Blockchain-based Security for Heterogeneous IoT Systems

Kale Yuzik
Department of Computer Science,
University of Saskatchewan
Saskatoon, SK, CANADA
kay851@usask.ca

Dwight Makaroff
Department of Computer Science,
University of Saskatchewan
Saskatoon, SK, CANADA
makaroff@cs.usask.ca

ABSTRACT

The Internet of Things (IoT) is being deployed in industry, public services, and even homes. These devices are making information more available and allow for greater automation and efficiencies. With the rapid growth this industry is experiencing, the security of IoT devices has not been given the attention it needs. Many of these devices leave sensitive information exposed or may allow for malicious actors to take control of them. The Internet of Things uses a vast range of hardware which has led to many different approaches to security. Administering a network with such variability makes it easy for insecure configurations to be overlooked.

This paper proposes the use of blockchain technology as the backbone to a security framework to unify IoT devices of varying resource constraints under one system. Ethereum is used to create a secure system that is Denial of Service resistant, store encryption keys, store encrypted data, and manage trust of devices. Using the Proof-of-Authority consensus method instead of the more common Proof-of-Work, allows for more efficient use of resources. This system features mechanisms to include the use of LoRa LPWAN technology, which is often used in IoT. Tests were run on a small network of devices while recording processor utilization. Latencies were also measured, showing that devices with fewer resources showed significant latencies, and suggestions as to how these latencies can be reduced are proposed.

CCS CONCEPTS

• **Information systems** → *Information systems applications*; • **Computer systems organization** → **Peer-to-peer architectures**; • **Security and privacy** → **Key management**; **Security services**; • **Networks** → *Network services*;

ACM Reference Format:
Kale Yuzik and Dwight Makaroff. 2020. Blockchain-based Security for Heterogeneous IoT Systems. In *Proceedings of CASCON 2020 (CASCON'20).* ACM, New York, NY, USA, 10 pages.

1 INTRODUCTION

The Internet of Things (IoT) is experiencing a rapid expansion in growth. ARM predicts that one trillion IoT devices will be manufactured between 2017 and 2035, and world will see a \$5 trillion boost in G.D.P. due to the industrial use of IoT technologies by 2035 [27]. The Internet of Things offers great value for uses such as monitoring critical infrastructure, which will inevitably lead to the deployment of these systems throughout cities. Manufacturers are driven by economic factors and those fastest to market benefit the most. This encourages manufacturers to cut corners and take calculated risks and there is no exception when it comes to the security of these products. Many of these IoT devices are deployed in remote or inaccessible locations and use low bandwidth connections. This makes servicing or updating them far more challenging than conventional computer networks. As the use of IoT systems expands, the risks involved with failure or security breaches become increasingly severe.

IoT traffic lights have been developed to synchronize with other traffic lights within road networks to minimize delays and reduce congestion [1]. While the benefits of smart road infrastructure are considerable, if targeted by an attacker, traffic collisions could be caused, putting lives at risk. Commercially available, internet connected cardiac implants were found to contain a critical security vulnerability [28]. This not only exposed data collected by the implants, but the administration of shocks by pacemakers and defibrillators could be altered. This documented vulnerability is irrefutable evidence that with the growing adoption of IoT technologies the benefits are immense, but the cost of breaches will be financially expensive and may endanger lives. For these reasons, it is critical these systems be secure at the time they are deployed.

Current approaches to IoT networks employ cloud-based services to collect and process data from IoT devices. These cloud-based IoT services (such as The Things Network[1]) introduce a single point of failure by means of an external agency. Should the cloud service become compromised, all guarantees of data confidentiality, integrity, and availability are lost. This exposure may be acceptable for some applications, but for critical services for which society may come to depend upon, minimizing/eliminating these exposures is vital. A blockchain-based security framework is proposed to address these issues with cloud-based IoT services.

Due to the inexpensive and low-power hardware used for IoT systems, five categories of constraints apply: compute power, memory capacity, persistent storage capacity, connectivity bandwidth, and power source. Some limitations that may exist for one IoT device may not be an issue for another. Given this broad range of devices, the question of how to design a security framework that caters to the needs of these heterogeneous devices arises. Using dissimilar solutions for devices of varying hardware resources is not only cumbersome, but introduces security concerns itself. With more solutions come greater potential for configuration errors and complexity of administration.

[1]https://www.thethingsnetwork.org/

This paper explores the application of blockchain technology to create a unified security framework for IoT devices with heterogeneous compute resources. The remainder of this paper is organized as follows. Section 2 describes the component technology of the problem domain, while Section 3 gives a brief overview of similar previous work. Section 4 outlines the implementation and configuration of the test environment, Section 5 provides proof-of-concept results for the test network, and Section 6 draws some insight and analysis. Section 7 provides a summary and suggests future directions.

2 BACKGROUND

When assessing information security there are three fundamental qualities that must be considered. These qualities form what is referred to as the CIA triad: Confidentiality, Integrity, and Availability [18]. Confidentiality refers to the secrecy of data from those who are not authorized to view or access it. Data can be kept confidential using cryptographic techniques. Integrity is an assurance that the data can neither be altered or forged. The integrity of data can be protected through the use of digital signatures. These signatures provide means to authenticate the origin of the data as well as detect if the data has been altered. Availability is the property that adversaries cannot prevent or hinder access to data or services required to process/transmit/receive that data. This can be guarded using peer-to-peer technologies to provide redundancy, thereby increasing the availability of data.

Decentralized technologies such as blockchain present a unique advantage over the traditional client-server model. They offer a resistance to Denial of Services (DoS) and Distributed Denial of Service (DDoS) attacks [19], owing to the lack of a single point of failure [2] and distributed ledger containing the desired data. Cisco has projected 15.4 million DDoS attacks will occur in 2023, nearly double the 7.9 million which were expected in 2018 [12].

2.1 Blockchain

Dwork and Naor [15] first introduced the idea of Proof-of-Work, a way of providing means for making an assertion without the need of cryptographic trust, a precursor to Blockchain. Vishnumurthy *et al.* [29] made use of the concept of Proof-of-Work by creating a credit system to incentivize equal contribution of all nodes within peer-to-peer systems. This system provided a public ledger of transactions and involved the payment of "karma", a digital token for work performed by peers. Nakamoto [24] developed the idea of a decentralized, anonymous digital currency, now known as Bitcoin.

Blockchain is essentially a distributed database [21] that consists of chunks of data (blocks) that are linked together in a linear order. Each block contains the cryptographic hash of the block prior [24].

In the Proof-of-Work (PoW) consensus scheme, miners assemble a block with pending transactions. A miner assigns an arbitrary value to the nonce (number used once) field and calculates the hash of this proposed new block. The miners then check if the hash is less than the difficulty value [3]. When a miner succeeds, it broadcasts its newly mined block to connected peers, who then verify its validity. If valid, the network accepts the block, and work begins on the next block. Miners race with others to find values which satisfy these criteria. The difficulty is adjusted to maintain a predetermined duration of time between creation of new blocks, which is called the "block time".

A change in any block along the chain will result in one of these hashes not matching. For an attacker to successfully alter an existing portion of a blockchain, they must re-mine every block from the victim block on until the length of their altered blockchain exceeds the length of the currently accepted chain.

Finding a hash which meets the required difficulty parameter involves continual computation [3], and because mining is a race for the next block, it is only viable on hardware above a threshold of computational power. For this reason, Proof-of-Work is an impractical solution for securing a blockchain running on a network of low power IoT devices.

As an alternative to Proof-of-Work consensus, a voting-based system known as Proof-of-Authority (PoA) [13] may be used, in which blocks are approved (or rejected) by authorized accounts known as signers. The use of a PoA consensus algorithm creates a permissioned blockchain, whereas with PoW the blockchain would be permissionless. De Angelis *et al.* [13] analyzed permissioned blockchain consensus algorithms in terms of the CAP (Consistency/Availability/Partition tolerance) theorem [16] and performance. The implementations of PoA known as Aura and Clique were examined, as well as *Practical Byzantine Fault-Tolerant* (PBFT) schemes. While there were trade-offs in terms of the CAP theorem, Clique requires the least number of messages to achieve consensus, thereby making it advantageous for use on resource constrained systems.

On PoA, signers approve blocks by signing them with their cryptographic key and for a network to consider a block as valid, it must be signed by a majority of the authorized signers. Upon genesis of the blockchain, initial signers are defined. Accounts which maintain the transaction process of the blockchain accumulate positive reputation. Thus, signers can be voted in or out, based on their reputation within the blockchain network. This system eliminates the computationally demanding operations required by the Proof-of-Work scheme. Additionally, PoA allows for the block time to be explicitly set, thus allowing for some degree of control over the latency of contract functions which mutate the contract state and by extension, the latency in our proposed system.

2.2 Ethereum and Smart Contracts

Blockchain is best known for its use in implementing cryptocurrencies, but its applications are far more broad. Smart contracts are compiled code that is uploaded to the blockchain [30, 31]. These contracts contain functions that may be executed in a distributed manner as required. Contracts can contain persistent state information that is global to all devices on the blockchain. In order for the results of contract execution to be accepted by the network, there must be consensus on the postconditions of execution.

Smart contracts, as they are referred to in the context of Ethereum, contain functions which are divided into two groups: those that modify the contract state and those that do not. They have substantially different performance properties. Contract functions modifying the contract's state are called by sending a transaction [30]. This is done by broadcasting the transaction data to other devices mining on the blockchain, for which the outcome state of the contract must be agreed upon by the miners. The mining nodes execute the function and must come to a consensus, introducing a latency which is primarily dependent upon the block time. Contract functions that do not modify the state variables of a contract need only be executed locally on the device. There is no need to come to a consensus on the result of this computation, as it does not modify public information in any way, so the latency is imperceptible.

The Ethereum Virtual Machine (EVM) is used to execute smart contract functions. EVM allows for looping, and thus introduced the possibility of poorly written or malicious code to invoke an infinite loop. As a remedy, the concept of "gas" is introduced. Each instruction depletes a finite quantity of gas allotted to a transaction. The sender of a transaction may choose the initial amount of gas available; the spent gas determines transaction fees charged to the account from which the transaction originated.

Ethereum accounts each possess their own pair of cryptographic keys that are used to sign transactions. These same keys are used to sign blocks when using Proof-of-Authority consensus.

2.3 Cryptography

Symmetric cryptography uses the same key to both encrypt and decrypt information for both parties, whereas asymmetric cryptography involves both actors having different cryptographic knowledge and abilities [17]. Asymmetric ciphers are more computationally expensive and a solely asymmetric approach to encryption is not feasible in many domains. Algorithms such as Diffie-Hellman allow for a shared key to be derived only from knowledge of one's own key-pairs and the partner's public key, preventing third parties from determining the shared key. Another unique advantage with asymmetric cryptography is the possibility for data to be digitally signed and verified [17], providing a very high degree of confidence that the data originated from the believed source (the actor which possesses the specific key-pair).

Both Rivest-Shamir-Adleman (RSA) and Elliptic Curve Cryptography (ECC) are asymmetric cryptographic systems. They both provide the same functionality, but differ in underlying mathematics, computation difficulty, and security [22]. As keys become larger, the security the cipher provides increases. When an ECC key becomes larger, RSA keys must grow at a disproportional rate to be able to match the level of security [20]. ECC can offer an equal level of security with a much shorter set of keys.

The Advanced Encryption Standard (AES) symmetric cipher has been heavily used since its acceptance by NIST in 2001 [25]. AES uses keys of either 128, 192, or 256 bits, with 10, 12, and 14 rounds respective of key length. There have been limitations and shortcomings identified with AES in the intervening years. A cache timing-based attack [6] on AES exposed the possibility of key recovery. This not only breaks the confidentiality of the current ciphertext, but all other messages that are encrypted with the same key.

The Salsa20 stream cipher [9] offers encryption that is consistently faster than AES. Salsa20 may be applied using a differing number of rounds, with Salsa20/20 (20 rounds) being the recommended standard. Cryptanalysis of Salsa20 has shown Salsa20/8 or fewer rounds to be vulnerable to attacks [5, 9].

The Salsa20 family of ciphers uses 3 operations: 32 bit addition, 32 bit XOR, and constant-distance 32-bit rotation. These instructions are all CPU friendly, and therefore faster across a wider number of platforms than other ciphers such as AES [9]. Due to the lack of S-box lookup tables, Salsa20 also avoids the cache timing attacks possible with AES.

XSalsa20 specifies a longer nonce than Salsa20 (128 bits vs. 64 bits) [8]. The nonce does not need to be secret; a third party obtaining the nonce does not compromise security of the cipher. The longer nonce makes it safe to use a randomly-generated nonce. XSalsa20 offers the exact same speed as Salsa20, with the minimal extra cost of generating the larger nonce.

A cipher with a higher degree of diffusion does a better job in hiding the relationship between plaintext and ciphertext. The family of ChaCha ciphers [7] is based on the Salsa cipher and provides improved diffusion. This modification does not increase the computational expense, nor does it reduce the potential for parallelism. In fact, ChaCha20 uses one fewer register than Salsa20, which on some platforms may yield minor performance gains. Aumasson *et al.* [5] performed a differential cryptanalysis of Salsa20 and ChaCha. They found that while they could break up to 8 rounds of Salsa20, they were only able to break up to 7 rounds of ChaCha (ChaCha7). For symmetric encryption, XChaCha20 was selected due to its improved strength against cryptanalysis over other variants in the Salsa20 family, its imperviousness to side channel attacks, and CPU friendly operations which allows for efficient operation on embedded systems.

3 RELATED WORK

Biswas and Muthukkumarasamy [10] conducted an analysis of smart cities and how blockchain technology could be used to provide a security framework to protect them. These researchers point out that IoT devices used in smart cities utilize various communication layer technologies such as Ethernet, Wifi, Bluetooth, 6LoWPAN, 3G, and 4G. They argue a security framework should support these technologies and allow for communication between differing communication systems. The recommendation for use of a permissioned blockchain was made over an permissionless blockchain, due to faster consensus and reduced potential for anonymous attacks.

Huh, Cho, and Kim [19] proposed an Ethereum-based system for managing RSA public keys as an IoT management system. Their proof of concept modelled electrical appliances and monitored power consumption. Smart contracts provided an interface to set a power usage limit when the devices would be automatically turned off.

Özyilmaz and Yurdakul [26] investigated an Ethereum-based IoT data collection system. Wireless nodes used LoRaWAN to communicate with a "smart proxy" that performed blockchain-related functions. This work focused on blockchain technology for decentralized storage and robust data availability, but did not employ cryptography to ensure data confidentiality. Data was stored using Ethereum's SWARM storage service, a peer-to-peer data storage system. Many of the design aspects of Özyilmaz's work will be used in the formulation of the system in this paper.

Minoli *et al.* [23] conducted an analysis of blockchain technology in the scope of providing security for IoT. Proposals were made for the different roles a "Network Element" (NE) may serve in the greater scope of the network/blockchain. Some of these configurations defer protection of data integrity to other more powerful devices within a network to account for NEs which may be less capable of securing the integrity on their own. These devices include gateways, and concentration nodes (routers, switches, firewalls, etc.). Additional uses for blockchain systems for IoT are also suggested, including device configuration, data storage, micro-payments, automated payments between things to create a shared economy, Digital Rights Management (DRM), history of ownership throughout the supply chain, smart cities, device communication/synchronization, and software rollout.

Dorri *et al.* [14], examined the design of the blockchain itself in the context of smart home IoT. The authors highlighted barriers to using cryptocurrency-based blockchain systems with IoT systems. These issues include high consumption of system resources, latency, and scalability problems arising from the need for consensus among nodes. A layered design of the network is proposed, involving no need for use of Proof-of-Work. In one layer, a private blockchain is used to connect a group of devices within a home. One device in the home with plenty of computational resources is designated the Smart Home Manager (SHM), which acts as the miner. At the top layer, smart homes are connected to a public blockchain (independent from the private blockchain), for which the SHM relays transactions, and communicates with cloud-based services. This separation of blockchains greatly reduces the storage needs of the resource constrained IoT devices, as well as reduces the bandwidth and energy demands placed on them.

4 METHOD

Ethereum will be used as the underlying blockchain technology due to the Turing complete virtual machine it makes available for distributed computation. While other blockchain technologies such as Bitcoin also make scripting possible, these alternatives are not Turing complete [11, 19], as looping is not possible. This design choice greatly limits their practicality for use in our framework.

The resource constraints of the IoT devices restrict our design parameters. In order to encompass this range of devices into one system, a proxy is built into the design. This allows devices that are incapable of running an Ethereum client to participate in the network. The programming language used must allow for efficient use of hardware, and allow for multiple threads to make best use of resources. We chose C++ with a custom system to manage communication with our selected Ethereum client (Geth, see Section 4.2) and its JSON API through Unix domain sockets, as the commonly used Web3.js library is written for JavaScript.[2] This keeps resource consumption as low as possible.

A conceptual overview of the proposed system is found in Figure 1. Three different types of devices exist: devices running a Geth client, without LoRa (Section 4.5.1), devices running a Geth client and operating as a LoRa proxy (Section 4.5.2), and devices not running Geth with LoRa, requiring the services of a proxy (Section 4.5.3).

[2]https://web3js.readthedocs.io/en/v1.2.6/

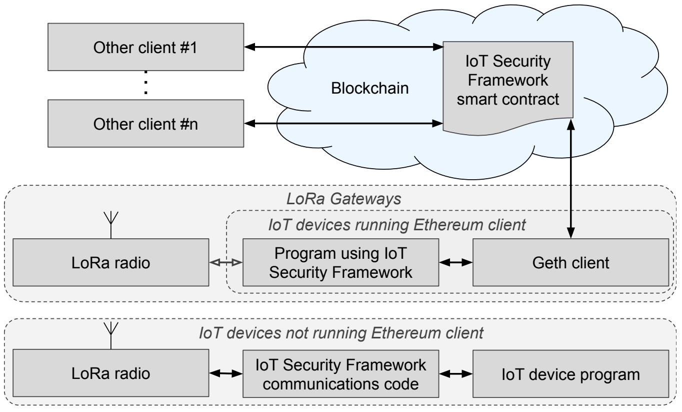
Other client #1
Other client #n
Blockchain
IoT Security Framework smart contract
LoRa Gateways
IoT devices running Ethereum client
LoRa radio
Program using IoT Security Framework
Geth client
IoT devices not running Ethereum client
LoRa radio
IoT Security Framework communications code
IoT device program

Figure 1: Architecture of the network

4.1 Design Considerations

4.1.1 Compute Power. Embedded systems generally possess low compute power. This amplifies the trade-offs when selecting cryptographic algorithms. The trade-offs between computational latency and security become far more pronounced than on devices with faster processors. Security will be prioritized when reasonable, while minimizing computational complexity.

4.1.2 Memory and Storage. On some devices, memory and storage become severely limited, in some cases as low as tens of kilobytes. Offloading much of the work to a proxy/gateway will minimize the memory footprint of the compiled binary for these platforms.

4.1.3 Network Bandwidth. Many IoT devices use wireless communications to perform their functions. One such common technology is LoRa [4]. LoRa allows for throughput ranging from 0.3 Kbps to 50 Kbps, depending on configuration and regional differences [21]. The proxy solution must operate over these low bandwidth connections, while maintaining a high degree of security.

4.1.4 Power Source. Very often, IoT devices have limited power supply such as batteries or solar power. Both the processor and wireless radios can be significant consumers of energy; minimizing power consumption is important for the feasibility of a solution.

4.1.5 Cryptographic Functions. The required cryptographic functions consist of public/private key generation, Diffie-Helman key exchange, a symmetric key cipher, and signature creation and verification. ECC was selected over RSA, due to both its smaller key size without compromised security and computational speed, which is well suited to embedded systems [22].

4.2 Ethereum

The blockchain security framework was tested on a private Ethereum network, using Proof-of-Authority as the method of consensus. Using PoA over PoW allows for more devices to participate in the voting process, as compared to the mining process in PoW. This makes the security of the blockchain dependent on the quantity of signers, rather than the mining compute power. In general, this lends itself well to a network of IoT devices, since such networks often consist of hundreds or thousands of devices. Additionally, control over block time is an advantage since this will directly impact the latency of data sent by devices on the network.

A block time of 5 seconds was used, as it provides lower latency than the block time of 12 seconds used on Ethereum's public blockchain which uses PoW consensus. While 1 second would result in even lower latency, the rate at which storage costs grow must also be weighed. A test was carried out with an Ethereum blockchain, a 5 second block time, 1 signer, and no transactions being made. The size of chain data on the filesystem was recorded at 5 second intervals, which showed the chain grew at an average rate of 3465 bytes per block.

4.3 Go Ethereum Client

We selected the Go Ethereum client (Geth).[3] The implementation of Proof-of-Authority used in Geth is known as Clique. Geth provides many options and modes which allow for control over the extent that the blockchain is stored and verified locally. These settings allow for some adjustment over the use of system resources, such as processor, memory, storage, and bandwidth. It has three different modes of operation/communication: full sync, fast sync, and light sync.

In **full sync** mode, Geth stores the entire blockchain on the device and verifies every block created and transaction contained within the blocks. This is the most resource demanding mode of the three. As the blockchain adds blocks to the chain, the costs of storing the chain increases. **Fast sync** mode, like full sync mode, obtains all blocks since genesis and verifies all blocks, but does not verify transactions, until a set number of blocks behind the present head of the blockchain.[4] This mode trades some processing power for bandwidth. Once a fast sync client has obtained the entire chain, it functions the same as full sync mode. **Light sync** mode consumes the least amount of system resources, with the exception of bandwidth. In this mode, all block headers and data are downloaded, but transactions are not obtained. Geth only randomly validates blocks in light sync mode. The use of light mode requires full sync mode devices on the network to serve light clients which must be explicitly enabled on the full sync client.

4.4 Contract Design

The smart contracts will be used to store the following information for each device:

- Human friendly name,
- Numeric ID ("device ID"),
- Device creation timestamp,
- Public encryption key,
- Public signature key,
- Encrypted data & nonce,
- Timestamp of when data was last received,
- Numeric ID of the decrypting device ("data receiver"), and
- whether this device is managed by a gateway/proxy (T/F) ("gateway managed").

[3]https://geth.ethereum.org/

[4]Szilágyi, Péter. October, 2015. eth/63 fast synchronization algorithm #1889. https://github.com/ethereum/go-ethereum/pull/1889

The contract facilitates the allocation of new devices, removal of devices, changing of cryptographic keys, and storage and retrieval of encrypted data. Some of these are administrative functions that should only be callable by authorized Ethereum accounts. The contract allows for an arbitrary number of Ethereum accounts to be granted access to call such functions.

Each device is assigned a partner device which may decrypt the sender's data, termed a "data receiver". Allowing for a specific data receiver to access data from one or more devices permits data privacy even with multiple users within a blockchain security framework. This limits potential damage if a cryptographic key becomes compromised.

4.5 Devices

The hardware utilized in the test network consists of 3 types Raspberry Pi devices and AdaFruit Feather M0 devices[5] as described more fully in Table 1. In addition to the information in the table, the Raspberry Pi 2B+s are equiped with a Dragino LoRa (SX1276) & GPS HAT. One AdaFruit Feather M0 device uses a 1200mAh LiPo battery and the other devices are mains powered. All LoRa chips are of the RF9X family, operating in the 915MHz frequency range. All Raspberry Pis ran Raspbian on a headless installation. The devices were run in a network as illustrated in Figure 2.

4.5.1 *Devices Running Geth Client, without LoRa.*

Devices which are connected to the Internet and have sufficient system resources will run the Go Ethereum client locally. The blockchain security framework will communicate with Geth through Unix domain sockets to request services of the smart contract. Only devices with more capable hardware will run Geth in full sync mode as a signer. Other devices will be tested in light sync mode.

4.5.2 *LoRa Gateways/Proxies.*

Devices operating as a LoRa gateway constantly listen for transmissions from broadcasting LoRa devices and run a local instance of Go Ethereum. When the gateway receives an incoming message, it retrieves the public signature key that corresponds to the device ID in the LoRa packet from the smart contract. If the signature is verified as valid using the public key, the gateway can be confident the message originated from the claimed device. Since the gateway is already registered on the blockchain, the smart contract implicitly trusts the gateway and the gateway may forward the already encrypted message payload to the blockchain. In order for the smart contract to permit the gateway acting on the behalf of the device, the device must be registered on the blockchain as "gateway managed". Any gateway is capable of pushing the data of any registered LoRa device to the blockchain. This allows for geographic mobility of these devices.

4.5.3 *LoRa Connected Devices.*

Systems using LoRa for connectivity do not possess the bandwidth necessary to run Go Ethereum locally. These devices may also lack other resources to run Go Ethereum. The Adafruit Feather M0 meets none of these requirements, as is true for a large portion of IoT devices; mechanisms for this category of device must be included.

Since these devices cannot run Go Ethereum, this must be done by proxy/gateway. A protocol was created to allow for a LoRa

[5]https://www.adafruit.com/product/3178

Table 1: Test Devices (all ARM CPUs)

Device	CPU	Cores	Frequency (MHz)	RAM	Network
Raspberry Pi 2B+	Cortex-A53	4	900	1 GB	WiFi and LoRa
Raspberry Pi 4B	Cortex-A72	4	1500	4 GB	WiFi
Raspberry Pi Zero W	117676JZF-S	1	1000	512 MB	WiFi
AdaFruit Feather M0	Cortex M0 ATSAMD21G18	1	1000	32 KB	LoRa module (SX127X)

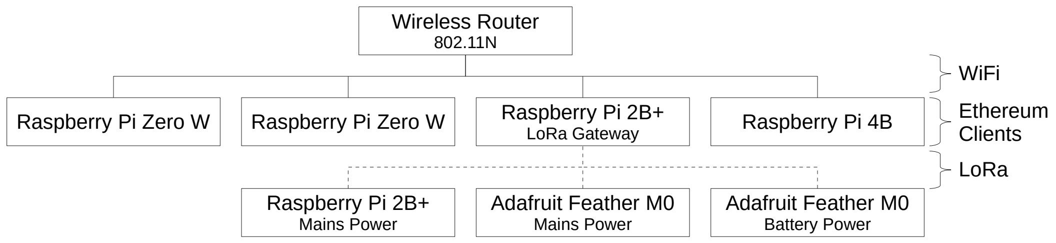

Figure 2: Configuration of test network

gateway to act upon a device's behalf, while maintaining data confidentiality and integrity. Devices communicating over LoRa must be authenticated and cannot be implicitly trusted. Our protocol detects forged/altered data and prevents eavesdropping as described in Section 4.6.

The custom LoRa protocol allows for a payload of up to 154 bytes which is three times larger than LoRaWAN and by extension, The Things Network. The packet structure is as follows:

- Source and destination device ID (4 bytes each),
- Message ID (1 byte),
- Packet fragment number (1 byte),
- Flags (1 byte),
- Reserved (1 byte),
- Data length (1 byte),
- Message signature (64 bytes),
- Cryptographic nonce (24 bytes), and
- Encrypted data (max 154 bytes).

The "message ID" and "packet fragment number" are presently unused, but are left in for future versions, to enable fragmented messages, similar to packet fragmentation in IPv4.

When a LoRa device boots, it pre-calculates the shared key used to encrypt data for its data receiver. This key is stored to avoid having to recompute it, wasting processor cycles and power. Once data is ready to be sent, it is encrypted (Section 4.6) and encapsulated in a packet that is then digitally signed and transmitted.

4.6 Cryptography

The libSodium[6] library provides the desired algorithms and supported all but one of the platforms being used for testing. Minor changes[7] were required to port the library to the ARM Cortex M0.

Since embedded LoRa devices do not run a local Go Ethereum client, an external cryptography library will be used for digital signatures in addition to key generation, key exchange, and symmetric encryption. Digital signatures will utilize the Edwards-Curve Digital Signature Algorithm (ECDSA) using *edwards25519* parameters. This creates a 512-bit signature that the LoRa gateway can verify to ensure the authenticity of the sender. Should the message have been altered or corrupted after it is signed, it is discarded. libSodium's *crypto_sign_init(), crypto_sign_update(), crypto_sign_final_create()* are used to create a signature and *crypto_sign_final_verify()* to verify a signature.[8]

Before data can be encrypted, the shared key must be computed between the device and its data receiver using Elliptic Curve Diffie-Hellman key exchange (ECDH). Once the shared key has been determined, it is used with the symmetric XChaCha20 stream cipher to encrypt the data being transmitted. A 192-bit, randomly generated nonce is used during encryption and must also be transmitted and stored on the blockchain. This nonce itself does not need to be kept secret, but is required for decryption.

Data is encrypted on an end-to-end basis. Before a LoRa device transmits data to a gateway, it is both encrypted and signed. Upon receipt, the gateway verifies the signature. The gateway does not decrypt the data, as it does possess the necessary key to do so, unless it is designated as the data receiver for the originating device. In the case of a device running its own instance of Geth, data is pushed to the blockchain in encrypted format and the identity of the sender is verified in the smart contract. Data remains in this encrypted format while on the blockchain. A data receiver may choose to subscribe to changes to new data on the blockchain for which they are capable of decrypting. These notifications are implemented through Ethereum's event logs and Go Ethereum's *eth_subscribe* JSON API calls.

The public cryptographic keys of all devices are stored on the blockchain and can be trusted as authentic. When a signed LoRa

[6]https://libsodium.org

[7]Limited to removal of function pointers which permitted different functions to be used. No alterations were made to functions which impact the integrity/security of the cipher.

[8]https://doc.libsodium.org/public-key_cryptography/public-key_signatures

message must be verified, the gateway retrieves the devices public signature from the blockchain to perform the verification.

When a data receiver wishes to read encrypted data on the blockchain, it obtains the public key of the device it is reading data from, as well as the ciphertext, and nonce. The data receiver then calculates the shared key using libSodium's key exchange function *crypto_kx_server_session_keys()* to perform ECDH key exchange. This shared key is then used to decrypt the data.

5 RESULTS

In our experiments, devices pushed data to the blockchain every 6 seconds. This interval was selected so latency tests would not be synchronized with the block time (5 seconds) and consequently skew measurements. Data generated by each type of device consists of the following:

- Adafruit Feather M0 (LoRa): power source voltage and uptime,
- RPi 2B+ (LoRa): `uptime`,
- RPi Zero W: `uname -a`, `nproc`, `uptime`, and `free -h`, and
- RPi 4B: `uname -a`, `nproc`, `uptime`, and `free -h`.

LoRa device transmissions have fewer bytes due to the limitations of packet payload size, and for the Feather M0's, the lack of a general-purpose operating system. Since the Raspberry Pi Zero Ws are the most resource constrained devices that run their own Go Ethereum client, data was collected with Geth running in full sync and also in light sync mode. Fast sync was not used as it only affects the speed of joining a blockchain and not normal operation.

While attempting to run Geth in light sync mode on the Raspberry Pi Zero Ws, issues with memory usage arose. An initial *--cache* value of 400 (MB) was used. This resulted in the system over-using the swap space. The cache value was decreased to 128, but did not alleviate the issue entirely. To further adjust for these memory demands, the memory reserved for the GPU was decreased to 8 MB, all non-essential system services were disabled, and the system "swappiness" value was set to 1. These changes resulted in stable operation on the Raspberry Pi Zero Ws.

The LoRa gateway/proxy (Raspberry Pi 2B+) did not experience any memory-related issues, as with the Raspberry Pi Zero Ws. With 1GB of memory, the gateway is the second most memory constrained device. This device was run with a *--cache* value of 512 (MB) and had a total of 969 MB of usable memory (the difference is reserved for the GPU).

Latency was measured on The Things Network over LoRaWAN using a Raspberry Pi 2B+ as a single channel gateway and the other Raspberry Pi 2B+ as a LoRaWAN end node. The end node also ran an MQTT client which subscribed to new data on this The Things Network application. The node logged the UNIX epoch time in milliseconds upon transmission and data notification. A summary of the measured latencies is found in Table 2. The network RTT from the LAN to The Things Network router was measured using the *tcptraceroute* utility, as the router does not reply to pings. An average network latency of 63.9 ms was measured to *us-west.thethings.network:1883*.

Figures 3 and 4 show latency under the experimental scenarios. These measurements were made by transmitting text data. With both The Things Network and the blockchain-based system, messages are subscribed to, and the time between transmission and notification are recorded. The measurements on The Things Network showed a mean latency of 353 milliseconds. Measurements on the Raspberry Pi 4B, and Raspberry Pi Zero Ws with 1 light serve node and 2 light serve nodes showed a mean latency of 3949 ms, 19488 ms, and 18934 ms respectively.

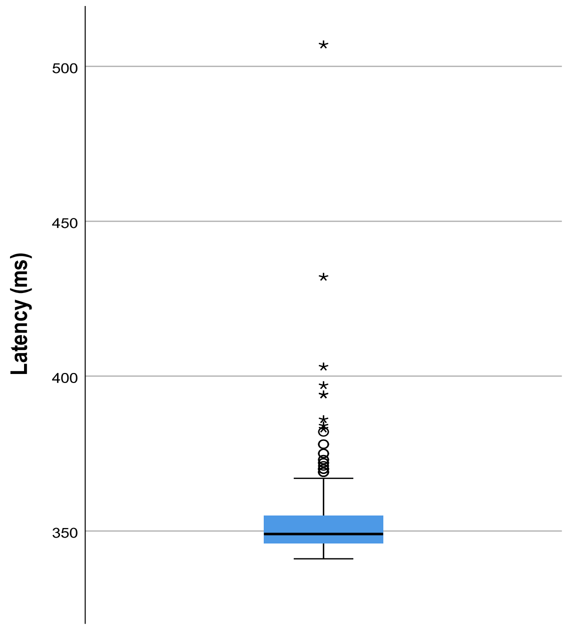

Figure 3: Measured Data Latency: The Things Network

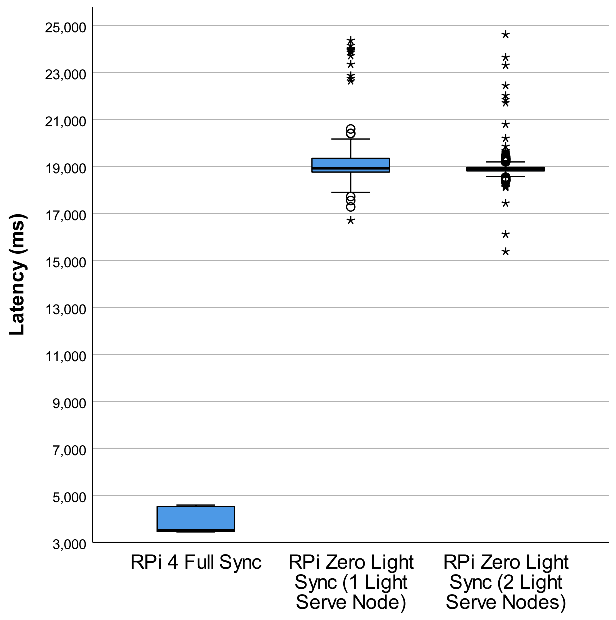

Figure 4: Measured Data Latency: RPi 4B Full Sync and RPi Zero W Light Sync

Latency measurements made on the Raspberry Pi 4B in full sync mode showed a larger standard deviation (529.2) than the measurements from The Things Network (13.8), however, it showed no outlier values. This is likely due to the lack of a dependence on an external system to process the request and transport over the public internet. The Raspberry Pi Zero Ws also resulted in many larger outliers in both tests. The large variance in latency observed on the Raspberry Pi Zero Ws may be an issue for some IoT applications. Cloud-based services such as The Things Network offer considerably lower latencies than the blockchain-based system in all configurations examined.

From the latency measurements, it is clear that a blockchain-based system introduces a considerable latency. This latency is exacerbated when using Go Ethereum's light sync mode to reduce processor and memory requirements. Since IoT tends to run on abundant, inexpensive hardware, it is clear that more IoT devices would be likely to run an Ethereum client in a light sync mode over full sync mode. This would restrict these devices to applications where latencies of approximately 19.5 seconds is permissible. This cannot rival cloud-based services, such as The Things Network, for fast delivery of data. Even on devices with sufficient resources to use full sync mode, the latencies will clearly exceed those of cloud-based services.

The second latency test conducted on the Raspberry Pi Zero Ws had an additional Raspberry Pi 4B on the test network (not shown on Figure 2). Both Raspberry Pi 4Bs were run in full sync mode and served the Raspberry Pi Zero Ws in light sync mode. The measurements of this experiment are shown in the right-most boxplot of Figure 4. The additional light serve node did not reduce the average latency, but did reduce the variance in latency. Both of these scenarios did, however, have a substantial number of outliers.

To assess the demand on the compute resources of the devices, the load averages were sampled over time. Data from the first 16 minutes of each experiment was discarded to eliminate any startup effects. The Raspberry Pi Zero W load averages were sampled at intervals reflecting the observed latency whilst the other devices were sampled every 6 seconds.

While running the Raspberry Pi 4B in full sync mode (Table 3) and serving light clients, the load averages were well below the systems total load capacity of 4.0. The demand on this client will increase as additional light clients would be added to the blockchain. It has the capacity to service more light sync clients, but how many cannot be concluded without larger scale evaluation.

The Raspberry Pi 2B+ operating as a LoRa gateway/proxy was run as a a full sync node with signing autority for the Proof-of-Authority consesus scheme. The load averages measured on this device during operation are found in Table 4. This device was configured to not serve light sync clients at any point. Despite the fact that this device has less compute power than the Raspberry Pi 4B, it experienced less load on its processor due to the lack of serving light clients.

In full sync mode, the Raspberry Pi Zero Ws (Table 5) were observed to have load averages well above their capacity of 1.0. On a single core device this indicates the system is overloaded. The Raspberry Pi Zero W is therefore not capable of running Go Ethereum as a full sync node. When tested as a light sync node (Table 6), on average it did not overload the processors, although the maximum load averages do indicate periods in which they were overloaded.

Go Ethereum exhibited periodic bursts of heavier processor utilization (Figure 5) on the Raspberry Pi 4B in full sync mode (Table 3). This behaviour was not present on the Raspberry Pi Zero W in full sync mode, which may be attributed to the system being overloaded.

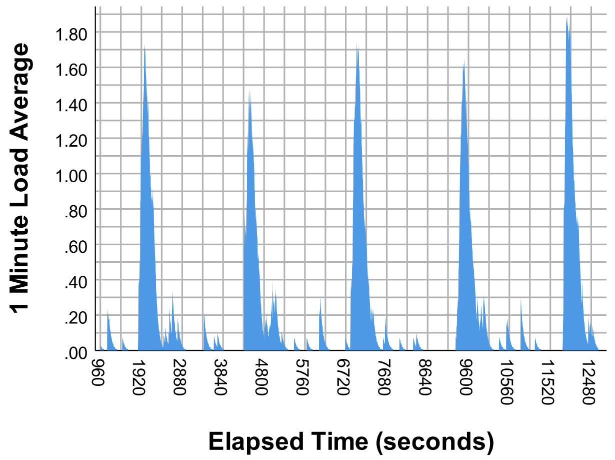

Figure 5: Load Average: RPi 4B Full Sync Node, Light Serve

6 DISCUSSION

For IoT devices which provide real-time data, such as a security camera, this system may not offer data storage, but other services can be rendered by the blockchain. These services include a cryptographic key management system, a registry of the devices IP address, and a remote device administration platform.

The use of a private network allowed for greater control of blockchain parameters. These included a degree of control over latency, avoiding need for transaction fees, and the use of Proof-of-Authority consensus over Proof-of-Work. This also allowed for the inclusion of some IoT devices in the security of the blockchain itself through the voting process used in Proof-of-Authority. This could be further leveraged in scaled up networks through the use of multiple segregated blockchains to limit blockchain growth rate, reduce the network throughput on each device, and increase security through isolation.

Although the Raspberry Pi Zero Ws did manage to run the blockchain IoT security framework as a light client, it used the vast majority of their resources and may border on being impractical. Due to the broad range of hardware used in IoT systems, an all-encompassing system will require more modes of operation to best tailor the system to the needs of each device. Future designs for a blockchain-based IoT security framework could account for such devices (IPv4/IPv6 connected, but limited compute or memory resources), by extending the concept of the proxy used for LoRa devices to devices over IP networks. This would not only serve to shift a considerable amount of the demands on the processor and memory to a more capable device, but also reduce the latency closer to those measured on the Raspberry Pi 4B.

Table 2: Measured Data Latencies (ms)

System	N	Min	Max	Mean	Std. Deviation
The Things Network	310	341	507	353	14
RPi 4B Full Sync	249	3,441	4,586	3,949	529
RPi Zero W Light Sync	104	16,706	24,364	19,488	1,689

Table 3: RPi 4B Load Avg as Full Sync, Light Serve

	1 minutes	5 minutes	15 minutes
Mean	0.22	0.21	0.17
Minimum	0.00	0.00	0.00
Maximum	1.89	0.91	0.45

N = 1950 (after discarding)

Table 4: RPi 2B+ Load Avg as Full Sync, LoRa Gateway

	1 minutes	5 minutes	15 minutes
Mean	0.10	0.10	0.13
Minimum	0.00	0.06	0.08
Maximum	0.34	0.18	0.18

N = 193 (after discarding)

Table 5: RPi Zero W x2 Load Avg as Full Sync

	1 minutes	5 minutes	15 minutes
Mean	1.44	1.50	1.48
Minimum	0.43	1.00	1.04
Maximum	3.16	2.22	1.87

N = 2881 (after discarding)

Table 6: RPi Zero W x2 Load Avg as Light Sync

	1 minutes	5 minutes	15 minutes
Mean	0.47	0.48	0.46
Minimum	0.00	0.21	0.21
Maximum	1.41	0.92	0.72

N = 1462 (after discarding)

Data in this system remains encrypted end-to-end. While the data on the blockchain itself must be considered publicly visible, data exists on the blockchain in encrypted form. The public keys of devices also exist on the blockchain, and it is possible to publicly determine the public key of the recipient device. Since the blockchain is also an immutable ledger, this history will persist on the blockchain. Although the cryptographic systems used are not known to be insecure, it should be noted that if any of these ciphers are broken, the history on the ledger will be exposed.

The integrity of data is maintained in two ways. Data that is already on the blockchain remains immutable by virtue of the design of the blockchain system itself. Secondly, data being sent to the blockchain is validated for integrity either by the LoRa gateway (which is trusted by the smart contract), or if the device is running its own Go Ethereum client, the blockchain network will validate the signature of the transaction sent to the contract. While the gateways are not insecure, gaining control of a single gateway would permit an attacker to exploit the smart contracts implicit trust of the gateway. This would allow the attacker to submit data to the blockchain on behalf of any LoRa device, but not devices which do not use this proxy system. This level of exposure is due to the design choice to allow any LoRa device to operate with any LoRa gateway on the system to allow for geographic mobility of devices. The alternative of this being each LoRa device may only communicate with a specific gateway rendering nodes less mobile.

The availability of data that already exists on the blockchain is extremely considerable, due to the distributed nature of blockchain technologies. This makes the existing data almost impervious to Denial of Service attacks. Individual nodes may be targeted and temporarily disabled, but the greater system itself would continue to function and previously received data from the victim device would continue to be available.

The proposed system is vulnerable to jamming attacks by virtue of the LoRa LPWAN technology itself. As with any wireless communications technology, a transmission can be disabled or interrupted by overwheming the channel(s) with noise. LoRa is particulairly succeptible to this due to its low transmission power and use of license-free frequencies. While nothing can be done to eliminate this vulnerability, monitoring of the most recent data timestamps could provide an indication of potential communications issues.

Since the only devices that presently do not run their own Ethereum client are devices communicating exclusively over LoRa, these devices are the only class of device which cannot be implicitly trusted. To address this, the use of digital signatures was used to authenticate the sender. This addresses the potential issue of data forgery, but the issue of replay attacks remains. While an attacker cannot view the message due to it being encrypted, nor can they alter the message due to the signatures, replaying the exact same message will appear to LoRa gateways to be authentic. This can be addressed by introducing a mechanism at the gateway which examines a message identifier which must be incremented by the sender.

While the system supports the changing of both encryption and signature keys, issues exist regarding the communication of new keys between LoRa-only devices and gateways. Since LoRa is an unreliable communication network, this creates potential inefficiencies/overhead when synchronizing keys between LoRa-only devices and gateways. Should either class of device change one of its keys, it would then need to inform the devices it communicates with over LoRa of its new public key. Should this message not be properly received, this would lead to the public keys being out of sync and breaking communications between the pair of devices. A reliable protocol for the exchange of keys is therefore required for this system to be practical in real-world applications, as changing of cryptographic keys is paramount to the ongoing confidentiality and integrity of data.

7 CONCLUSIONS AND FUTURE WORK

The Internet of Things is a rapidly growing industry that can solve many novel problems and improve the efficiency of others, but it also exposes much risk if it is not properly secured. The dangers of vulnerable IoT devices is not merely hypothetical; security flaws have already been found which endangered lives [28]. Blockchain technology can provide the backbone required to create a strong, unified security framework for a network of heterogeneous IoT systems. Utilizing a blockchain-based solution introduced longer latencies, but did successfully consolidate a broad range of hardware into one security framework. Through additional modes of operation, such as a proxy over IP, the maximum latencies of the system could be reduced. In addition, our system delivers a superior resistance to the growing threat of Denial of Service attacks, by virtue of the distributed nature of blockchain systems. The system presented caters well to wireless sensor networks and other delay tolerant applications, and with further development can be significantly improved.

Our system as described and implemented is useful for a subset of IoT applications and could offer other functionality in a Denial of Service resistant manner. As part of future work, the system will be extended to more classes of devices to provide a comprehensive framework. Ways of improving the usage of system resources will be further explored and compared to lower the threshold of capabilities that are required of devices in order to participate in the Proof-of-Authority voting process. The use of distributed storage systems, such as SWARM and IPFS will be explored for the use of data storage, opposed to the smart contract state. Metrics will be gathered with these technologies and compared to determine the most feasible solution for resource constrained systems.

Many additional mechanisms could be added to further harden the security of this system and expand its utility. Security can be improved by addressing the issues of LoRa replay attacks and changing cryptographic keys described in Section 6. Additional features could include a registry of IP addresses, and a secure remote device administration platform. It will also be necessary to evaluate these systems at scale and examine the tails of response time distributions in more detail, because of the need for IoT security to be deployed in real-time.

REFERENCES

[1] D. R. Aleko and S. Djahel. 2019. An IoT Enabled Traffic Light Controllers Synchronization Method for Road Traffic Congestion Mitigation. In *2019 International Smart Cities Conference (ISC2)*. IEEE, Casablanca, Morocco, 709–715.

[2] Pelin Angin, Melih Burak Mert, Okan Mete, Azer Ramazanli, Kaan Sarica, and Bora Gungoren. 2018. A blockchain-based decentralized security architecture for IoT. In *International Conference on Internet of Things*. Springer, Seattle, WA, 3–18.

[3] A.M. Antonopoulos. 2014. *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*. O'Reilly Media, Sebastapol, CA.

[4] A. Augustin, J. Yi, T. Clausen, and W.Wm. Townsley. 2016. A Study of LoRa: Long Range & Low Power Networks for the Internet of Things. *Sensors* 16, 9 (2016), 1–18. Article 1466.

[5] J. P. Aumasson, S. Fischer, S. Khazaei, W. Meier, and C. Rechberger. 2008. New features of Latin dances: Analysis of Salsa, ChaCha, and Rumba. In *Fast Software Encryption*, Vol. 5086 LNCS. Springer, Lausanne, Switzerland, 470–488.

[6] Daniel J. Bernstein. 2004. Cache-timing attacks on AES.

[7] Daniel J. Bernstein. 2008. ChaCha, a variant of Salsa20. https://cr.yp.to/chacha/chacha-20080128.pdf

[8] Daniel J. Bernstein. 2008. Extending the Salsa20 nonce. https://cr.yp.to/snuffle/xsalsa-20110204.pdf

[9] Daniel J. Bernstein. 2008. *The Salsa20 Family of Stream Ciphers*. Springer Berlin Heidelberg, Berlin, Heidelberg, 84–97.

[10] K. Biswas and V. Muthukkumarasamy. 2016. Securing Smart Cities Using Blockchain Technology. In *2016 18th International Conference on High Performance Computing and Communications; 14th International Conference on Smart City; 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, Sydney, Australia, 1392–1393.

[11] Vitalik Buterin. 2014. A next-generation smart contract and decentralized application platform. (2014), 36 pages. White Paper.

[12] Cisco Systems Inc. 2020. Cisco Annual Internet Report (2018-2023) White Paper. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[13] Stefano De Angelis, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. 2018. PBFT vs Proof-of-Authority: Applying the CAP Theorem to Permissioned Blockchain. In *Italian Conference on Cyber Security*. CINI, Milan, Italy, 1–11.

[14] A. Dorri, S. S. Kanhere, and R. Jurdak. 2017. Towards an Optimized BlockChain for IoT. In *2nd International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE/ACM, Pittsburgh, PA, 173–178.

[15] Cynthia Dwork and Moni Naor. 1993. Pricing via Processing or Combatting Junk Mail. In *Advances in Cryptology — CRYPTO' 92*. Springer Berlin Heidelberg, Berlin, Heidelberg, 139–147.

[16] Seth Gilbert and Nancy Lynch. 2002. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. *SIGACT News* 33, 2 (June 2002), 51–59.

[17] J. Hoffstein, J. Pipher, and J.H. Silverman. 2014. *An Introduction to Mathematical Cryptography*. Springer New York, New York, NY.

[18] Sunghyuck Hong. 2017. Secure and light IoT protocol (SLIP) for anti-hacking. *Journal of Computer Virology and Hacking Techniques* 13, 4 (01 Nov. 2017), 241–247.

[19] Seyoung Huh, Sangrae Cho, and Soohyung Kim. 2017. Managing IoT devices using blockchain platform. In *2017 19th International Conference on Advanced Communication Technology (ICACT)*. IEEE, Phoenix Park, PyeongChang, South Korea, 464–467.

[20] K. Lauter. 2004. The advantages of elliptic curve cryptography for wireless security. *IEEE Wireless Communications* 11, 1 (2004), 62–67.

[21] Jun Lin, Zhiqi Shen, and Chunyan Miao. 2017. Using Blockchain Technology to Build Trust in Sharing LoRaWAN IoT. In *Proceedings of the 2nd International Conference on Crowd Science and Engineering*. Association for Computing Machinery, Beijing, China, 38–43.

[22] Kerry Maletsky. 2015. RSA vs ECC comparison for embedded systems. (2015), 4 pages.

[23] Daniel Minoli and Benedict Occhiogrosso. 2018. Blockchain mechanisms for IoT security. *Internet of Things* 1-2 (2018), 1–13.

[24] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System.

[25] National Institute of Standards and Technology. 2001. *FIPS PUB 197: Announcing the ADVANCED ENCRYPTION STANDARD (AES)*. National Institute of Standards and Technology, Gaithersburg, MD.

[26] K. R. Özyilmaz and A. Yurdakul. 2019. Designing a Blockchain-Based IoT With Ethereum, Swarm, and LoRa: The Software Solution to Create High Availability With Minimal Security Risks. *IEEE Consumer Electronics Magazine* 8, 2 (March 2019), 28–34.

[27] Phillip Sparks. 2017. White Paper: The route to a trillion devices. https://community.arm.com/iot/b/internet-of-things/posts/white-paper-the-route-to-a-trillion-devices

[28] US Food and Drug Administration. 2017. Cybersecurity vulnerabilities identified in St. Jude Medical's implantable cardiac devices and Merlin@ home transmitter: FDA safety communication. https://www.fda.gov/MedicalDevices/Safety/AlertsandNotices/ucm535843.htm

[29] Vivek Vishnumurthy, Sangeeth Chandrakumar, and Emin Gun Sirer. 2003. Karma: A secure economic framework for peer-to-peer resource sharing. In *Workshop on Economics of Peer-to-peer Systems*. Berkeley School of Information, Berkeley, CA, 1–6.

[30] S. Wang, L. Ouyang, Y. Yuan, X. Ni, X. Han, and F. Wang. 2019. Blockchain-Enabled Smart Contracts: Architecture, Applications, and Future Trends. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 49, 11 (2019), 2266–2277.

[31] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. (Oct. 2014), 32 pages. https://ethereum.github.io/yellowpaper/paper.pdf.