, and blockchain frameworks to name a few areas of research. Samaniego, Jamsrandorj, and Deters [1] have conducted research on using cloud or fog computing for IoT blockchain applications. Due to the nature of IoTs being low-storage and low-processing power devices, a blockchain network must rely on other forms of computing power to calculate a valid hash. The research found that edge devices are often too constrained with limited power and bandwidth, which lead to testing of fog and cloud platforms. With a network latency analysis, fog computing outperformed cloud computing.

M. Singh, A. Singh, and S. Kim [2] have conducted research on blockchain implementation and frameworks. Several proposals to increase blockchain security for IoTs including the storage of encryption keys on the blockchain for authentication, methods for discerning legitimate blockchain users from illegitimate nodes, and proper IoT configuration.

Fakhri and Mutijarsa [3] have conducted research on smart home applications of IoT devices. They found that IoTs with blockchain technology has a higher level of security compared to platforms without blockchain. This was found by monitoring the "avalanche effect" of hashing functions and after attack simulations. They found out that the Keccak-256 has a higher level of security than SHA-256 because it has more of a unique output for each bit change, or a better avalanche effect.

Mughal, et al. [5] conducted research on signature schemes for IoTs authentication to compensate for their lack of processing power. They proposed a Shortened Complex Digital Signature Algorithm (SCDSA) that offers better resilience to capture attacks while offering better verification with lower communication cost compared to other signature schemes like Elliptic Curve Digital Signature Algorithm (ECDSA) and the Digital Signature Algorithm (DSA). While the SCDSA is still being researched, ECDSA still provides strong security at small key sizes [6] [7].

III. Implementing a Blockchain Framework

The Internet of Things smart network is set up with a structure like Fig. 1. IoT devices are connected to a gateway, which is connected to the blockchain network. The network operator is a trusted party that hosts the blockchain on a server, which is used in conjunction with computing resources (ex. Cloud computing) to perform the resource-intensive work on behalf of the IoT devices.

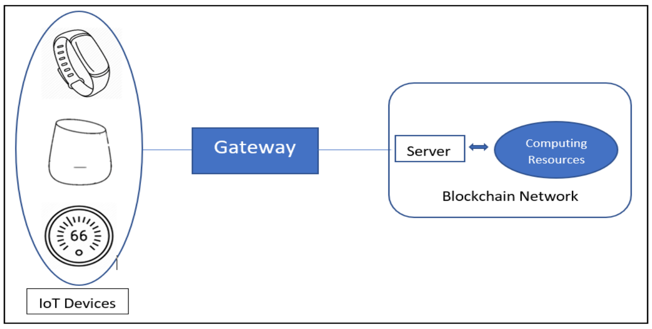


Fig. 1. A smart home network structure using a blockchain-based key management protocol.

A. *Blocks on the Blockchain*

The blockchain is comprised of blocks containing the necessary data fields to provide a basis for a simulated network blockchain for IoTs. In this implementation, the blocks are objects and the blockchain is an array list. If any data fields are not used in the block, they default to NULL.

The following explains the various data fields in use. The *Sender* data field holds the ID of the IoT device that is adding the message to the blockchain. The *Receiver* data field holds the public key used to sign messages. This data field is used for device initialization and mutual session key generation by the ID of the IoT device that is the designated recipient of the message. The *PlainText* data field holds messages or instructions. The *Signature* data field holds the signature of the IoT sender using a public key. The *PublicKey* data field allows for the sharing of public signature verification keys with other IoT devices using the blockchain. The *PreviousHash* data field holds the hash of the previous block. The *CurrentHash* data field holds the value of the current block's hash, calculated using an SHA function. The *Time* data field holds the time of block generation at a predetermined time interval. The *Proof* data field holds the Proof of Work, or the nonce that generates a valid hash according to the blockchain parameters.

B. *Calculating the Hash of a Block*

The hash of a block is generated in the following way. The IoT devices are linked to a gateway that is connected to a cloud or edge computing cluster to carry out the many hash calculations. All data fields in the block except *CurrentHash* are concatenated into a single string of data. Using an agreed-upon hashing algorithm function for hashing the data, the large string of data is hashed and converted into hexadecimal. This hash can now be used to update the *CurrentHash* data field or used to compare an existing hash to check if a block is valid, (explained in the following subsection).

C. *Validating the Blockchain*

The blockchain is verified using a loop that iterates through all blocks in the blockchain and checks three conditions per iteration. If all methods verify each element of a block for all blocks in the blockchain, the blockchain is considered valid.

1) Check CurrentHash: This method checks if CurrentHash has been calculated correctly by generating the hash of the block's data and comparing the result to the hash stored on the existing block.

2) Check PreviousHash: This method checks if *PreviousHash* is the same hash as *CurrentHash* of the previous block.

3) Check Target: The target is a string of characters that determine if a hash is valid. In this method, the target is compared to a substring of the *CurrentHash* of a block. If the target and the substring hash match, the hash is considered valid by the blockchain parameters.

D. *The Difficulty Parameter*

a. Defining the Difficulty Parameter

The difficulty parameter is a positive integer that designates how many characters of the calculated hash must match the target string for a hash in a block to be considered valid in the blockchain. For example, in Bitcoin's blockchain, the target parameter is a string of all 0s. If the difficulty chosen is 7, the first 7 characters of a block's calculated hash must contain all 0s. In our implementation, the target string is compared to the hash substring. If the target string is not equal to the hash substring, the *Proof* (nonce value) must be altered (or incremented) and the hash must be recalculated and compared to the target until a valid hash is calculated. Once a valid hash is found, the block is ready to be appended to the blockchain.

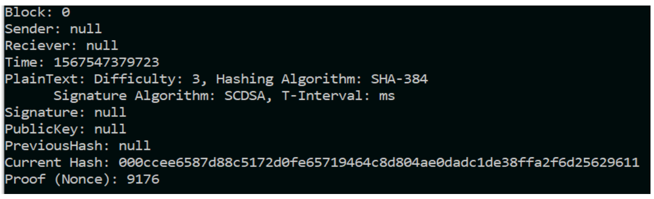
b. Proposed Difficulty Parameter

In the Bitcoin implementation of the difficulty parameter, valid hashes contain a predictable amount of preceding 0s. Our proposed difficulty parameter seeks to reduce the hash reusability of blocks with the same contents. To do this, the target string starts with all 0s. Each new block increments the target string by 1. The blocks are modular of the maximum value possible at the current difficulty level. For example, at difficulty 2, there are 256 possible values ranging from 0 to 255 (in hexadecimal). At block 256, the target string would restart at all 0s. With this solution, the blocks with the same content have a smaller chance of having different valid block hashes. While the chances of a block having exactly the same content and same time is very minimal, it is still a preventative and easy-to-implement solution.

E. *The Genesis Block*

The genesis block (or 0th block) is important in standardizing the blockchain across all devices on the network. The genesis block has a limited filled number of data fields because it is the first block in the blockchain. Since the network operator is the server of the blockchain, it is the network operators' responsibility to initialize the following blockchain parameters using the *PlainText* data field as shown in Fig. 2. The genesis block sets the blockchain parameters inside the *PlainText* data field. For example, in Fig. 2 the blockchain difficulty is set to 3, the hashing algorithm is set to SHA-384, the signature algorithm is set to SCDSA [5], and the time interval is set to milliseconds.

- Difficulty- The difficulty is a positive integer that estimates the computational complexity of mining a valid hash of the blockchain by giving the required length of matching characters of a hash substring to its target string (refer to section II, part D).
- Hash Algorithm- The hash algorithm specifies which SHA function to utilize when generating and calculating hash strings for blocks (ex. SHA-256).
- Key Pair- The key pair parameter specifies the algorithm used to sign messages in the *Signature* data field of a block (ex. SCDSA [5]).
- Time Interval- The time interval parameter specifies the precision of the timestamps (ex. milliseconds).



```
Block: 0
Sender: null
Reciever: null
Time: 1567547379723
PlainText: Difficulty: 3, Hashing Algorithm: SHA-384
      Signature Algorithm: SCDSA, T-Interval: ms
Signature: null
PublicKey: null
PreviousHash: null
Current Hash: 000ccee6587d88c5172d0fe65719464c8d804ae0dadc1de38ffa2f6d25629611
Proof (Nonce): 9176
```

Fig. 2. Genesis block of the blockchain, where the Operator initializes the difficulty parameter, hashing algorithm, signature algorithm, and time interval.

We thank CyberFlorida for funding this research work with the grant # 3910-1004-00-D

IV. Establishing a Blockchain Key Management Protocol

A key management protocol that utilizes blockchain technology as a communication ledger significantly improves upon several of the shortcomings of a centralized key management protocol with a network operator as the authority. The network

A. *The Network Operator*

In a traditional network hierarchy (Fig. 3), the network operator acts as the Key Authority, which manages and distributes key pairs to various IoTs and authenticates users on the network. The problem with this implementation is that some IoTs may not be able to store every single public key received, and the network operator may cost the network significant bandwidth in order to send mass communications to all IoT devices for each update.

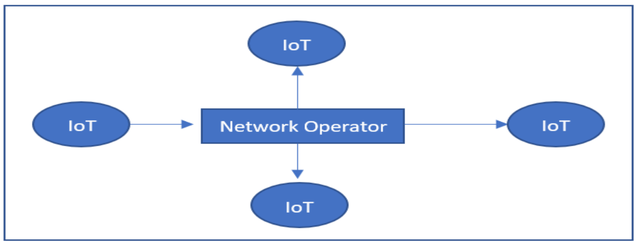


Fig. 3. Traditional Key Distribution: An IoT with a new signature key pair must send the network operator the new public key and the network operator distributes the key to all devices.

For a decentralized blockchain (Fig. 4), a network operator does exist, but operates with less authority. In this blockchain-based key management protocol, the network operator acts as a trusted server for the blockchain to ensure the blockchain is secure and performing optimally. The blockchain acts as shared storage and ledger for all communications. IoTs can extract information as needed.

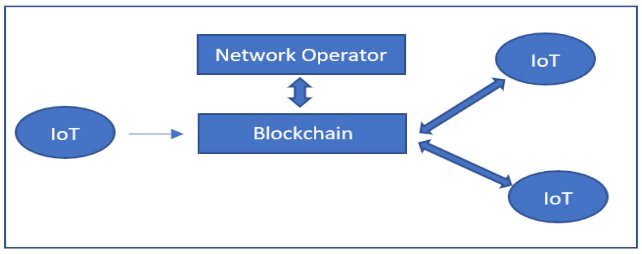
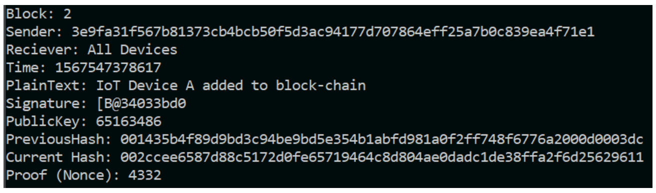


Fig. 4. A Blockchain-based key distribution: An IoT with a new signature key pair adds the public key pair to a new block on the blockchain. IoTs can read from the blockchain is necessary to extract the public key.

The network operator sets the initial difficulty parameter along with other standardization for the blockchain at the genesis block. All devices using the blockchain hosted by the network operator must use the difficulty parameter as the minimum prerequisite of a valid hash for a block to be considered valid by the blockchain and its users. As more devices use the blockchain, more computing resources may be required. Hence, the network operator may adjust the difficulty over time by appending a message to the blockchain with an updated difficulty value. After the operator updates the difficulty with a new block on the blockchain, all blockchain users must use the new difficulty parameter henceforth for the hashes to be considered valid.

B. *Adding Devices to the Blockchain Ledger*

Every IoT device on the network has some arbitrary user-defined or preinstalled identification. Using the hash algorithm specified in the genesis block, the identification is hashed and is now the IoT's blockchain ID, which will be placed in the *Sender* data field when creating a new block. When initializing a device on the blockchain, a key pair is generated using the algorithm specified in the Key Pair section of the genesis block. The public key is added to the *PublicKey* data field of the block, which allows other devices on the network to extract that public key and verify block signatures to authenticate the sender. Once an IoT device has generated a key pair and added the public key to the block, the *PlainText* data field must have a message containing the device's current state, for example, 1 for ON and 0 for OFF, and any IoT-specific data (ex. A smart thermostat's current temperature setting). Finally, the *Receiver* data field is updated with previously agreed upon identification that represents all devices on the blockchain and the block is mined and appended to the blockchain as shown in Fig. 5. Essentially, this message is broadcasted to all devices in the blockchain. In Fig. 5, block 2 contains the initialization of the IoT of ID "A". Since the IoT device has an ID of "A", the *Sender* data field contains the hash value of "A" using the hashing algorithm of the blockchain. Before filling out the rest of the data fields, the IoT device must generate a key pair (which contains a public and a private key) using the key pair algorithm of the blockchain. The public key of the key pair is added to the *PublicKey* data field. The device's signature is generated using the private key of the key pair and added to the *Signature* data field.



```
Block: 2
Sender: 3e9fa31f567b81373cb4bcb50f5d3ac94177d707864eff25a7b0c839ea4f71e1
Reciever: All Devices
Time: 1567547378617
PlainText: IoT Device A added to block-chain
Signature: [B@34033bd0
PublicKey: 65163486
PreviousHash: 001435b4f89d9bd3c94be9bd5e354b1abfd981a0f2ff748f6776a2000d0003dc
Current Hash: 002ccee6587d88c5172d0fe65719464c8d804ae0dadc1de38ffa2f6d25629611
Proof (Nonce): 4332
```

Fig. 5. Adding the IoT device "A" to the blockchain network.

C. *Sending and Receiving Messages with the Blockchain*

a. Sending Authenticated Messages

With IoTs initialized in the blockchain, the devices can now add authenticated messages to the blockchain. To create an authenticated block, a message must be generated and added to either the *Data* section for unencrypted messages or the *Bytes* section of the block for encrypted messages. The Data section is concatenated into a single data string and hashed using the current hashing algorithm of the blockchain. Then, a system time nonce value is concatenated with the hash of the message contents. Finally, using the private key of the most recent Key Pair generated by the IoT *Sender*, a signature is generated using the concatenated message hash and system time nonce. This signature is then added to the *Signature* section of the block. Once the other block data fields are filled with the appropriate contents, the block is mined and appended to the blockchain upon finding the *Proof* to make the hash of a block's contents valid. The nonce value aids in limiting the reproducibility of the signature while the hash function transforms a message of arbitrary length to fixed-length data, which is useful for signature algorithms that cannot handle large inputs.

b. Reading from the Blockchain and Authenticating Messages

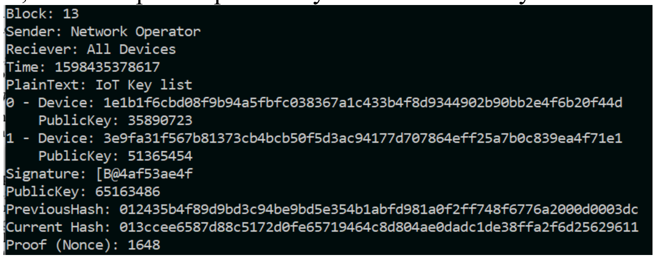
With an arbitrary number of messages on the blockchain in the form of blocks, IoTs need to process the messages designated specifically for them and messages designated to all devices on the blockchain network. A linear search must traverse the blockchain for all new blocks added to the blockchain since the last time the blockchain was checked. For each block, the IoT must check the *Receiver* data field to see if the message is designated for itself or for all devices. If it is not, it must continue traversing until all unchecked blocks are checked. If the block is designated for the IoT or all devices, the IoT must look at the block's *Sender* data field and traverse the blockchain until it finds the block with the sender's most recent public key. Since most IoTs have limited storage and computing capacities, saving the public keys and their corresponding IDs on memory for all devices using the blockchain is inefficient for all parties involved.

One solution to this problem is for every block generated by the network operator should contain the most updated public key pairs and its corresponding identification within the *PlainText* data field (see Fig. 3). Instead of saving all information related to the public key, the IoT simply saves the block number of the most recent block generated and appended by the network operator. Instead of finding the device's most recent public key, the IoT can directly access the block and traverse all devices for the public key rather than all messages on the blockchain. With this public key, the IoT can verify the signature of the block. This is done by computing the hash of the *PlainText* data fields and comparing this value with the un-signed signature after removing the time-based nonce value. If these values are the same, the block's message contents remain unscathed by potential Man-In-The-Middle (MITM) attacks. If the values are not the same, the data has tampered and the block's contents should not be trusted. The Network Operator may issue a conflict resolution action to fix the blockchain so that it contains all authenticated blocks.

V. Updating the Blockchain Parameters and Keys

The organization structure of the network is dynamically changing as IoTs fluctuate in and out of the network. Over time, the network operator must maintain a structure that ensures the continuity of security. This includes tasks like monitoring and updating the difficulty parameter and hashing algorithms. Other tasks include monitoring the lifetime of Key Pairs used by IoTs to create and verify the signatures of devices, storing updated public keys, and validating the blockchain to prevent fraudulent blocks from affecting the network.

To issue these updates to the blockchain network, a block is mined with the network operator's blockchain ID and the updates are listed in the *PlainText* data field. To prevent a fraudulent network operator from making changes to the blockchain, a signature must also be generated to prove the identity of the operator. In the event of an operator's public key update, the operator's updated block must contain a signature using the old public key, and have the new public key placed in the *PublicKey* data field (see Fig. 3). If one or more IoT devices have a signature-generating public key, the network operator's updated block must also list the most updated public keys for all devices as shown in Fig. 3 below. The figure represents block 13. The network operator has reached an arbitrary maximum public key lifetime limit, and has generated a new key pair for authentication signatures. The new public key is placed in the *PublicKey* data field of the block, while the old public key is used to generate the block's signature in the *Signature* data field to verify the operator's identity. In the *PlainText* data field, IoT devices A, B, and C's updated public keys are listed for easy extraction.



```
Block: 13
Sender: Network Operator
Reciever: All Devices
Time: 1598435378617
PlainText: IoT Key list
0 - Device: 1e1b1f6cbd08f9b94a5fbfc038367a1c433b4f8d9344902b90bb2e4f6b20f44d
    PublicKey: 35890723
1 - Device: 3e9fa31f567b81373cb4bcb50f5d3ac94177d707864eff25a7b0c839ea4f71e1
    PublicKey: 51365454
Signature: [B@4af53ae4f
PublicKey: 65163486
PreviousHash: 012435b4f89d9bd3c94be9bd5e354b1abfd981a0f2ff748f6776a2000d0003dc
Current Hash: 013ccee6587d88c5172d0fe65719464c8d804ae0dadc1de38ffa2f6d25629611
Proof (Nonce): 1648
```

Fig. 6. This Network Operator block (13) accomplishes two objectives: it updates the Operator's public key in the PublicKey data field and lists the most updated public keys for 2 IoT devices in the PlainText data field.

VI. Creating Mutual Session Keys for Secure Data Transmission

Secure data transmission is important when using IoTs that may transmit sensitive information. Fortunately, blockchain technology offers the perfect environment for creating mutual session keys between IoT devices. Generating public keys using elliptic curves [6] [7] offers a lightweight solution for creating mutual keys. The elliptic curve offers similar security compared to its counterparts while having a key at a fraction of the bits. For comparison, a 224-bit Elliptic Curve Cryptography (ECC) key has roughly the same security as a 2048-bit RSA key. For this reason, IoTs with low memory and processing power can still create secure keys. Once a key is generated, the public key is appended to the *PublicKey* data field along with the designated IoT recipient and mined onto the blockchain.

Since secure data transmission via blockchain is not recommended due to the public nature of a broadcasted blockchain on a server, a blockchain-based public key exchange allows for IoTs to set-up Non-Interactive Key Management Protocols [8]. With a Non-Interactive Protocol, session key rekey sequences using the various blockchain data fields as 'salt' (ex. IoT devices using the last 8 bits of the hash of every odd block on the blockchain for the past 31 blocks) could offer an efficient solution to updating IoT session keys for secure data transmission, but more research is needed in this area.

VII. Analysis

A. The Difficulty Parameter and Mining Complexity

Computing a valid hash as defined by the difficulty parameter is a process that takes exponential time to compute. This is shown as follows, with *b* as the numeral system, *n* as the hash output length of base *b*, and *d* is the difficulty parameter:

$$b = 16 \qquad (1)$$

$$\text{n = hash output length of base b} \qquad (2)$$

$$\text{d = difficulty parameter} \qquad (3)$$

$$d \leq n \qquad (4)$$

$$A(n) = b^{d} \qquad (5)$$

Where Eq. (1) represents the numeral system used (ex. Hexadecimal) and Eq. (2) represents the hash output length in terms of the chosen numeral system. Eq. (3) specifies that Eq. (4) states that a difficulty parameter exceeding the hash output length cannot exist in terms of our difficulty parameter definition (see section III, part D) as this would be impossible. Eq. (5) specifies that the average number of iterations an algorithm needs is the numeral system *b* raised to the number of characters in the target sub-string *d*. If the numeral system remains fixated, as the difficulty parameter increases to accommodate more computing power on the blockchain network, the number of calculations increases exponentially.

After implementing the blockchain framework in Java, the average number of iterations per difficulty parameter was extracted and shown in Fig. 4. The data collected from our blockchain framework implementation in Java supports the conclusion that mining a block on a blockchain takes exponential time as our test blockchain implementation follows very closely to the Eq. (5).

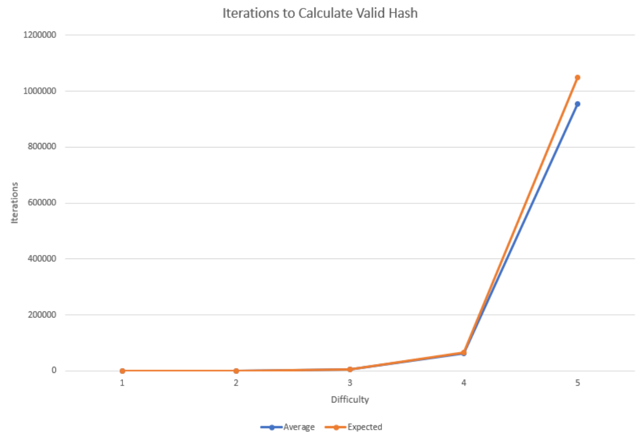
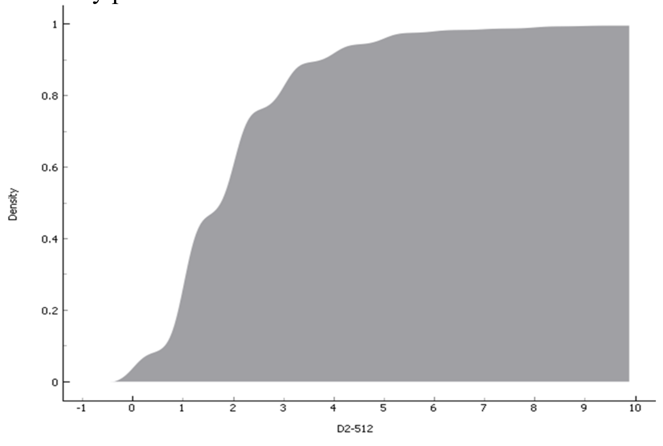
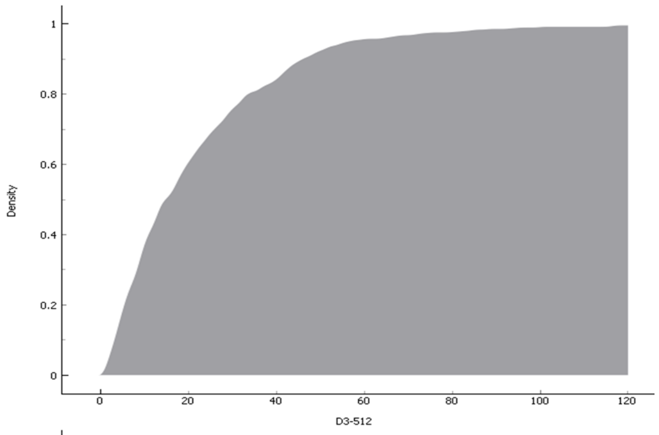


Fig. 7. Using our implementation of the IoT blockchain framework in Java to generate 500 blocks for the difficulty parameter ranging from 1 to 5, an average of the number of iterations per difficulty was taken and the results are shown above. The blue line is the average number of iterations per difficulty for our test blockchain. The orange line is the average expected number of iterations as defined by Equation 5.

B. Hashing Dispersion Analysis

The following Fig. 5 presents the data distributions for various difficulty parameters. The following distributions reflect the average time complexity of mining a single block using the SHA-512 hashing algorithm. As shown in Fig. 5, the time complexity distribution for mining blocks with the SHA-512 hashing algorithm was collected for various difficulty parameters.





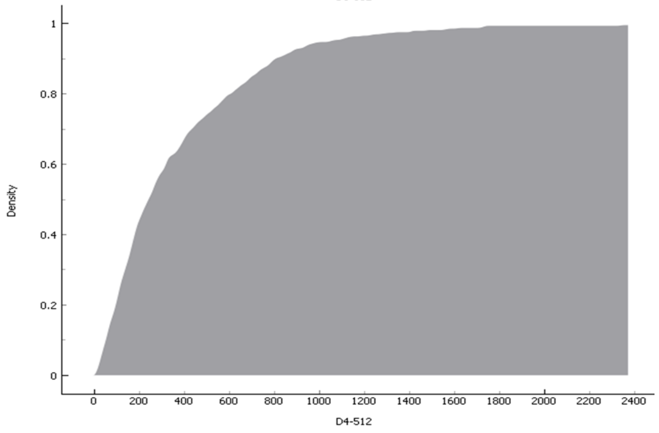


Fig. 8. Comparison of the time complexity dispersion of generating 500 valid blocks using our Java blockchain implementation with the SHA-512 hashing algorithm for difficulty parameter 2-4. The horizontal axis represents the time complexity to compute a single block in milliseconds and the vertical axis is the density of time complexity. Figure 5.1 represents the dispersion for difficulty parameter 2. Figure 5.2 depicts the dispersion for difficulty parameter 3. Figure 5.3 depicts the dispersion for difficulty parameter 4. A graph for difficulty parameter 1 was omitted due to the fast hash calculations taking less than or equal to 1 millisecond.

For difficulty parameter 2 as shown in Fig. 5.1, the graph is unimodal with a mean of around 2 milliseconds. As shown in Fig. 5.2 and 5.3 the difficulty parameter 3 and 4 represent a right-skewed distribution with a median of 20 and 351 respectively.

From the graphs of Fig. 5, the distributions become more right-skewed, or it has a longer tail in the positive direction on the number line. In terms of security, this right-skewed distribution is important because it shows that the block time complexity will be greater and more varied after the mode. This means outliers are likely to fall after the mode, rather than before, which shows that a single hash calculation will likely take a longer time to compute, compared to a unimodal distribution.

Fig. 6 below presents the box plot for difficulty 4. The first quartile is 108 milliseconds. The third quartile is 511 milliseconds. The mean is 351 milliseconds and the median is 235 milliseconds. The minimum is 1 millisecond, and the maximum is 2340 milliseconds.

As seen in Fig. 6, the box plot of the 4th difficulty shows the wide dispersion of the 500-time complexities collected when using the SHA-512 algorithm. Fig. 6 visually shows that 50% of the hashes had a time complexity range of 1 to 235, which is a tight grouping. However, the other 50% of hashes had a time complexity range of 235 to 2340, which is a wide grouping.

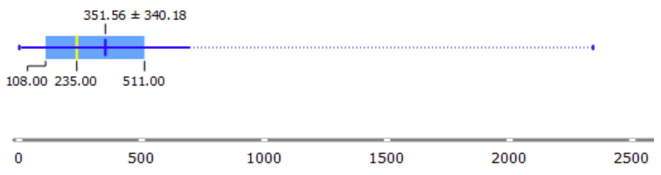


Fig. 9. Box plot for difficulty 4 (see Figure 5.3).

The index of dispersion is a calculation used to normalize the dispersion of a probability distribution. This index was calculated using the Orange analysis tool for difficulty 2, 3, and 4 using the following formula, where x_i is a data point, $\overline{x}$ is the mean, N is the number of elements in the data set, σ is the sample variance, and D is the index of dispersion:

$$\sigma = ((\sum(x_i - \overline{x})/(N-1)) \tag{6}$$

$$D = \sigma^2/\overline{x} \tag{7}$$

Orange automates the calculation of the index of dispersion of Eq. 7 using Eq. 6. Eq. 7 was calculated for difficulty parameters 2, 3 and 4 using the data set shown in Figure 5. Difficulty 2, 3, and 4 have an index of dispersion of 0.73, 0.93, and 0.97 respectively. As the difficulty increases, the index of dispersion increases. With an increasing dispersion, and a mean higher than the median, valid hashes are more secure at higher difficulties not only because there are more matching characters between the hash and target string, but also because the of the right-skewed nature of the hashing algorithms at higher difficulties and the increasingly large dispersion of the time complexity for calculating a single hash.

VIII. Hashing Algorithm Comparison

Choosing the right hashing algorithm for the blockchain is especially important when dealing with low memory and low power IoT devices. Another problem that may arise is having a congested network with too many IoTs making requests at the same time. This results in a large exchange of bits which may slow down or stop the network entirely depending on the network bandwidth, blockchain application, and publicity of the blockchain. A congested network with unfulfilled commands will leave the end-user unhappy.

Hashing complexity was observed during the runtime of the blockchain program in terms of the iterations and run-time in milliseconds for the hashing of a block. Specifically, data analysis for the average block computation time was conducted. In Fig. 7, 500 blocks were generated and mined for various Difficulty parameters (1, 2, 3 and 4) for some of the most common hashing algorithms such as SHA-224, SHA-256, SHA-512, SHA-384. The data was captured using a Java implementation on an i7 Windows 10 machine. As evident in the figure, the time complexity of mining a single block with increasing difficulty is of exponential order, despite the hashing algorithm. As the hashing algorithm produces more output bits, the time complexity exponentially increases to mine a valid hash as defined by the difficulty parameter. There is a trade-off between the hashing output length and security. For IoT blockchain applications, smaller hash sizes result in fewer data stored in each block. However, smaller hashes are easier to compute, which allows for an attacker to modify the blockchain easier. Smaller-output hashing algorithms require a larger difficulty to compensate for the decreased time complexity, while storing fewer data.

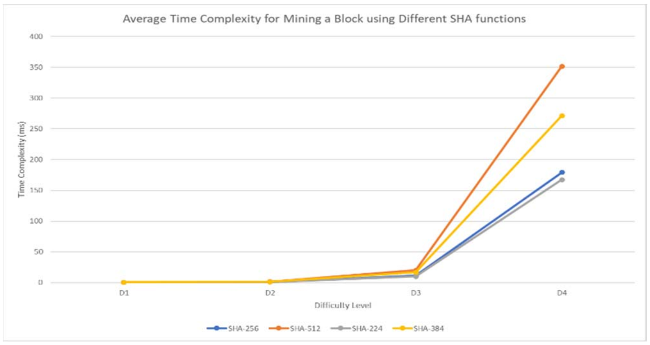


Fig. 10. Comparison of the blockchain mining time complexity

IX. Conclusion

In this paper, we discussed a possible blockchain framework that allows for simple public key exchanges, blockchain parameter updates, and public key updates to retain a secure and mutually authenticated network blockchain. This framework also attempts to improve upon existing frameworks with a more secure difficulty parameter, and it also offers suggestions for data-conscious applications in storing important blockchain parameters on lightweight devices. The paper also analyzes the hash mining process through the application of our proposed framework in Java. The mining complexity was represented as an equation in terms of the difficulty parameter compared to test results. The hash algorithm dispersion was analyzed for each major difficulty parameter, and the results showed a right-skewed dispersion and an increased index of dispersion, which was correlated with a higher level of security. Additionally, an analysis of varying hashing algorithms from the SHA-3 library for varying difficulties showed expected run-time results. Based on the data generated from our test blockchain framework, we propose that a faster hashing algorithm combined with a higher difficulty yields the most secure and data-conscious solution to the hash mining security.

Acknowledgment

We thank CyberFlorida for funding this research work with the grant # 3910-1004-00-D.

References

[1] M. Samaniego and R. Deters, "Blockchain as a Service for IoT," 2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Chengdu, 2016, pp. 433-436.

[2] M. Singh, A. Singh and S. Kim, "Blockchain: A game changer for securing IoT data," 2018 IEEE 4th World Forum on Internet of Things (WF-IoT), Singapore, 2018, pp. 51-55.

[3] D. Fakhri and K. Mutijarsa, "Secure IoT Communication using Blockchain Technology," 2018 International Symposium on Electronics and Smart Devices (ISESD), Bandung, 2018, pp. 1-6.

[4] P. Urien, "Blockchain IoT (BIoT): A New Direction for Solving Internet of Things Security and Trust Issues," 2018 3rd Cloudification of the Internet of Things (CIoT), Paris, France, 2018, pp. 1-4.

[5] Mughal, Muhammad Arif, "A Lightweight Digital Signature Based Security Scheme for Human-Centered Internet of Things." *IEEE Access*, vol. 6, 6 June 2018, pp. 31630–31643.

[6] D. P. Shah and P. G. Shah, "Revisting of elliptical curve cryptography for securing Internet of Things (IOT)," *2018 Advances in Science and Engineering Technology International Conferences (ASET)*, Jun. 2018.

[7] J. R. Shaikh, M. Nenova, G. Iliev, and Z. Valkova-Jarvis, "Analysis of standard elliptic curves for the implementation of elliptic curve cryptography in resource-constrained E-commerce applications," *2017 IEEE International Conference on Microwaves, Antennas, Communications and Electronic Systems (COMCAS)*, Nov. 2017.

[8] L. Celia and Y. Cungang, "(WIP) Authenticated Key Management Protocols for Internet of Things," *2018 IEEE International Congress on Internet of Things (ICIOT)*, Jul. 2018