Blockchain based security for heterogeneous IoT systems

Kale A. Yuzik
University of Saskatchewan
Saskatoon, Saskatchewan
kay851@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 expose information or may allow for malicious actors to take control of the devices. 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, allowed 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 measured, showing that devices with less resources showed significant latencies, and suggestions as to how these latencies can be reduced are proposed.

KEYWORDS

IoT, Internet of Things, Blockchain, Ethereum, Security, Networking, Key management, Distributed computing

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 The United States will see a $5 trillion boost in G.D.P due to the use of IoT technology in industry in 2035 [23]. The Internet of Things offers great value for uses such as monitoring critical infrastructure, and this value will surely lead to the deployment of these systems throughout cities. Manufacturers are driven by economic factors and those fast to market benefit the most. This encourages manufacturers to cut corners and take calculated risks which is no exception when it comes to the security of these products. With many of these IoT devices being deployed in remote or inaccessible locations and using low bandwidth connections, servicing or updating them can be far more challenging than conventional computer networks. With expanding and more frequent use of IoT systems, the risks involved with failure or security breaches become increasingly severe. With the integration of IoT in infrastructure, utilities, road networks, and healthcare, the cost of breaches will not only be financially expensive, but may also endanger lives. For these reasons it is critical these systems be secure at the time they are deployed.

Traditional approaches to security present unique problems when applied to the diverse range of hardware utilized in the Internet of Things. 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. With more solutions come greater potential for configuration errors and complexity of administration.

This paper explores the application of blockchain technology to create a unified security framework for IoT devices of varying resource limitations.

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 [13].

Confidentiality refers to the secrecy of data from those who are not authorized to view or access it. Data can be kept confidential using cryptography to encrypt it.

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 describes the accessibility of data and an assurance that adversaries cannot prevent or hinder access to it. 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 [14], owing to the lack of a single point of failure [1] and distributed ledger containing the desired data. With these types of attacks growing in both frequency and size, Cisco has projected 15.4 million DDoS attacks in 2023, nearly double the 7.9 million which were expected in 2018 [15]. With these types of attacks becoming more common, utilizing a decentralized solution is advantageous.

2.1 Blockchain

The roots of blockchain began in 1992 when Dwork and Naor [11] published a paper titled "Combatting Junk Mail". Their work introduced the idea of proof-of-work, a way of providing means for making an assertion without the need of cryptographic trust.

In 2003, a paper was published by Vishnumurthy et al. [24] which made use of the concept of proof-of-work that Dwork et al. described. This paper outlined a proof-of-work based credit system to incentivize equal contribution of all nodes within peer-to-peer systems. This system provided a public ledger of transactions, like those found in modern cryptocurrencies and involved the payment of "karma", a digital token for work performed by peers.

A paper was published under the alias Satoshi Nakamoto in 2008 [20], which described a decentralized, anonymous digital currency, which was referred to as Bitcoin. This expanded the scope of the work done by Vishnumurthy et al. into a token for use as a general currency. Since Bitcoin's inception, many new and different implementations of blockchain technology have emerged.

At its core, blockchain is a distributed database [17]. As the name suggests, it consists of chunks of data (blocks) that are linked together in a linear order. Each block contains the cryptographic hash of the block prior [20]. A change in any block along the chain will result in one of these hashes not matching. This makes it possible to verify the integrity of the blockchain by recursively checking that hashed value of a block's ancestor matches the hash stored in the successor of that block. 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 [20].

In the proof-of-work consensus scheme, mining nodes on the network assemble a block with pending transactions. Each block contains a numeric value known as the nonce (number only used once). A miner assigns an arbitrary a value to the nonce field and calculates the hash of this proposed new block. The miners then check if the hash is less than the difficulty value [2]. Should the new block and nonce combination result in a hash less than the difficulty, then the network will consider this a valid block. The difficulty is adjusted to maintain a predetermined duration of time between creation of new blocks. Miners race with others to find values which satisfy these criteria. When a miner succeeds, it broadcasts its newly mined block to connected peers, which then verify it meets the requirements. If it is indeed valid, they accept it, begin work on the next block and broadcast the new block to their neighbours.

In this system it is possible that a new block may be mined at, or near the same time. As these new blocks propagate through the network, other nodes will accept the first of the two they see. Because of this there may temporarily be multiple forked versions of the blockchain on a network at any given time. A node will always accept the longest valid chain; therefore, this tie will eventually be broken when the next valid block is mined and propagates through the network to each node [20].

Finding a hash which meets the required difficulty parameter involves constant computation [2], which consumes processor cycles, power, and because mining is a race for the next block, it is only viable on hardware of considerable computational speed. For these reasons, proof-of-work is an impractical solution for a consensus algorithm in securing a blockchain network of IoT devices, as this restricts the inclusion of IoT devices in the consensus process.

As an alternative to proof-of-work consensus, a voting-based system known as proof-of-authority (PoA) may be used. In the proof-of-authority consensus scheme, blocks are approved (or rejected) by approved accounts known as signers.[1] 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 signers.[2] Upon genesis of the blockchain, initial signers are defined. Accounts which maintain the transaction process of the blockchain accumulate positive reputation. 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, proof-of-authority consensus allows for the block time to be explicitly set, thus allowing for some degree of control over latency when calling contract functions which mutate the contract state.[3]

2.2 Ethereum

Blockchain is best known for its use in implementing a cryptocurrency, but its applications are far more extensive. Many implementations of blockchain introduce the concept of a smart contract; compiled code that is uploaded to the blockchain [25]. These contracts contain functions, no different than conventional computer programs, which 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. Bitcoin and Ethereum differ in their support for contracts. Ethereum provides a Turing complete virtual machine to execute contracts, while Bitcoin's contracts are not Turing complete [8] [14]. For this reason, Ethereum can be employed as a distributed computer. Languages used to implement smart contracts for Ethereum include Solidity, Serpent, and LLL[4].

Smart contracts, as they are often referred to in the context of Ethereum, can be called in two ways. Contract functions which do not modify the state variables of a contract need only be executed locally on the device calling upon this function.[5] There is no need to come to a consensus on the result of this computation, as it does not modify the public information in any way. This function call completes synchronously, in an imperceptible length of time.

With Ethereum's inclusion of a Turing complete virtual machine, the issue of the halting problem arises.[6] The Ethereum Virtual Machine (EVM) allows for looping, and thus introduced the possibility of poorly written or malicious code causing miners to enter an infinite loop. To fix this issue, 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 amount of gas

[1] https://github.com/poanetwork/wiki/wiki/POA-Network-Whitepaper
[2] Szilágyi, Péter. March 6, 2017. https://github.com/ethereum/EIPs/issues/225
[3] https://geth.ethereum.org/docs/interface/private-network
[4] Ethereum Frontier Guide. https://ethereum.gitbooks.io/frontier-guide/writing_contract.html
[5] https://geth.ethereum.org/docs/rpc/ns-eth#eth_call
[6] https://github.com/ethereum/wiki/wiki/White-Paper

a transaction may begin with; however, the spent gas determines transaction fees charged to the account from which the transaction originated. There also exists a network wide upper limit on gas, called the gas limit or block gas limit.

When a contract function modifies the contract's state, it must be called by sending a transaction.[7] 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. These transactions take a greater amount of time to complete, as mining nodes on the Ethereum network must execute the contract function and come to a consensus on the results. The latency of this is primarily dependent on the time that it takes the blockchain network to generate new blocks, referred to as the "block time".

While proof-of-work consensus algorithm, Ethash is the method of consensus on the public Ethereum network, it presents many problems which other consensus algorithms address. [3] The proof-of-authority (PoA) algorithm does not require repeated hashing as PoW does.

Ethereum accounts each possess their own pair of cryptographic keys which are used to digitally sign transactions. These same keys are used to sign blocks when using proof-of-authority consensus. This uses Elliptic Curve Digital Signature Algorithm (ECDSA) on the SECP-256k1 curve [25] [14].

Creating a private blockchain allows for certain parameters to be selected during genesis of the blockchain, which cannot be altered if using the public Ethereum blockchain. This includes, but is not limited to: the consensus algorithm (PoW vs. PoA), block time, transaction fees, and block gas limit. [3] It is imperative to note that the term "private chain" does not mean secure, but "reserved or isolated".[8] Ethereum clients actively seek peers to connect with and to establish a connection, however peers must have the same network ID and genesis block. [3] For these reasons, data on a blockchain must be considered publicly visible, even on a private chain.

2.3 Go Ethereum Client

The Go Ethereum client (Geth) is one of a few implementations of an Ethereum client. 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 on the blockchain: 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.[9] 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.[10] 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, as the name suggests, is the mode which 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.[11] 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. This is disabled by default and must be explicitly turned on.

Memory usage can be restricted using Geth's *--cache* flag, which limits cache size and transitively allows some for control over memory usage.[12]

Geth provides an API via JavaScript Object Notation (JSON). This can be called through either HTTP or Unix domain sockets.[13]

Data can be signed with the cryptographic keys associated with the Ethereum account used locally through Go Ethereum's API. Data signed with this call is first salted with *"\x19Ethereum Signed Message:\n" + len(message)*[14] to prevent malicious programs from being able to collect an arbitrary account's signature needed to authorize a transaction from the account.

To improve the security of the private chain, Geth's *--netrestrict* flag can be used. This allows for a list of CIDR subnet masks to be provided to limiting the addresses which Geth may communicate with. While this is effective, neglecting to use this on a single client will circumvent the security this provided on other clients on this blockchain.

Considering the above settings and features, running Go Ethereum on moderately resource constrained hardware may be viable.

2.4 LoRa, LoRaWAN & Cloud-Based Services

LoRa (Long Range) is a low-power, wide-area network (LPWAN) physical layer wireless network technology. It makes use of license-free radio-frequency bands below 1 GHz. LoRa supports packets as large as 255 bytes and can achieve ranges as far as 2 to 20 kilometers [17], depending on local obstructions and radio configuration.

LoRaWAN[15] is a medium access control (MAC) layer built on top of LoRa. LoRaWAN is employed with cloud-based services such as The Things Network[16], to capture, process, and forward data from devices. When data is received at a LoRaWAN gateway, it is encapsulated in either an MQTT or gRPC protocol before being sent to The Things Network router for processing.[17] Devices wishing to subscribe to this data stream can do so using the MQTT protocol to receive notifications. A maximum payload of 51 bytes can be sent per message.[18]

2.5 Cryptography

2.5.1 Symmetric Ciphers. The category of symmetric cryptography encompasses systems which use the same key to both encrypt

[7] https://github.com/ethereum/wiki/wiki/JSON-RPC#eth_sendtransaction
[8] https://geth.ethereum.org/docs/interface/private-network
[9] https://geth.ethereum.org/docs/getting-started
[10] Szilágyi, Péter. October, 2015. eth/63 fast synchronization algorithm #1889. https://github.com/ethereum/go-ethereum/pull/1889
[11] https://geth.ethereum.org/docs/getting-started
[12] https://geth.ethereum.org/docs/interface/command-line-options
[13] https://geth.ethereum.org/docs/rpc/server
[14] bas-vk. August, 2016. internal/ethapi: add personal_sign method #2940. https://github.com/ethereum/go-ethereum/pull/2940
[15] References to "LoRa" in this paper refer to the physical layer technology, and not LoRaWAN, the MAC layer.
[16] https://www.thethingsnetwork.org/
[17] https://www.thethingsnetwork.org/docs/gateways/packet-forwarder/
[18] https://www.thethingsnetwork.org/docs/lorawan/limitations.html

and decrypt information for both parties [12]. This requires that both participants know the shared key before the exchange of encrypted data. In most cases this requirement of prior knowledge is problematic. If the key must be known before an encrypted message can be sent, then a shared key must be agreed upon before a symmetrically encrypted information exchange may begin. Either the key must be exchanged in an unsecure manner or a different solution must be used. Despite these downsides, symmetric cryptography is simpler, and thus faster than asymmetric cryptography.

2.5.2 Asymmetric Ciphers. Asymmetric cryptography involves both actors having different cryptographic knowledge and abilities [12]. Each participant has their own pair of cryptographic keys; one private knowledge and one public knowledge. This category of ciphers offers many unique advantages over symmetric ciphers; however, it is not superior in all measures. Asymmetric ciphers are more computationally expensive and a solely asymmetric approach to encryption is not feasible in many domains.

Unlike symmetric ciphers, asymmetric ciphers do not require prior knowledge of any privileged information, other than one's own private key [12]. This provides a solution to the issue that arises with symmetric key exchange, which is commonplace. Algorithms such as Diffie-Hellman allow for a shared key to be derived only from knowledge of one's own key-pairs and the partners public key. When the partner performs the same calculation, the result is the same; the shared key. It is important to note that any information communicated between these parties does not provide a third party whom is observing the exchange any means to determine the shared key.

Another unique advantage with asymmetric cryptography is the possibility for data to be digitally signed and verified [12]. This allows for the source of data to be confirmed with a very high degree of confidence that the data originated from an actor who possessed a specific key-pair.

2.5.3 RSA vs. ECC. 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 speed, and security [18]. 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. This means that ECC can offer an equal level of security with a much shorter set of keys.

It was shown by Lauter [16] that ECC is faster than RSA for key sizes of comparable security by a large margin that widens with key length.

Table 1: ECC vs RSA - Security and Speed Comparison

ECC Key bits	RSA Key bits	ECC faster by
163	1024	5x to 15x
256	3072	20x to 60x
521	15,360	~400x

Data credit to Lauter [16].

2.5.4 Symmetric Cipher – Advanced Encryption Standard (AES). The Advanced Encryption Standard was first proposed under the name Rijndael in 1998 [9]. AES has been heavily used since its acceptance by NIST in 2001 [21]. It has been widely implemented in hardware for increased speed and efficiency.

In 2004, Daniel J. Bernstein published a paper on a cache timing-based attack on AES, which allows for possible key recovery [4]. Fault of this weakness was placed on the design of the algorithm and not weaknesses of any specific implementation.

2.5.5 Symmetric Cipher – Salsa20. Designed in 2005 by Bernstein, the Salsa20 stream cipher offers encryption that is continually faster than AES [7], the de-facto standard. Salsa20 may be applied using a differing number of rounds, with Salsa20/20 (20 rounds) being the recommended standard. Cryptanalysis on Salsa20 has shown Salsa20/8 or less rounds to be vulnerable to attacks [3] [7].

The Salsa20 family of ciphers use 3 operations [7]:

- 32 bit Addition
- 32 bit Exclusive Or
- Constant-Distance 32 bit Rotation

These instructions are all CPU friendly, and thus is faster across a wider number of platforms than other ciphers such as AES [7]. Due to the lack of S-Box lookup tables, Salsa20 also avoids the cache timing attacks that AES suffers from.

2.5.6 Symmetric Cipher – XSalsa20. Based upon Salsa20, XSalsa20 specifies a longer nonce [6]. The purpose of a nonce is to introduce variation in ciphertext when the same patterns may appear in plaintext. Allowing these patterns to appear in ciphertexts facilitate the breaking of the cipher. The nonce does not need to be secret; a third party obtaining the nonce does not compromise security of the cipher. Care should however be taken to ensure the same nonce is not used with the same key more than once. XSalsa20 uses a 192 bit nonce, as opposed to Salsa20 which uses a 64 bit nonce. With Salsa20, it is not considered safe to use a randomly generated nonce, but with XSalsa20's longer nonce it is.[19]

XSalsa20 offers the exact same speed as Salsa20, with the minimal extra cost of generating the larger nonce. Bernstein has proven that XSalsa20 is secure if Salsa20 is secure [6].

2.5.7 Symmetric Cipher – ChaCha. The family of ChaCha ciphers are based on Salsa20 [5]. The changes made in ChaCha were aimed to improve Salsa20's diffusion. A cipher with a higher degree of diffusion does a better job in hiding the relationship between plaintext and ciphertext. This modification does not increase the computational expense, nor does it alter the parallelizability or vectorizability over Salsa20. In fact, ChaCha20 uses one less register than Salsa20, which on some platforms may yield minor performance gains.

Aumasson et al. performed a differential cryptanalysis on Salsa20 and ChaCha [3]. 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). This suggests that ChaCha may indeed offer stronger security than Salsa20.

2.5.8 Symmetric Cipher – XChaCha. ChaCha and XChaCha bear the same relationship as with Salsa20 and XSalsa20; the extended

[19] https://libsodium.gitbook.io/doc/advanced/stream_ciphers/xsalsa20

192 bit nonce. This again allows for randomly generated nonce to be safely used. XChaCha, however, lacks a formal specification, which has limited its adoption.[20]

3 RELATED WORK

Similar research was done by Özyilmaz and Yurdakul on an Ethereum based IoT data collection system [22]. Wireless nodes used LoRaWAN to communicate with a "smart proxy" which performed blockchain related functions. This work focused on the aspects of blockchain technology for decentralized storage and robust availability of the data, but did not employ cryptography to ensure the stored data is confidential. 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. conducted an analysis of blockchain technology in the scope of providing security for IoT [19]. 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.

In research by Dorri et al. [10], the design of the blockchain itself was examined 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, it is not Turing complete [8] [14], which greatly limits its practicality for use as a framework such as this.

When considering the design of this system, the resource constraints which some IoT devices impose must be given consideration. In order to encompass this range of devices into one system, a proxy will be built into the design. This will allow devices which are incapable of running their own Ethereum client due to any of the five resource limitations to be included in the system.

In keeping with minimizing resource usage, the choice of language used must allow for efficient use of hardware, and allow for multiple threads. For these reasons, C++ was used. This also means that existing library, web3.js used for communicating with Geth cannot be used since this is done with JavaScript.[21] A custom system was written in C++ to manage communications with Geth with it's JSON API through Unix Domain Socket. This will keep resource consumption as low as possible.

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. With this low processing power, the trade-offs between latency due to computational complexity and security become far more pronounced than devices with faster processors. When navigating this issue security will be held paramount 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 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.[22] LoRa allows for throughput ranging from 0.3 kbps to 50 kbps, depending on various configurations and regional differences [17]. The solution must operate over a low bandwidth connection such as this, while maintaining a high degree of security.

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

4.1.5 Cryptographic Functions. Cryptographic functions required consist of: public/private key generation, Diffie-Helman key exchange, a symmetric key cipher, and signature creation and verification. Elliptic key cryptography was selected over RSA, due to both its smaller key size without compromised security and computational speed, which is well suited to embedded systems [18].

The implementation will be tested on a network consisting of the following devices:

2x Raspberry Pi 2B+

- Quad core 64bit ARM Cortex-A53 @ 900MHz
- 1Gb RAM

[20] https://libsodium.gitbook.io/doc/advanced/stream_ciphers/xchacha20

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

[22] https://www.semtech.com/lora/what-is-lora

- Dragino LoRa (SX1276) and GPS module v1.4[23][24]
- WiFi

1x Raspberry Pi 4B

- Quad core 64bit ARM Cortex-A72 @ 1.5GHz
- 4Gb RAM
- WiFi

2x Raspberry Pi Zero W

- Single core 32bit ARM1176JZF-S @ 1GHz
- 512Mb RAM
- WiFi

2x Adafruit Feather M0[25]

- ARM Cortex M0 ATSAMD21G18 @ 48MHz
- 32Kb RAM
- 256Kb Flash storage
- LoRa module (SX127X)
- 1x 1200mAh LiPo Battery[26]

All LoRa chips are of the RF9X family, operating on the 915MHz frequency range. All Raspberry Pis ran Raspbian on a headless installation. The above devices were run in a network as illustrated in Figure 1.

4.2 Ethereum

The blockchain security framework was tested on a private Ethereum network, using the proof-of-authority implementation in Go Ethereum, known as Clique. [3] Using PoA over PoW will allow for more devices to participate in the voting process, as compared to the mining process. This will make the security of the blockchain dependent on the quantity of signers, rather than the combined compute power of the miners. In general, this lends itself well to a network of IoT devices, since IoT devices use less powerful hardware in large quantity. 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. [3] While 1 second would result in even less 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 Contract Design

The smart contracts will be used to store information, including:

- User friendly name of devices
- Numeric ID to identify the device ("device ID")
- Timestamp of when the device was created
- Public encryption key of devices
- Public signature key of devices
- Encrypted data from the device
- Timestamp of when data was last received
- Numeric ID of the device which can decrypt this device's data
- Boolean value indicating whether this device is managed by a gateway/proxy

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 which should only be callable by authorized Ethereum accounts. The contract allows for arbitrarily many Ethereum accounts to be granted access to call such functions.

Each device is assigned a partner device, termed a "data receiver". This device functions as the destination for data from the given device. Allowing for a specific data receiver to access data from one or more devices permits for data to be kept private even with multiple users within blockchain security framework. This limits potential damage if a cryptographic key may become compromised.

4.4 Devices

4.4.1 Internet Connected Devices. 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 contract on the blockchain. Devices with more capable hardware will run Geth in full sync mode as a signer. Devices which do not meet requirements for full sync mode will be tested in light sync mode.

4.4.2 LoRa Connected Devices (Nodes). Systems using LoRa for connectivity do not possess bandwidth necessary to run Go Ethereum locally. These devices may also lack other resources to run Go Ethereum, such as compute power, memory, storage, or a general-purpose operating system. In the case of the Adafruit Feather M0, all of these requirements are not met. This is the case with a large portion of IoT devices and it is therefore important mechanisms for this category of device be included.

Since these devices cannot run Go Ethereum, this must be done by proxy. A protocol was created to allow for a LoRa gateway to act upon a device's behalf, while maintaining data confidentiality and integrity. Unlike devices running Geth locally, devices over LoRa must be authenticated and cannot be implicitly trusted. This protocol must provide means to detect forged/altered data and prevent eavesdropping.

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:

- Destination device ID (4 bytes)
- Source device ID (4 bytes)
- 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)
- Data (max 154 bytes)

[23] http://www.dragino.com/products/lora/item/106-lora-gps-hat.html
[24] https://wiki.dragino.com/index.php?title=Lora/GPS_HAT
[25] https://www.adafruit.com/product/3178
[26] https://www.adafruit.com/product/258

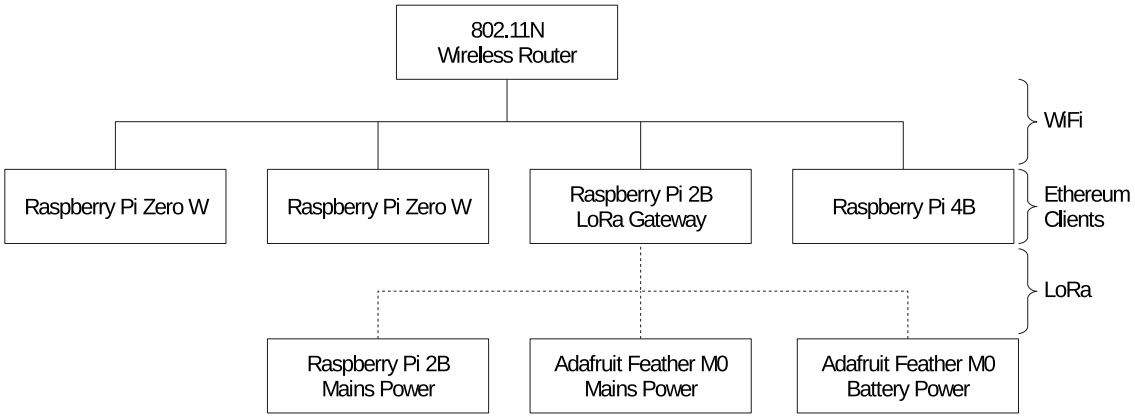

Figure 1: Configuration of test network

The "message ID" and "packet fragment number" is presently unused, but was left in for future versions. The intention of this is to allow for fragmented messages, which would allow for far larger messages to be sent. This would be similar to the packet fragmentation in IPv4.

When a LoRa device boots, it pre-calculates the shared key it must use to encrypt data for its data receiver to decrypt. This key is stored to avoid having to recompute it, wasting processor cycles and power. Once data is ready to be sent, the data is encrypted (see section 4.5) and encapsulated in a packet. The packet is then digitally signed and transmitted.

4.4.3 LoRa Gateways/Proxies. Devices operating as a LoRa Gateway constantly listen for transmissions from broadcasting nodes. This gateway must run an instance of Go Ethereum locally. 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 contract. With this public key the signature in the message is verified. If the signature is valid, the gateways can be confident the message originated from the device the message claims to be from. Since the gateway is already registered on the blockchain, the smart contract already trusts the gateway and the gateway may forward the already encrypted payload of the message to the blockchain. This allows for geographic mobility of these devices. In order for the smart contract to accept 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.

4.5 Cryptography

The use of Elliptic Curve Cryptography meets the requirements set out for cryptographic needs over RSA and therefore was used. 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 will be more efficient on embedded systems. libSodium[27] provides these algorithms and supports all but one of the platforms being used for testing. Although libSodium does not claim support for the ARM Cortex M0, after some experimentation and minor changes[28] , the library was made to work on this platform.

Despite Go Ethereum already providing an API to sign and validate digital signatures, this will not be used. This is due to LoRa devices not having access to a local Go Ethereum client to allow them to call these functions. Because of this, an external cryptography library will be used for digital signatures in addition to key generation, key exchange, and symmetric encryption.

Digital signatures utilize the Edwards-Curve Digital Signature Algorithm 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.[29]

Data within packets is encrypted using symmetric key cryptography. Before data can be encrypted, the shared key must be calculated between the device and its data receiver using Elliptic Curve Diffie-Hellman key exchange (ECDH). Once the shared key has been calculated it is used with the 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, nor does it possess the necessary key to do so. The only exception to this is if the gateway is designated as the data receiver for the device from which that packet originated from. 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

[27] https://libsodium.org

[28] These changes were limited to interfaces allowing for different functions to be swapped in. No alterations were made to functions which may impact the integrity/security of the cipher.

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

is verified in the smart contract. Data remains in this encrypted format while on the blockchain. The 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 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 encryption key of the device it is reading data from, as well as the nonce, and encrypted data. 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

Nodes pushed data to the blockchain at a set interval of time: every 6 seconds. This interval was selected so latency tests would not be synchronized with the block time and skew measurements. Data for each type of device consists of:

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

LoRa devices consist of less data due to the limitations of packet payload size and in the case of the Feather M0's, the lack of a general-purpose operating system. The data was captured with a block time of 5 seconds. Since the Raspberry Pi Zero Ws are the most resource constrained devices running their own Go Ethereum client, data was collected with their instances of Geth running in full sync and also in light sync mode. Fast sync was not used since 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 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 the the swap space not being used. Due to this observation, that in order for a device to run a local blockchain using Geth, the system memory cannot be less than approximately 504 MB.

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 and had a total of [TODO] MB of memory avaible (the difference is reserved for the GPU).

Latency was measured on The Things Network over LoRaWAN. This was done using a Raspberry Pi 2B+ as a single channel gateway and the other Raspberry Pi 2B+ as a LoRaWAN node. The node also ran a MQTT client which subscribed to new data on this The Things Network application. The node logged the epoch time in milliseconds upon transmission and data notification. A summary of the measured latencies is found in 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 63.93ms was measured to *us-west.thethings.network:1883*.

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 not an issue. 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.

Latency measurements made on the Raspberry Pi 4B in full sync mode (Figure 3) showed a larger standard deviation than the measurements from The Things Network (Figure 2), however it showed no outlier values. This was most likely due to the lack of a dependence on another system to process the request. The Raspberry Pi Zero Ws (Figure 4), like the latencies of The Things Network, also resulted in many larger outliers. The large variability and unpredictable latencies measured on the blockchain system may be an issue for some IoT systems.

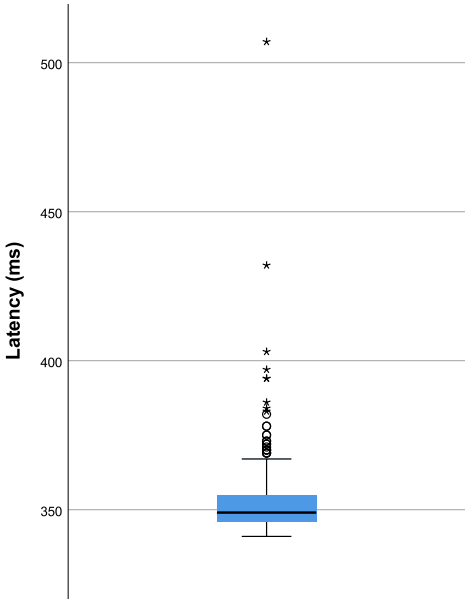

Figure 2: Measured Data Latency on The Things Network

To assess the demand on the compute resources of the devices, the load averages were sampled over time.

While running the Raspberry Pi 4B in full sync mode (Table 3) and serving light clients, the load averages were well below the

Table 2: Measured Data Latencies

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

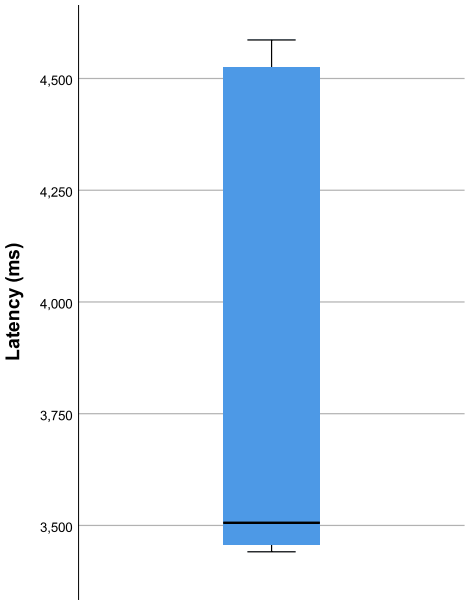

Figure 3: Measured Data Latency on RPi 4B Full Sync Node

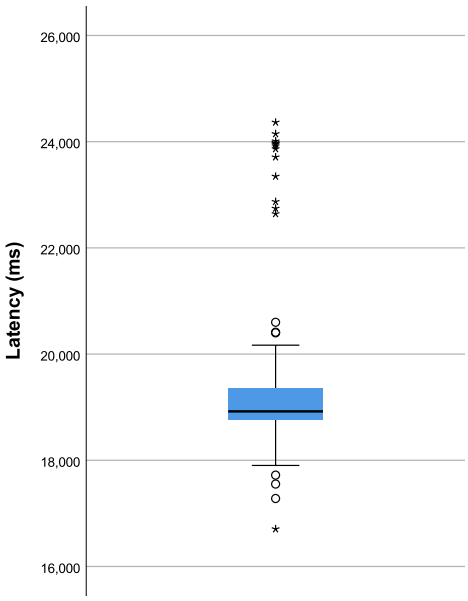

Figure 4: Measured Data Latency on RPi Zero W Light Sync Node

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 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 messured 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 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 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 busts 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.

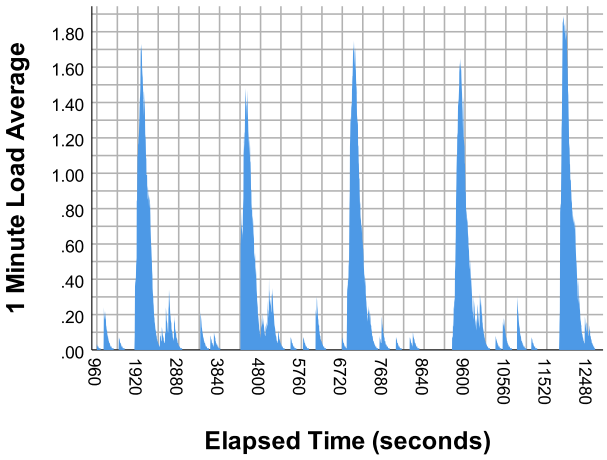

Figure 5: Load Average Periodicity on RPi 4B Full Sync Node, Light Serve

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

Table 3: RPi 4B (4 Core) 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

Sampled @ 6 second intervals. Discarded first 16 minutes of data. N = 1950 (after discarding).

Table 4: RPi 2B+ (4 Core) 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

Sampled @ 6 second intervals. Discarded first 16 minutes of data. N = 193 (after discarding).

Table 5: RPi Zero W x2 (1 Core) 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

Sampled @ intervals reflecting latency (~19.5 seconds). Discarded first 16 minutes of data. N = 2881 (after discarding).

Table 6: RPi Zero W x2 (1 Core) 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

Sampled @ intervals reflecting latency (~19.5 seconds). Discarded first 16 minutes of data. N = 1462 (after discarding).

can be rendered by the blockchain. These servies 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 parameters of the blockchain. These included a degree of control over latency, avoiding need for transaction fees, the use of proof-of-authority consensus. 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-work. This could be further leveraged in scaled up networks through the use of multiple segregated blockchains, to limit the rate a blockchain grows, 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 (directly internet 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.

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 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. LoRa devices, including the LoRa gateways are highly susceptible to jamming attacks due to the design of LoRa itself. LoRa is a low power transmission which uses license free frequencies. This makes it easy to overpower a LoRa transmission, preventing it from being correctly received by the intended recipient.

7 CONCLUSIONS

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. Blockchain technology can provide the backbone required to create a strong, unified security framework for a network of heterogeneous IoT systems. Utilizing blockchain solutions 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, this system delivers a superior resistance to the growing threat of Denial of Service attacks, by virtue of the distributed nature of blockchain systems. Although this system may not be as useful for some types of IoT devices, it can still offer other functionality in a Denial of Service resistant manner. The system presented caters well to wireless sensor networks and other delay tolerant applications, and with further development can be significantly improved. The deployment of IoT systems in smart homes, industry, and public services stand to offer many benefits to individuals, companies, and society as a whole, but only if the security of these systems is not neglected, to wit blockchain can provide the solution which is required.

ACKNOWLEDGMENTS

I would like to thank my supervisor, Dr. Dwight Makaroff for his guidance and support throughout this research project. I would also like to express my gratitude to Dr. Cameron Franc and Jarrod Pas for their help in understanding cryptography. Finally, I would like to thank Jason Goertzen for his support and encouragement to pursue an honours degree, which lead to this research project.

REFERENCES

[1] 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, 3–18.

[2] A.M. Antonopoulos. 2014. *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*. O'Reilly Media. https://books.google.ca/books?id=IXmrBQAAQBAJ

[3] J.-P. Aumasson, S. Fischer, S. Khazaei, W. Meier, and C. Rechberger. 2008. New features of Latin dances: Analysis of Salsa, ChaCha, and Rumba. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 5086 LNCS (2008), 470–488. https://doi.org/10.1007/978-3-540-71039-4_30

[4] Daniel J. Bernstein. 2004. Cache-timing attacks on AES. https://cr.yp.to/antiforgery/cachetiming-20050414.pdf

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

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

[7] Daniel J. Bernstein. 2008. *The Salsa20 Family of Stream Ciphers*. Springer Berlin Heidelberg, Berlin, Heidelberg, 84–97. https://doi.org/10.1007/978-3-540-68351-3_8

[8] Vitalik Buterin. 2014. A next-generation smart contract and decentralized application platform. *white paper* 3, 37 (2014).

[9] Vincent Daemen, Joan; Rijmen. 2003. AES Proposal: Rijndael. http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf

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

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

[12] J. Hoffstein, J. Pipher, and J.H. Silverman. 2014. *An Introduction to Mathematical Cryptography*. Springer New York. https://books.google.ca/books?id=cbl_BAAAQBAJ

[13] 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. https://doi.org/10.1007/s11416-017-0295-5

[14] Seyoung Huh, Sangrae Cho, and Soohyung Kim. 2017. Managing IoT devices using blockchain platform. In *2017 19th International Conference on Advanced Communication Technology (ICACT)*. Phoenix Park, PyeongChang, South Korea, 464–7. https://doi.org/10.23919/ICACT.2017.7890132

[15] 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

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

[17] 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 (ICCSE'17)*. Association for Computing Machinery, Beijing, China, 38–43. https://doi.org/10.1145/3126973.3126980

[18] Kerry Maletsky. 2015. RSA vs ECC comparison for embedded systems. *White Paper, Atmel* 5 (2015).

[19] Daniel Minoli and Benedict Occhiogrosso. 2018. Blockchain mechanisms for IoT security. *Internet of Things* 1-2 (2018), 1–13. https://doi.org/10.1016/j.iot.2018.05.002

[20] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf

[21] National Institute of Standards and Technology. 2001. *FIPS PUB 197: Announcing the ADVANCED ENCRYPTION STANDARD (AES)*. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

[22] 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. https://doi.org/10.1109/MCE.2018.2880806

[23] ARM (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

[24] 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, CA, USA, 1–6.

[25] Gavin Wood. 2019. ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER. https://ethereum.github.io/yellowpaper/paper.pdf