Securing Over–The–Air IoT Firmware Updates using Blockchain

Xinchi He, Sarra Alqahtani, Rose Gamble, Mauricio Papa
Tandy School of Computer Science, The University of Tulsa
Tulsa, OK, USA
{xinchi-he,sarra-alqahtani,gamble,mauricio-papa}@utulsa.edu

ABSTRACT

Over–the–air (OTA) firmware updates are very common in IoT (Internet of Things) devices with wireless capabilities. Although they are convenient, they may also be open to attack since physical access may not be needed. In addition, most frameworks use a centralized architecture to update a potentially large number of devices increasing the threat landscape. An alternative solution, that relies on a blockchain framework with smart contracts, is proposed in this paper to protect the integrity of the firmware update process. The proposed system is suitable for use in smart cities or scenarios with a large number of devices and service providers where nodes are authenticated, communications protected, and update conditions specified and enforced through smart contracts. A proof–of–concept system was implemented and tested using an open–source blockchain framework and a WiFi–capabable ESP8266–based board. The system was evaluated for scalability and response to denial of service (DoS) and man–in–the–middle (MitM) attacks. Preliminary experimental results show that the approach is feasible and a viable substitute for a centralized solution.

CCS CONCEPTS

• **Security and privacy** → *Embedded systems security.*

KEYWORDS

Firmware update, blockchain, Internet of Things, network security

ACM Reference Format:
Xinchi He, Sarra Alqahtani, Rose Gamble, Mauricio Papa. 2019. Securing Over–The–Air IoT Firmware Updates using Blockchain. In *INTERNATIONAL CONFERENCE ON OMNI-LAYER INTELLIGENT SYSTEMS (COINS), May 5–7, 2019, Crete, Greece.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3312614.3312649

1 INTRODUCTION

IoT devices are controlled and monitored using network connections over the Internet with their behavior and functionality largely determined by the underlying firmware. The firmware is the piece of software that resides on the hardware to support the basic device functions [16]. It is usually stored in EEPROM or flash memory [9].

Firmware updates occur periodically and are critical to IoT device operation as they often enhance security, add new features, fix existing bugs and support compliance for new protocols and standards [9]. The firmware update process, which needs to be secured, can be performed either through a direct physical connection or a remote wireless connection (OTA).

Some hardware manufacturers allow USB drives to plug in to the device to update the firmware [3] or program the hardware from a host through an onboard serial port, such as Arduino boards [1]. OTA firmware updates allow the device to download the update through its wireless network interface and then update itself [3]. For most IoT devices, this process is the preferred mechanism to apply a firmware update for several reasons: (i) it is possible to update devices without physical contact, (ii) convenience and (iii) cost-effectiveness [7].

Needless to say, the OTA firmware update process itself can still be compromised, i.e., a potential attacker does not need physical access to the device. For instance, an intruder could interfere with the process through a MitM or DoS attack. Securing the process is challenging because the limited hardware resources available on IoT devices make it difficult to adopt traditional and mature security protection mechanisms.

Most of the existing solutions for IoT OTA firmware updates adopt a client-server model in which device manufacturers use servers (possibly using cloud providers) to distribute firmware updates to IoT client devices. This centralized approach exhibits a single point of failure for both the availability and the integrity of the firmware update [4]. Relying on a blockchain framework, instead of a centralized approach, has the following advantages:

- Easy to keep track of all events (stored in the immutable ledger) associated with the firmware update
- Manufacturers can use smart contracts to specify update conditions in a flexible manner
- Distributed nature of the blockchain frameworks makes them more resilient to network failures and cyber attacks

In order to eliminate the single point of failure and make the process more robust, it is possible to replace the server side of the OTA update with a blockchain [4, 12, 13, 18]. The system proposed in this paper demonstrates not only the feasibility of the approach but also its resilience to cyber attacks. Blockchain is a widely popular, emerging technology that utilizes distributed immutable ledgers, consensus algorithms, and smart contracts to essentially provide an incorruptible digital ledger that can be used to record and validate any kind of transaction. For instance, Bitcoin uses blockchain to record cryptocurrency transactions. Similarly, our proposed system uses blockchain to protect firmware-related transactions.

In this paper, we propose a system that employs a permissioned blockchain to secure OTA firmware updates for IoT devices that could be deployed in smart cities or across different households. The

COINS, May 5–7, 2019, Crete, Greece

ACM ISBN 978-1-4503-6640-3/19/05…$15.00
https://doi.org/10.1145/3312614.3312649

approach deploys smart contracts to blockchain networks to verify the legitimacy of a firmware update and to handle failed verification instances that may be the result of a device under attack (such as DoS or MitM). Our proposed system eliminates the single point of failure present in centralized firmware update architectures by replacing it with a distributed blockchain solution.

The proposed system is validated using a prototype implementation that uses the Wemos D1 Mini board (ESP8266 chipset based) and the Hyperledger Fabric open-source blockchain framework. We evaluate the performance and scalability of our approach in a blockchain network with programmed simulations, while also testing resilience to different types of cyber attacks. Preliminary results show that the approach is feasible without compromising performance and can protect IoT devices against cyber attacks during the firmware update process.

2 BLOCKCHAIN BASICS AND JUSTIFICATION

Blockchain technologies have been used, researched, and validated in a number of different domains, such as cryptocurrency, healthcare, supply chain, and Internet of Things. There are mainly two types of blockchain architectures: public and permissioned [17]. Public blockchains, such as Bitcoin and Ethereum, allow any party to join the network freely by utilizing PoW (Proof of Work) or PoS (Proof of Stake) consensus algorithms for better security; an approach that sacrifices performance and flexibility. Permissioned blockchains, such as Hyperledger, use CAs (certificate authorities) to authenticate participants that use security tokens to join the network. In addition, most permissioned blockhains use Byzantine Fault Tolerance-based (BFT) consensus algorithms for better cost efficiency and improved performance [17]. BFT is a consensus protocol that ensures consistency and low latency even if some participants are compromised or misbehaving.

Because blockchains can be used to safeguard transaction records of any kind, its use has been explored in a number of domains that include firmware updates. In particular, authors in [4, 12, 13, 18] have investigated the use of blockchain distributed ledgers and consensus algorithms to improve availability and integrity during firmware updates over traditional centralized architectures that expose a single point of failure. Baza et al. [4] have worked in the vehicular domain to secure firmware updates using blockchain. Authors in [12, 13, 18] have proposed solutions applicable to generic IoT devices using public blockchains.

3 RELATED WORK

A four-layered smart city architecture, called CitySense, that uses blockchain technology is described and proposed by Ibba et al. [8]. Sensors and smart objects are identified as the physical layer of the architecture and constitute the foundation of the approach. The role of the blockchain can be used to ensure consistency of sensor data collected across the city [10]. Missing from these two architecture descriptions are (i) details regarding the management of the blockchain layer, (ii) interactions between the layers and (iii) a prototype implementation.

A few proposed solutions rely on a centralized client-server model. The main disadvantage of this type of architecture (as opposed to the use of a distributed architecture like blockchain) is that it offers a single point of failure. Choi et al. [6] have questioned the myth that firmware is usually more secure than software. For that reason, they promote the importance of validating and securing firmware updates in consumer device home networks. Their approach partitions the firmware into a fixed number of blocks and then uses hash chaining. Their solution, however, relies on consumer devices with more computational power than is usually found in IoT devices. Thus, the approach may impact an IoT device's ability to calculate the hash due to more limited resources (both in memory and computing capabilities). Kvarda et al. [11] propose a firmware update mechanism that uses hashing and PKI (Public Key Infrastructure) with functionality implemented both at the software and hardware levels. With this mechanism, integrity, authentication, and privacy are readily present to help secure firmware updates for IoT devices. Prada-Delgado et al. [15] demonstrate a new protocol that relies on physical unclonable functions (PUF) to provide a light-weight firmware update for IoT devices in a trustworthy manner. There are several advantages of using PUF to support IoT firmware updates: (i) no need to reveal information about shared secrets, (ii) devices can generate their own fresh keys, (iii) all communication goes over an encrypted channel to prevent sniffing, MitM and replay attacks.

Others have proposed distributed solutions that eliminate the single point of failure. Chandra et al. [5] introduce an OTA firmware update solution for IoT devices based on the Lightweight Mesh (LWMesh) network protocol. Similar to our proposed system, this solution does not rely on a centralized management service, avoiding a single point of failure. However, their approach is entirely dependent on the IoT platform using LWMesh and cannot be easily generalized for the entire domain.

Blockchain has been shown to be a systematic approach in eliminating single points of failure. For instance Lee and Lee [12] have proposed an alternative version of the Bitcoin blockchain [14] to be used for firmware updates. Nodes in their framework are assigned different roles in the process of verifying firmware updates. This solution, which assumes that PoW occurs at the device, has not been implemented or evaluated in practice. The PoW assumption imposes a heavy load on IoT devices (that already have very limited resources) to perform this task. Lim et al. [13] expand on the above approach by introducing a triangular trade protocol that increases the functionality of the platform. Their customized blockchain has been implemented from scratch using Python, as opposed to using available open-source solutions that can be easily adopted and generalized.

However, using public blockchain and PoW consensus algorithms may not be practical in a smart city domain or in scenarios with a moderately large number of sensors because such an approach is too resource intensive for IoT devices with limited computational capabilities. Our solution resolves this issue by delegating this function to an open-source permissioned blockchain framework.

4 APPROACH

Our solution centers around a modular, distributed, and layered architecture that uses a blockchain framework to validate updates. The architecture, the mechanism used to validate firmware updates,

and security properties describing the threat model and incident handling are described in the following three sections.

4.1 Architecture

The architecture proposed in this paper to secure IoT firmware updates consists of six layers: hardware layer, protocol layer, proxy layer, blockchain layer, service layer, and application layer as shown in Figure 1.

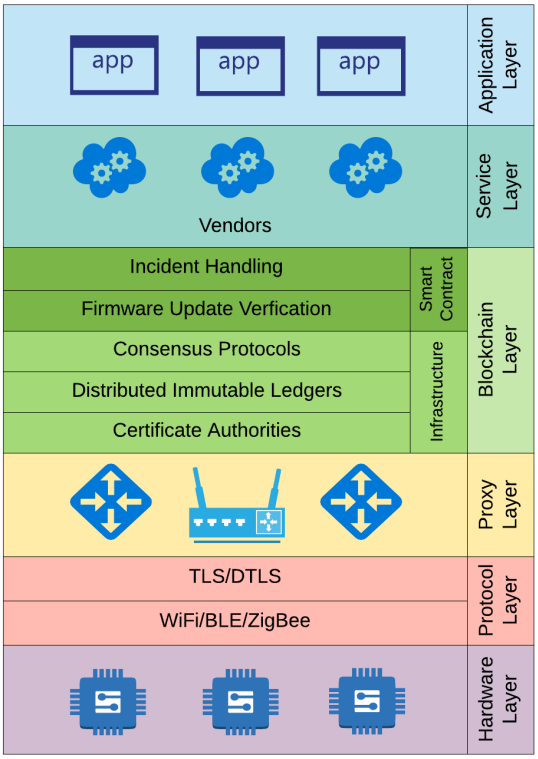

Figure 1: System architecture

The hardware and protocol layers together constitute the foundation of the system. The hardware layer consists of heterogeneous IoT devices, from different manufacturers that have been deployed across a community. IoT devices then communicate with the various elements of the architecture using protocols defined by the protocol layers (with support for encryption). These devices may communicate with their hosting services using WiFi, ZigBee or Bluetooth protocols. In order to protect the communications between vendor services or application services and IoT devices, we propose the use of TLS (Transport Layer Security) or DTLS (Datagram Transport Layer Security) whenever possible.

The proxy and blockchain layers act as the middle layer and are used to exchange, authenticate, and verify information. Ideally, the proxy layer is transparent [19] to interconnected IoT devices and components. Transparent gateways are used to facilitate connection of IoT devices to the Internet, translate traffic to HTTP (for non HTTP protocol based IoT devices, such as ZigBee and BLE) and interface with external APIs. Wireless routers can be used in the proxy layer to support IoT devices that can run HTTP/HTTPS natively, while a smart gateway (such as [2]) can be deployed to allow connection of BLE devices to the Internet.

The blockchain layer is the central piece of the approach. Two sub-layers, infrastructure and smart contracts, help incorporate the functionality needed to support a diverse number of providers, devices, and services. Components supporting the basic functionality of a permissioned blockchain network, that is CAs for authentication and peer nodes, form the infrastructure sublayer. Nodes in this layer are part of a consortium of IoT device manufacturers and application service providers that are jointly responsible for maintaining and hosting the distributed immutable ledger. Consensus protocols ensure the integrity of the transactions in the blockchain network so that the distributed ledger holds consistent records. The smart contract sublayer adds an extra level of functionality and intelligence to the system. The sublayer is responsible for implementing self-executable business logic (through smart contracts) to verify firmware update authenticity and to handle abnormal (and possibly malicious) incidents that may occur during the update process.

Finally, the service and application layers complete the picture by supporting a variety of use case scenarios. More specifically, the service layer supports vendor services that IoT device manufacturers want to distribute by a firmware update. The application layer deploys and supports end user applications such as a web-based blockchain transaction monitor or a mobile phone app that checks firmware update status.

4.2 Firmware Update Verification

The main goal of the proposed system is to enable verification of firmware updates in a large distributed system consisting of IoT devices, manufacturers, and vendors; a process that is particularly challenging as an OTA process. A robust verification mechanism is needed to ensure firmware legitimacy before any updates are accepted.

There are three parties involved in the verification process: the IoT device, the vendor service, and the blockchain:

(i) The IoT device is the actual end device in the system, responsible for collecting and relaying data (e.g., smart sensors) or executing commands (e.g., smart switches).
(ii) The vendor service provides the functionality needed to deliver firmware updates.
(iii) The blockchain is the permissioned blockchain network that supports the verification process through smart contracts.

The verification process of deploying a new firmware update being offered through a vendor service consists of the following six steps (Figure 2):

(1) Vendor service initiates a new blockchain transaction that contains the target IoT device information and SHA1 hash of the new firmware update. The result of this transaction (uniquely identified by a ***transaction_id***) is to add a new block to the distributed ledger.
(2) Vendor service pushes the firmware update binary to the target IoT device along with the ***transaction_id*** from the previous step.

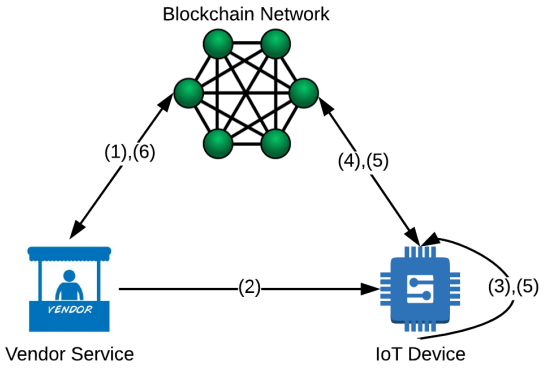

Figure 2: Firmware update verification process

(3) IoT device receives the OTA firmware update binary from the vendor service and it computes the SHA1 hash of the received binary.
(4) IoT device queries the distributed ledger along with the calculated SHA1 hash and the ***transaction_id*** to validate the update.
(5) If validation suceeeds, the IoT device applies the firmware update and sends a status update to the blockchain that registers the operation in the ledger. Otherwise, the OTA firmware update process will be aborted and a failure notice will be recorded in the ledger for such attempt (the device continues to use the same firmware it had before the update notification).
(6) Vendor service queries the blockchain to collect update statistics to determine further actions (for instance in case a portion of the devices fail to execute the update).

Each blockchain transaction in the distributed ledger is defined by the following attributes:

- ***transaction_id***: a unique ID to reference a firmware update initiated by a vendor service
- ***timestamp_send***: a time stamp recording the time a vendor service pushes the update to the blockchain
- ***timestamp_receive***: a time stamp recording the time when the blockchain accepts and commits the transaction to the ledger
- ***device_type***: the IoT device type (e.g. ESP8266)
- ***firmware_version***: the version of the firmware update
- ***firmware_hash***: SHA1 hash of the firmware update binary as provided by the vendor service
- ***status***: to indicate whether the targeted IoT device succeeded in updating the firmware. Its initial value is "unverified"
- ***failed_attempts***: a counter recording the number of failed firmware update attempts.

The ledger is only used to keep a record of transactions, in this case firmware update requests, but (by default) it cannot enforce the verification process. This lack of enforcement is why smart contracts are needed. They enforce execution of the verification process within the blockchain network.

Smart contracts can perform three operations: `push()`, `verify()` and `query()`. A `push()` operation is used by a vendor service to create the initial transaction for a new firmware update request. An IoT device calls `verify()` to verify a firmware update legitimacy. Moreover, a vendor service may monitor the firmware update status by using `query()` operations anytime after the update request. This operation would typically provide update status information including information about devices that could not successfully run the update, how many attempts were made, and other useful diagnostic information.

status and ***failed_attempts*** are "updated" by the smart contracts to document success or failure. While any updates would add immutable records to the ledger, these two attributes would support queries that report the entire history associated with the firmware update requests (i.e., a set of records that share the ***transaction_id*** but have different ***status*** and ***failed_attempts*** values).

4.3 Threat Model and Incident Handling

An important part of the process of validating firmware updates is to consider the system's ability to deal with cyber attacks. Permissioned blockchains already have built-in capabilities to protect communications between external parties and the blockchain network via CAs with encryption enabled.

Our solution assumes that vendor services credentials to authenticate to the blockchain via a CA are not compromised. This assumption is not unreasonable considering that a vendor service would be backed by a large organization and expected to have appropriate security controls. On the other hand, IoT devices are considered more vulnerable to DoS and MitM attacks precisely because of their limited hardware resources.

A DoS attack could temporarily disable the availability of IoT devices. In this case, a vendor service would not be able to push the firmware update binary to the IoT devices. As a result, the new firmware update would not be delivered as scheduled. A MitM attack to the IoT devices could alter the content of the binary associated with the firmware by injecting malicious code.

The verification algorithm used by the proposed system (Algorithm 1) helps mitigate the risk of both DoS and MitM attacks by using two types of thresholds. A time-based threshold defines a valid time window for the update verification to complete. Assuming that the operation occurs within this time window, a count-based threshold is used to impose an upper bound on the number of consecutive failed attempts (to avoid a case where a device is stuck in an update loop).

The algorithm then has four possible outcomes:

(i) ***time window has expired*** (line 21) is returned when the time window has been exceeded, perhaps the result of a DoS attack. Note that a vendor service may learn about update failures by simply using a `query()` operation (as defined in the smart contract). In case an update fails for a particular device, the vendor service is responsible for re-initiating the operation.
(ii) ***too many failed attempts*** (line 18) is returned to indicate too many consecutive failures or attempts, as this could be the result of a device under attack (perhaps by overloading the verification interface).
(iii) ***hash code mismatch*** (line 15) is returned when the hash calculated by the IoT device does not match the associated

Algorithm 1: Firmware update verification with thresholds

```
1  function verify() (tid, cf_hash);
   Input  : transaction ID tid, calculated firmware SHA1 hash
            cf_hash
   Output : code of indicating verification status
2  transaction = retrieve(tid);
3  f_hash = transaction.firmware_hash;
4  t_receive = transaction.timestamp_receive;
5  f_counter = transaction.failed_attempts;
6  if current_time − t_receive < time_threshold then
7      if f_counter < counter_threshold then
8          if cfhash == f_hash then
9              transaction.status == "verified";
10             update(tid, transaction);
11             return success code;
12         else
13             transaction.failed_attempts++;
14             update(tid, transaction);
15             return hash code mismatch code;
16         end
17     else
18         return too many failed attempts code;
19     end
20 else
21     return time window has expired code;
22 end
```

record in the ledger. This event could be the result of a MitM attack attempting to corrupt the integrity of the firmware update.

(iv) ***success*** (line 11) is returned as an indication that the verification process has completed correctly in a timely fashion.

5 IMPLEMENTATION

A proof-of-concept of the entire system (Figure 3) was implemented using an ESP8266-based IoT board, Hyperledger Fabric (an open-source permissioned blockchain framework) and a web-service based vendor service.

5.1 IoT Chip (ESP8266)

A low-cost, quarter-sized (25.6mm x 34.2mm) IoT board Wemos D1 Mini (Figure 4) was used as the IoT device. This popular board was chosen because it is WiFi-capable and Arduino-compatible (easily extensible with a ample selection of sensors and shields). The Wemos D1 Mini runs at 80Mhz and has 4MB of non-volatile flash memory.

5.1.1 OTA Update Library and Limitations. A special library allows developers to perform OTA updates rather than using the on-board USB port. This library exposes the ESP8266 chip as an HTTP server that can accept firmware uploads via HTTP POST requests. Due to very limited available memory (~50KB RAM), firmware updates must be processed in fragments. Firmware update fragments are written to a temporary buffer in Flash memory (Figure 5) through

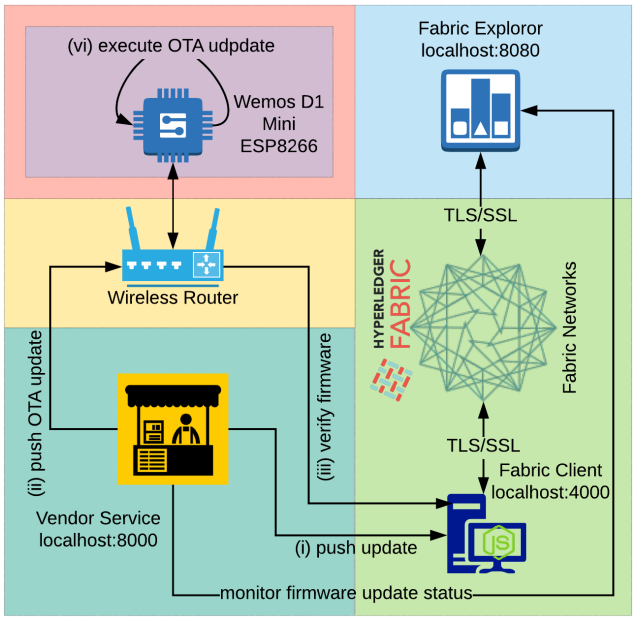

Figure 3: End-to-end implementation overview

Figure 4: Wemos D1 Mini (middle) and its accessories: relay shield (left) and temperature/humidity sensors shield (right)

the ESP8266 built-in *Updater* library. The device reboots once the update is completed.

It is important to note that this library, as provided by the ESP8266 community, has two significant limitations:

- Only local MD5 validation is supported. Firmware update binaries and their associated MD5 checksums could be intercepted and altered by an adversary and the target device could then be compromised. Furthermore, local MD5 validation is not always enabled by default.
- The default update process does not support verification by an external party before committing to the new firmware version.

For these reasons, the update library had to be modified as described below.

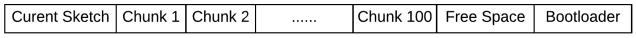

Figure 5: Firmware in the Flash

Table 1: Vendor service APIs

Route	Request Type	Request Body
/put_update	POST	JSON
/push_binary	POST	transaction ID, firmware
/query_status	GET	transaction ID

5.1.2 Firmware SHA1 Hash. The OTA update library was modified by intercepting the firmware update process to support blockchain–based SHA1 hash verification before the update is finalized. SHA1 is chosen as the hash algorithm because it ensures better integrity properties than MD5 and is more efficient than SHA256, given the limited resources available on the Wemos board.

There are two methods to calculate the SHA1 hash: (i) merging all the fragments and calculating it at once or (ii) aggregating partial results from each fragment. Calculating it at once is simpler but not feasible for the ESP8266 chip due to insufficient memory. In addition, the ESP8266 *Hash* library is not capable of calculating the hash using the second method (aggregating partial results). Therefore, the library was modified and extended to enable calculation of the SHA1 hash on–the–fly using firmware fragments and the capability to interface with blockchain RESTful APIs for verification.

5.2 Vendor Service

The vendor service was implemented using Golang and the *net/http* library. It is a simple web service capable of three operations (Table 1):

- A JSON object of firmware update transaction attributes can be posted to the blockchain ledger through a smart contract
- Deliver a firmware update binary along with the blockchain transaction ID
- Monitor update status

5.3 Blockchain

Hyperledger Fabric 1.1.0 was used as the permissioned blockchain for our proof-of-concept implementation. It is an open-source platform that scales well and has rich functionality. *DockerCompose* templates are used to configure the structure of the blockchain network and its participants.

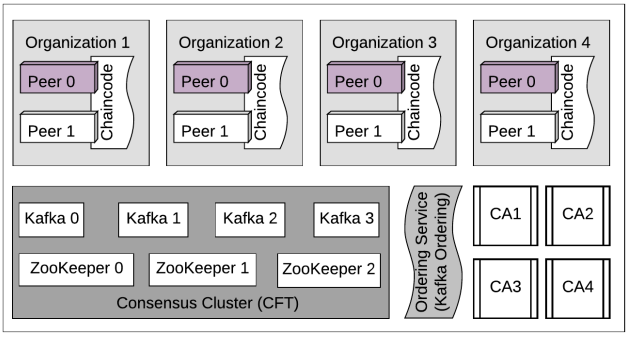

Figure 6: Hyperledger Fabric Network

The topology of the implemented Hyperledger Fabric can be seen in Figure 6. Four organizations are defined for IoT device manufacturers, each holding two peer nodes for the ledger. *Peer*0 (the purple node) in each organization is the anchor node used to bridge communications among organizations. Chaincode (a smart contract implementation) is installed on all peer nodes. Since a PBFT (Practical Byzantine Fault Tolerance) consensus algorithm has not been implemented in Hyperledger Fabric 1.1.0, a Kafka ordering service is launched using Apache Kafka and ZooKeeper ensembles to coordinate distributed communications within the blockchain network. Kafka ordering is a CFT-based (Crash Fault Tolerance) consensus algorithm that provides robustness and scalability. Four CAs work as the gateways to authenticate the parties joining the blockchain network with security tokens.

The three smart-contract operations described in Section 4.2, are exposed using an HTTP RESTful API (Table 2) through a Hyperledger Fabric Client (version 1.0.2 of the client was used). This client uses a Node.js–based (version 8.12) library that allows the blockchain network to expose its functionality (such as querying and invoking the Chaincode). The vendor service can also use the API to query the status of a firmware update associated with a particular transaction ID. Fields ***status*** and ***failed_attempts*** are used to indicate results.

The Wemos D1 Mini was programmed to use the API, using a transaction ID and calculated firmware hash, to verify the firmware update. The four possible outcomes are ***time window has expired***, ***too many failed attempts***, ***hash code mismatch*** and ***success*** (see Section 4.3).

In addition, Hyperledger Fabric Explorer 0.3.5 was deployed (as a blockchain application) to help us (as blockchain consortium administrators) visualize and monitor ongoing transactions and blocks.

6 EXPERIMENTAL RESULTS AND EVALUATION

Two types of attacks (DoS and MitM) were performed against the Wemos D1 Mini to test system response and its ability to protect the integrity of the OTA firmware update process. Recall that two thresholds are defined in Algorithm 1 to control the impact of these attacks. A time threshold of 60 seconds is set to complete the firmware update request while the maximum number of failed attempts was set to 3. The current firmware update version running on the Wemos D1 Mini was set to 1.0 and the OTA firmware update aims to bring it to version 2.0. Furthermore, to evaluate performance and scalability, a simulation was programmed to measure the response time of the RESTful APIs when there are 1 to 10000 transactions in the ledger.

6.1 Denial-of-service Attack

The current version of the OTA firmware update library for the ESP8266 used by the Wemos D1 Mini is vulnerable to the following exploit. If the uploaded firmware format is invalid, the board reboots automatically. This vulnerability is used to conduct a DoS attack on the IoT device. The goal of the attack is to verify that even if the IoT device is under a DoS attack, the terms of the smart contract will not be violated. A *bash* script was built to continuously upload

Table 2: RESTful APIs for smart contract

Smart Contract	Request Type	Parameters	Response
`push()`	POST	transaction attributes as JSON object	transaction ID
`verify()`	POST	transaction ID, firmware SHA1 hash	verification results
`query()`	GET	transaction ID	verification status

a small size file that contains random characters that would keep the Wemos D1 Mini in a rebooting loop for as along as the script is run.

The attack is then performed as follows:

- Execute the script for two minutes to DoS the Wemos D1 Mini, effectively bringing it offline
- While the Wemos D1 Mini is offline, use the vendor service to push a new firmware update
- Have the vendor push the firmware update at the end of the two minutes when the Wemos D1 Mini is back online

The system behaved as expected and the final firmware update push was rejected because the time threshold of 60 seconds was exceeded. Figure 7 displays the output from the Arduino serial console for the ESP8266 board under attack. Line 20 shows that the transaction has expired, thus the verification of the firmware update fails and the update process is aborted. While the firmware update is not performed, the system resisted the attack because the terms of the smart contract with respect to the time threshold were preserved.

```
1  Booting Sketch...
2  firmware v1.0
3
4  Update: v2.ino.bin
5  sleep disable
6  Chunk 1 size: 2048 Hash: 22c47838cb13932a3ac369f14e3532cf6a99b399
7  Chunk 2 size: 2048 Hash: eb2aacf95642ff895ded64ad4ca86ea6c3c7ec84
8  Chunk 3 size: 2048 Hash: 0e682492ab3a819805c7a3043cd6caa441d69de3
9  Chunk 4 size: 2048 Hash: 181652e107cd162d3356cfd1ce57b322a1f5c14b
10 Chunk 5 size: 2048 Hash: fa50f70248c8512a7a2a323b89694ef52229712a
11 ...
12 Chunk 145 size: 2048 Hash: 01d041ca710e15c218736471187b08dbf714ebfd
13 Chunk 146 size: 144 Hash: 60688f5c953f2fd247fe31dda3c97bad14b3b569
14 final hash is 457de643c3113667f18660bf12c999db721a3fc4
15 Verifying with blockchain...
16 Hyperledger Client Token: eyJhbGciOiJIUzI1NiIsInR5cCI6I...O9itsGBM
17 {"fcn": "verify",
18 "args": ["ESP00001", "457de643c3113667f18660bf12c999db721a3fc4"]}
19 Failed to order the transaction.
20 Message: ERROR!!! transaction is expired.
21 Verification FAILED! OTA update aborted! Will REBOOT!
22
23 Booting Sketch...
24 firmware v1.0
```

Figure 7: Partial serial console output for DoS attack

6.2 Man-in-the-middle Attack

A MitM attack is conducted while the vendor service is pushing the firmware update to the IoT device to evaluate system response. Two tools, *mitmproxy* and *SwitchyOmega*, were used to conduct the attack. *mitmproxy* is a command-line based MitM attack proxy server with built-in functions to intercept traffic with customized filters. *SwitchyOmega* is a Chrome web browser extension that helps route HTTP traffic through a proxy server.

First, *mitmproxy* (using `~q & ~t "multipart/form-data"` as a filter) is launched on port 8080 to intercept firmware upload traffic to the Wemos D1 Mini. A new profile is then configured with *SwitchyOmega* to use the *mitmproxy* proxy server already running on port 8080. At this point, characters are randomly injected in the body of the message using the in-app editor of *mitmproxy* to simulate malicious code injection.

```
1  Booting Sketch...
2  firmware v1.0
3
4  Update: v2.ino.bin
5  sleep disable
6  Chunk 1 size: 2048 Hash: 22c47838cb13932a3ac369f14e3532cf6a99b399
7  Chunk 2 size: 2048 Hash: eb2aacf95642ff895ded64ad4ca86ea6c3c7ec84
8  Chunk 3 size: 2048 Hash: 0e682492ab3a819805c7a3043cd6caa441d69de3
9  Chunk 4 size: 2048 Hash: 181652e107cd162d3356cfd1ce57b322a1f5c14b
10 Chunk 5 size: 2048 Hash: fa50f70248c8512a7a2a323b89694ef52229712a
11 ...
12 Chunk 145 size: 2048 Hash: 42734ab9b4cf8bf26a4ca06667e67d404bdb57f9
13 Chunk 146 size: 148 Hash: 2ddf057494744c9cb5d647afc62424940bcad109
14 final hash is 6026c76c51d4592124ded04292b07591e0165633
15 Verifying with blockchain...
16 Hyperledger Client Token: eyJhbGciOiJIUzI1NiIsInR5cCI6Ik...PsCYTmQ
17 {"fcn": "verify",
18 "args": ["ESP00001", "6026c76c51d4592124ded04292b07591e0165633"]}
19 Transaction ID:
20 969a9557dcb03e7e4d5c7076152567fac054c372981c99123585a26872612980
21 {"status":"unverified","failed_attempts":1}
22 Firmware hash does NOT match. Failed attempts made: 1
23 Verification FAILED! OTA update aborted! Will REBOOT!
24
25 Booting Sketch...
26 firmware v1.0
```

Figure 8: Partial serial console output for MitM attack

Figure 8 shows output from the Arduino serial console indicating that it has failed to verify the firmware update because the calculated hash (Line 14) does not match the hash of the original firmware update binary `457de643c3113667f18660bf12c999db721a3fc4` as obtained from the Hyperledger Fabric.

We also conducted the same attack multiple times within the preset time threshold of 60 seconds to evaluate response when the count threshold (for update attempts) is exceeded. As expected, the "too many failed attempts" error message was shown in the output for the 4th attempt when using a preset quantity threshold of 3. Once this error is detected, and even if the correct firmware update was pushed, the update would still fail. Once again, this scenario shows the proposed system preserves its integrity during a MitM attack with respect to the terms specified in the smart contract.

6.3 Performance and Scalability

Since one of the goals is to design a system that can be used in a smart city or a domain with a large number of devices, it is important to evaluate scalability. A total of three different probes

are deployed, tasked with measuring response time for the RESTful API `push()`, `verify()`, and `query()` operations from 1 to 10,000 transactions in the ledger. The Hyperledger Fabric keeps a cache image of the entire state of the system to evaluate any single query regardless of size. We expect the response time to converge and stabilize even as the number of records increase. A Dell Precision Tower workstation with dual Intel Xeon E5 processors (12 cores), 32G of RAM, 512G of SSD that runs Gentoo Linux with LTS kernel 4.14.83 was used as the testing platform.

After collecting raw data from the probes, the average response times are computed for each API for 100, 200, 500, 1000, 2000, 5000 and 10000 transactions in the ledger. Figure 9 shows the average response time in milliseconds for `push()`, `verify()` and `query()`. The average response time of `push()` is slightly larger than that of `verify()` because of the extra overhead associated with transaction attributes. The results matched the expectations, showing that the proposed system has good scalability properties.

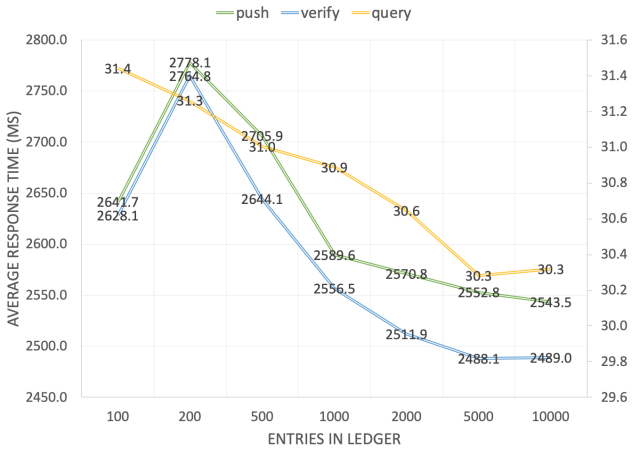

Figure 9: API average response time

7 CONCLUSIONS AND FUTURE WORK

A system is proposed to preserve the integrity of IoT OTA firmware updates using a permissioned blockchain rather than the more commonly used centralized solution. The approach is a viable alternative, capable of supporting a heterogeneous environment involving a large number of IoT devices and a variety of vendor or service providers. Authentication and security services help protect the integrity of the system, while smart contracts are used to describe enforceable terms associated with the firmware update.

A proof-of-concept system was built to demonstrate the feasibility of the approach using the Hyperledger Fabric (blockchain), Chaincode (smart contracts) and the Wemos D1 Mini board (ESP8266–based IoT device). Preliminary results are encouraging, both from an operational and cyber security point of view. The system scales well and smart contracts offer the flexibility needed to incorporate logic and conditions. DoS and MitM attacks were used to show that the terms and conditions established by the smart contracts were preserved even when the system is under attack.

Future work will concentrate on (i) incorporating a diverse variety of IoT devices, (ii) adding a layer of intelligence that facilitates the creation of smart contracts and (iii) improving monitoring capabilities of the entire system.

REFERENCES

[1] [n. d.]. Arduino–Home. https://www.arduino.cc/. Accessed: 2019-01-11.

[2] [n. d.]. TIDC–BLE–TO–WIFI–IOT–GATEWAY Bluetooth to Wi–Fi IoT Gateway Referece Design | Ti.com. http://www.ti.com/tool/TIDC-BLE-TO-WIFI-IOT-GATEWAY. Accessed: 2019-01-11.

[3] Carlos E. Andrade, Simon D. Byers, Vijay Gopalakrishnan, Emir Halepovic, Milap Majmundar, David J. Poole, Lien K. Tran, and Christopher T. Volinsky. 2017. Managing Massive Firmware-Over-The-Air Updates for Connected Cars in Cellular Networks. In *Proceedings of the 2Nd ACM International Workshop on Smart, Autonomous, and Connected Vehicular Systems and Services (CarSys '17)*. ACM, New York, NY, USA, 65–72. https://doi.org/10.1145/3131944.3131953

[4] Mohamed Baza, Mahmoud Nabil, Noureddine Lasla, Kemal Fidan, Mohamed Mahmoud, and Mohamed Abdallah. 2018. Blockchain-based Firmware Update Scheme Tailored for Autonomous Vehicles. arXiv:arXiv:1811.05905 https://arxiv.org/pdf/1811.05905.pdf

[5] Hans Chandra, Erwin Anggadjaja, Pranata Setya Wijaya, and Edy Gunawan. 2016. Internet of Things: Over-the-Air (OTA) firmware update in Lightweight mesh network protocol for smart urban development. In *2016 22nd Asia-Pacific Conference on Communications (APCC)*. 115–118. https://doi.org/10.1109/APCC.2016.7581459

[6] Byung-Chui Choi, Seoung-Hyeon Lee, Jung-Chan Na, and Jong-Hyouk Lee. 2016. Secure firmware validation and update for consumer devices in home networking. *IEEE Transactions on Consumer Electronics* 62, 1 (February 2016), 39–44. https://doi.org/10.1109/TCE.2016.7448561

[7] Giovani Gracioli and Antônio A. Fröhlich. 2008. An Operating System Infrastructure for Remote Code Update in Deeply Embedded Systems. In *Proceedings of the 1st International Workshop on Hot Topics in Software Upgrades (HotSWUp '08)*. ACM, New York, NY, USA, Article 3, 5 pages. https://doi.org/10.1145/1490283.1490287

[8] Simona Ibba, Andrea Pinna, Matteo Seu, and Filippo Eros Pani. 2017. CitySense: Blockchain-oriented Smart Cities. In *Proceedings of the XP2017 Scientific Workshops (XP '17)*. ACM, New York, NY, USA, Article 12, 5 pages. https://doi.org/10.1145/3120459.3120472

[9] Jinsik Kim and Pai H. Chou. 2010. Energy-Efficient Progressive Remote Update for Flash-Based Firmware of Networked Embedded Systems. *ACM Trans. Des. Autom. Electron. Syst.* 16, 1, Article 7 (Nov. 2010), 26 pages. https://doi.org/10.1145/1870109.1870116

[10] Alexander Kuzmin. 2017. Blockchain-based structures for a secure and operate IoT. In *2017 Internet of Things Business Models, Users, and Networks*. 1–7. https://doi.org/10.1109/CTTE.2017.8260937

[11] Lukas Kvarda, Pavel Hnyk, Lukas Vojtech, and M Neruda. 2017. Software Implementation of Secure Firmware Update in IoT Concept. *Advances in Electrical and Electronic Engineering* 15 (11 2017). https://doi.org/10.15598/aeee.v15i4.2467

[12] Boohyung Lee and Jong-Hyouk Lee. 2017. Blockchain-based Secure Firmware Update for Embedded Devices in an Internet of Things Environment. *J. Supercomput.* 73, 3 (March 2017), 1152–1167. https://doi.org/10.1007/s11227-016-1870-0

[13] Jea-Min Lim, Youngpil Kim, and Chuck Yoo. 2018. ChainVeri: Blockchain-based Firmware Verification System for IoT environment. In *2018 IEEE International Conference on Blockchain (Blockchain-2018)*. 1050–1056. https://doi.org/10.1109/Cybermatics_2018.2018.00194

[14] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system, http://bitcoin.org/bitcoin.pdf.

[15] M. A. Prada-Delgado, A. Vázquez-Reyes, and I. Baturone. 2017. Trustworthy firmware update for Internet-of-Thing Devices using physical unclonable functions. In *2017 Global Internet of Things Summit (GIoTS)*. 1–5. https://doi.org/10.1109/GIOTS.2017.8016282

[16] Chu Jay Tan, Junita Mohamad-Saleh, Khairu Anuar Mohamed Zain, and Zulfiqar Ali Abd. Aziz. 2017. Review on Firmware. In *Proceedings of the International Conference on Imaging, Signal Processing and Communication (ICISPC 2017)*. ACM, New York, NY, USA, 186–190. https://doi.org/10.1145/3132300.3132337

[17] Xiwei Xu, Ingo Weber, Mark Staples, Liming Zhu, Jan Bosch, Len Bass, Cesare Pautasso, and Paul Rimba. 2017. A Taxonomy of Blockchain-Based Systems for Architecture Design. In *2017 IEEE International Conference on Software Architecture (ICSA)*. 243–252. https://doi.org/10.1109/ICSA.2017.33

[18] Alexander Yohan, Nai-Wei Lo, and Suttawee Achawapong. 2018. Blockchain-based Firmware Update Framework for Internet-of-Things Environment. In *Proceedings of the 2018 International Conference on Information and Knowledge Engineering (IKE '18)*. 151–155. https://csce.ucmss.com/cr/books/2018/LFS/CSREA2018/IKE9004.pdf

[19] Kazım Rıfat Özyılmaz and Arda Yurdakul. 2017. Work-in-progress: integrating low-power IoT devices to a blockchain-based infrastructure. In *2017 International Conference on Embedded Software (EMSOFT)*. 1–2. https://doi.org/10.1145/3125503.3125628