IoTCop: A Blockchain-Based Monitoring Framework for Detection and Isolation of Malicious Devices in Internet-of-Things Systems

Sreenivas Sudarshan Seshadri, David Rodriguez, *Member, IEEE*, Mukunda Subedi, Kim-Kwang Raymond Choo, *Senior Member, IEEE*, Sara Ahmed, *Member, IEEE*, Qian Chen, *Member, IEEE*, and Junghee Lee, *Member, IEEE*

***Abstract*—Unlike conventional servers housed in a centralized and secured indoor environment (e.g., data centers), Internet-of-Things (IoT) devices such as sensor/actuator are geographically distributed and may be closely located to the physical systems where IoT devices are utilized. However, the resource-constrained nature of IoT devices limits their capacity to deploy sophisticated security solutions. The proposed approach assumes that a device can be compromised and hence, the need to be able to automatically isolate the compromised device(s). In order to enforce security policies even when devices are compromised, we propose using blockchain in the monitoring framework. Unlike existing centralized or distributed security solutions (which do not consider the possibility that the solutions themselves can be compromised), the proposed blockchain-based framework can enforce the security policies as long as a majority of the devices are not compromised. By employing the permissioned blockchain (Hyperledger Fabric) and add-on hardware modules, the proposed framework offers significantly lower latency and overhead compared to permissionless blockchain frameworks (e.g., Ethereum) and allows existing IoT devices to join the framework without modification.**

***Index Terms*—Blockchain, Internet-of-Things (IoT), monitoring, security.**

I. Introduction

ADVANCES in semiconductor and communication technologies and the interconnectivity of our society have partly contributed to the popularity of networked embedded systems, which are often found in cyber–physical systems and Internet-of-Things (IoT) devices. As more IoT devices are connected to different systems in our society (e.g., those in the critical infrastructure sectors), it is important to secure such devices, or isolate compromised devices, to prevent them from being used as a launch pad for subsequent attacks. For example, consumer IoT devices had reportedly been used to commit a Distributed Denial of Service (DDoS) attack to websites, such as Twitter, Netflix, Spotify, Airbnb, Reddit, Etsy, SoundCloud, and The New York Times on October 21st, 2016 [1]. Recent studies from security organizations such as Trend Micro [2], [3] have also suggested that IoT devices are increasingly been targeted by cyber criminals.

It is challenging, however, to prevent IoT devices from being compromised, particularly devices that are placed in publicly accessible places (unlike servers that are typically maintained in a secure place like a data center) [4]. Since the physical security of IoT devices cannot always be guaranteed, they are more easily compromised and consequently, fall under the full control of an adversary. The resource-constrained nature of these devices compounds the challenge of preventing attacks to their firmware and hardware components.

Therefore, in this article, our proposed approach assumes that any individual device can be compromised at some stage and *automatically isolates the device* if this happens. This approach differs from existing security frameworks, which are generally focused on building a secure device. To enforce security policies even when there exist compromised devices, blockchain is employed as the underlying mechanism in our proposed approach. In other words, as long as a majority of the devices in the network are not compromised, the security of the entire system can be guaranteed. Therefore, compared to existing centralized or hierarchical approaches [5]–[8], this blockchain-based approach offers a higher level of assurance. Existing approaches such as those based on peer to peer [9] may avoid the single-point failure, but they do not support runtime policy updates since such approaches do not employ a consensus protocol (unlike a blockchain-based approach). Furthermore, the proposed approach monitors *all network traffic* and automatically isolates a malicious device by blocking all outgoing traffic from a particular device.

To apply blockchain to IoT systems, the following challenges must be addressed.

Manuscript received April 11, 2020; revised August 12, 2020; accepted September 2, 2020. Date of publication September 7, 2020; date of current version February 19, 2021. This work was supported by a Korea University Grant. *(Corresponding author: Junghee Lee.)*

Sreenivas Sudarshan Seshadri, David Rodriguez, Sara Ahmed, and Qian Chen are with the Department of Electrical and Computer Engineering, University of Texas at San Antonio, San Antonio, TX 78249 USA (e-mail: sreenivassudarshan.seshadri@my.utsa.edu; drodriguez1724@yahoo.com; sara.ahmed@utsa.edu; guenevereqian.chen@utsa.edu).

Mukunda Subedi is with Tata Consultancy Services, Mumbai, India (e-mail: merc.subedi@gmail.com).

Kim-Kwang Raymond Choo is with the Department of Electrical and Computer Engineering and the Department of Information Systems and Cyber Security, University of Texas at San Antonio, San Antonio, TX 78249 USA (e-mail: raymond.choo@fulbrightmail.org).

Junghee Lee is with the School of Cyber Security and the Institute of Cyber Security and Privacy, Korea University, Seoul 02841, South Korea (e-mail: jlee@korea.ac.kr).

Digital Object Identifier 10.1109/JIOT.2020.3022033

1) *(Challenge 1) Latency:* In permissionless blockchain frameworks, such as Bitcoin and Ethereum, it takes between 1 and 10 min to reach consensus [10]. Unlike blockchain-based currency, such latency is not acceptable for time- and delay-sensitive applications such as a smart grid.
2) *(Challenge 2) Applicability:* An IoT system generally consists of different types of devices from different manufacturers. Hence, it is not practical to assume that all devices support the same blockchain framework.
3) *(Challenge 3) Resource Constraints:* For the blockchain to be effective, there must be a sufficient number of participating devices. However, due to the resource-constrained nature of IoT devices, such as sensors and actuators, they are unlikely to be suitable blockchain client candidates.

In our approach, these three challenges are addressed by employing *Hyperledger Fabric* and an *add-on hardware module* to implement the proposed blockchain client. As a proof of concept, we apply the proposed approach to the "building automation and control network" (BACnet). One example security policy we implement for BACnet is a whitelisting technique. Its average latency in IoTCop is 128.95 ms, whereas it takes 9.6 s if the same whitelisting is implemented on the Ethereum blockchain framework. In addition, the add-on hardware modules allow existing commercial off-the-shelf (COTS) devices to join the framework without modification. The computation demand of Hyperledger Fabric is significantly less than that of permissionless blockchain frameworks because the former uses the practical Byzantine fault tolerance (PBFT) protocol for consensus.

In other words, our proposed framework offers high flexibility, where resource-constrained devices may run only a part of blockchain clients and resource-capable devices (e.g., servers and desktops) can run blockchain clients on behalf of the resource-constrained devices. We also remark that IoTCop is the first attempt to introduce a general IoT monitoring framework using blockchain. We also formally analyze the security of IoTCop and demonstrate its efficiency by evaluating the prototype and an analytical model.

The remainder of this article is organized as follows. Section II presents our research motivations. After reviewing the related literature in Section III, we present an overview of proposed framework and its technical details in Sections IV and V. A case study and its evaluation results are presented in Sections VI and VII. Finally, Section VIII concludes this article.

II. Motivations and Background

A. Motivations

IoT systems are characterized by their capability to contact with the physical world, for example, via sensors and actuators. Sensors, actuators, and many other (inexpensive) IoT devices usually have a small processor that handles control of such devices and their network connectivity. Thus, such devices are susceptible to both hardware and software (firmware) attacks, and their resource constraints limit the adoption of full-fledged security solutions [11]–[13].

Attack scenarios can be classified based on their target, as shown in Fig. 1 [11]. Countermeasures to these attack scenarios often assume that the *underlying* components are trustworthy. Secure boot [14], [15], for example, is a representative countermeasure to malicious firmware attacks. This approach typically assumes the underlying hardware cannot be compromised. Another example is the encryption of a network message. If the end-point devices (source and destination) are compromised, the confidentiality of an encrypted message cannot be guaranteed too. Therefore, the security of devices is critical because it is the foundation of network and system security. Here, devices refer to end-point sensor/actuator devices, as well as all components in the network, such as gateways, routers, access points, and edge devices (e.g., local servers).

Unlike traditional computer networks, it is challenging to guarantee the physical security of devices in IoT systems. As previously discussed, servers may be maintained in a secure place such as a data center, but IoT devices are located in the environment they need to interact with. Furthermore, their resource-constrained nature limits their defensive capability.

The assumption that an individual device can be compromised and automated isolation of compromised device in our approach is analogous to governments isolating individuals convicted of a crime from the society (e.g., imprisonment). Clearly, we still need to make every effort to prevent devices from being compromised (malicious firmware or physical attacks), although their limited resources and lack of physical security compound the challenge of securing such devices. Not being able to fully trust individual devices in a network will have implications on the overall security of the network.

Therefore, the proposed approach seeks to ensure that an IoT system consists of only *trusted* devices by isolating *untrusted* devices when they are detected. Here, a trusted device means a device that complies with all *security policies* of the system. Although we are primarily focusing on mitigating malicious firmware and physical attacks, our approach can also address other security threats in a network and system.

If we assume an individual device that is part of a system can be compromised, then naturally we need to determine how to consistently enforce security policies. Many existing countermeasures employ a centralized approach [5]–[8], where a powerful central server (e.g., cloud or edge node) monitors the behavior and network traffic of all devices. This centralized approach, however, has a risk of single-point failure. If the central entity does not work for some reason (cyberattack, network problem, sudden power failure, physical fault, etc.), utilized security solutions will not be effective. In addition, depending on the network architecture, they may not be able to monitor all network traffic (e.g., *ad hoc* IoT network [16]), especially if a device directly sends a message to another device which is not supposed to do. Distributed hierarchical approaches [17], [18] enable monitoring all network traffic by distributing agents across the network. However, they are not free from faults (compromised agents) because if any agent is compromised, its assigned part of the system cannot be monitored. It can be addressed by redundancy, but the number of redundant agents is usually limited [19]. In other words, there is a nonnegligible possibility that all redundant agents can be

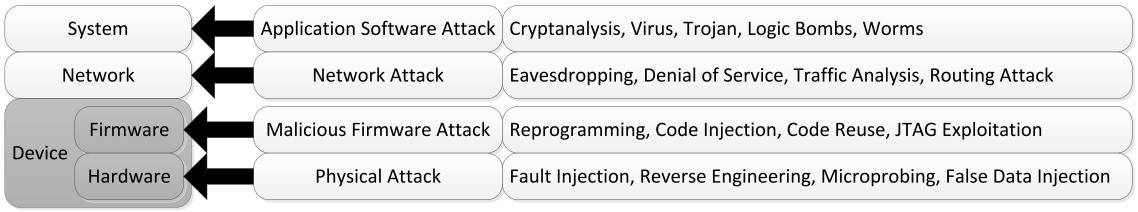


Fig. 1. Classification of attack scenarios in networked embedded systems, where selected representative examples are shown. The primary target of the proposed framework is handling a device compromised by malicious firmware or physical attacks.

TABLE I
COMPARISON OF APPROACHES FOR IOT SECURITY

Approach	Fault tolerance	Runtime update	Monitoring all traffic
Centralized [5]–[8]		◯	
Hierarchical [17], [18]		△	△
Peer-to-peer [9]	◯		◯
Blockchain	◯	◯	◯

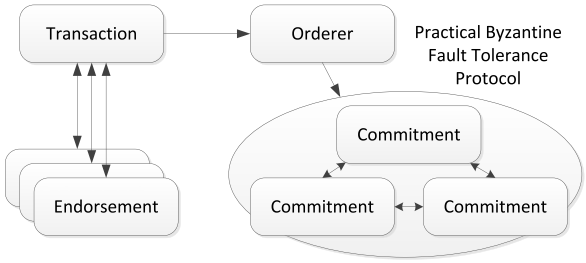


Fig. 2. Hyperledger Fabric blockchain framework.

compromised at the same time. The peer-to-peer approach [9] offers fault tolerance by employing a voting mechanism. Since the decision is made collaboratively, it can guarantee security policy enforcement as long as the majority of devices are not compromised. However, it does not employ a consensus protocol, which limits its capability. If any information needs to be shared by devices and the information needs to be updated runtime, the existing peer-to-peer approach cannot support it.

The proposed approach enforces security policies using *blockchain*. This allows us to enable trusted computing on untrusted devices. As long as a majority of the devices in the network are not compromised, the security of the entire system can be guaranteed. Therefore, compared to existing centralized and distributed approaches, this blockchain-based approach offers a higher level of assurance. By using the consensus protocol, the blockchain-based approach can support the runtime update of shared information. Furthermore, the proposed approach monitors all network traffic and automatically isolates a malicious device by blocking all traffic from it. Table I summarizes it.

The technical contribution of this article is the solution to the three challenges posed in the previous section, which should be addressed when blockchain is applied to the IoT setup. We address them by employing Hyperledger Fabric and add-on hardware modules.

Hyperledger Fabric [20] is designed to work with *permissioned* private networks, where membership must be controlled. Such an environment is typical of IoT systems. One advantage of Hyperledger Fabric over the permissionless blockchain is the order-of-magnitude lower latency since the computationally expensive Proof of Work (PoW) is not required. This allows us to address the latency limitation (challenge 1).

In order to enable COTS devices that would not otherwise support the proposed framework to be able to do so, an add-on hardware module is introduced (challenge 2). By pairing with an add-on module, an existing device can join the framework without any modification to the actual device. The blockchain client is implemented on the add-on module; hence, allowing us to mitigate the limitation of resource-constrained devices (challenge 3). Employing Hyperledger Fabric also helps reduce the resource demand because it uses a PBFT protocol as a consensus protocol instead of PoW.

The existing blockchain-based IoT security solutions [21]–[28] do not address these challenges. Thus, they have been used for a secure firmware update, configuration management, and energy trading that do not require to address the challenges. To the best of our knowledge, IoTCop is the first attempt to offer a *general monitoring framework* that can be used to mitigate various attacks on devices.

Since the add-on module is an additional component in the system, it may increase the cost of the system. However, employing an alternative security solution is also not free. To employ existing centralized or distributed solutions, we need to employ additional devices or increase the capacity of existing devices. Since the add-on module can be used for any kind of device, the cost of one module can be amortized by mass production.

B. Hyperledger Fabric

The Fabric blockchain framework is one of the projects in Hyperledger. Hyperledger includes various development projects of frameworks and tools for business and industrial blockchain applications. The Fabric project aims at providing a permissioned blockchain framework for private or consortium networks. Since the membership of participants is controlled, it does not have to assume the harsh environment that permissionless blockchain frameworks (e.g., Bitcoin and Ethereum) operate in. Fabric employs the PBFT protocol as its consensus protocol, which is less computationally demanding than PoW.

Fig. 2 shows the workflow of the Hyperledger Fabric framework. When a new transaction is issued, it is broadcasted to endorsing peers. Each endorsing peer executes a smart contract that determines whether the transaction should be accepted

based on its endorsement policy. If the transaction is endorsed by endorsing peers, then it is sent to the orderer. The orderer collects transactions and makes a block. Although the orderer is shown as a single entity in the figure, it is implemented in a distributed manner for fault tolerance. The presence of the orderer prevents wastage of computing power to mine a block that will not be used. Finally, the block is stored in committing peers by using the PBFT protocol.

The distributed nature of the Fabric provides resilience to faults (compromised peers). The consistency of the shared information is guaranteed by the consensus protocol. The use of the PBFT protocol (instead of PoW) offers low latency and low-performance overhead. For these reasons, we select the Hyperledger Fabric as the underlying blockchain framework to ensure the security of IoT systems. However, we implemented the framework from the scratch instead of using their implementation, as it includes many features that are not essential for our purpose.

III. Related Works

This article is the first attempt to provide a *blockchain-based monitoring framework* that detects and isolates compromised devices under malicious firmware attacks or physical attacks, at the time of this research. In prior IoT security frameworks, to protect devices from firmware or physical attacks, researchers generally focused on *design methodologies* that ensure security properties at *design time* with an emphasis on hardware–software code sign [11], a holistic framework encompassing process, people, technology, and organization [29], [30], combining the security requirements of an IoT ecosystem [31], and so on.

The threat we considered in this article is a *malicious firmware attack*, where an adversary has the capability to modify the firmware or add malicious code to the firmware so that they can execute arbitrary code of their choice. The firmware can be compromised statically or dynamically. A static firmware attack compromises the firmware by replacing the code in flash memory or working memory (e.g., DRAM). It can be prevented by secure boot [14], [15] and attestation of firmware [32]–[34]. A dynamic firmware attack exploits the dynamic memory regions (stack and heap) to circumvent secure boot and attestation. Code injection [35] and code reuse [36] attacks are representative examples. The other threat considered is a *physical attack* targeting the hardware, which includes fault injection, reverse engineering, microprobing, and false data injection [37].

Despite efforts spent on designing firmware and hardware compromise mitigation solutions, there are still possibilities of devices being compromised. Existing security solutions for general-purpose computers are not suited for deployment on systems involving the resource-constrained nature and lack of physical security in IoT devices. Our approach is also compatible with existing prevention techniques but assumes that a device can be compromised (rather than the typical "honest-but-curious" adversary model).

If a device is compromised, then it can be detected by checking for violation of the security policies. The proposed framework enforces security policies using *blockchain*, and the blockchain-based framework offers a higher level of assurance than existing centralized or distributed approaches [5]–[8], [38], which generally does not assume that the security solutions themselves can be compromised.

There are IoT security solutions that employ blockchain. These techniques ensure a certain security property of the IoT system, such as firmware update [21], [24], configuration management [22]–[25], traceable data management [39]–[41], trust model [42], and authentication [26], [43]. Blockchain has also been used to facilitate energy trading in a smart grid [44]–[46]. More recently, in 2020, Liu *et al.* [47] presented a new blockchain framework for the IoT environment. However, the propagation delay in the proposed framework is longer than 10 s for 30 nodes. While it is significantly shorter than the delay of permissionless blockchain frameworks (which is between 1 and 10 min [10]), it is still longer than what IoTCop offers (which is up to 730.11 ms for 30 nodes).

IV. IoTCop Overview

This section provides an overview of the proposed framework, IoTCop. IoTCop is a blockchain-based monitoring framework that monitors network messages of devices and isolates (malicious) devices that do not comply with security policies.

A. Threat Model

The primary goal of IoTCop is to isolate a device whose *network message* violates security policies. The behavior of a device may change after it has been infected by malicious firmware or physical attacks. The changed behavior may not be observable, for example, the device may eavesdrop on the network traffic, without sending any unauthorized or malicious messages to other devices. Hence, this is not the focus of the proposed framework. If the compromised device attempts to send information to an unauthorized device, however, it will be blocked. IoTCop seeks to prevent a malicious device from *seeking to affect the system* by sending unauthorized or malicious messages.

IoTCop is tasked with *enforcing* the security policies, which may be application dependent and must be provided by the system administrators. The quality/effectiveness of security policies (e.g., whether they are sufficiently effective to detect all malicious behaviors) is beyond the scope of this article.

Denial-of-Service (DoS) attack is not considered by the current version of IoTCop. However, though the DoS attack may hinder normal operations of the framework and the system, it cannot break the security properties guaranteed by IoTCop. In other words, malicious devices are still isolated even under the DoS attack, although benign devices may not be able to perform normal operations due to the DoS attack.

B. Location

IoTCop works between the application layer and the network layer, as shown in Fig. 3. The network layer is required to facilitate communication. The framework monitors the messages of the application layer to check whether devices

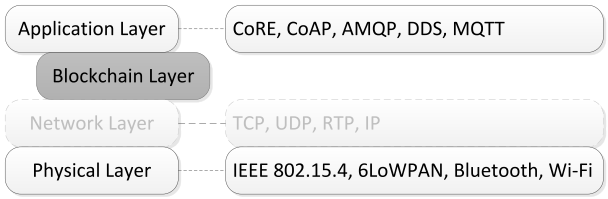


Fig. 3. IoTCop is located between application and network layers. In the absence of the network layer, it can work on top of the physical layer.

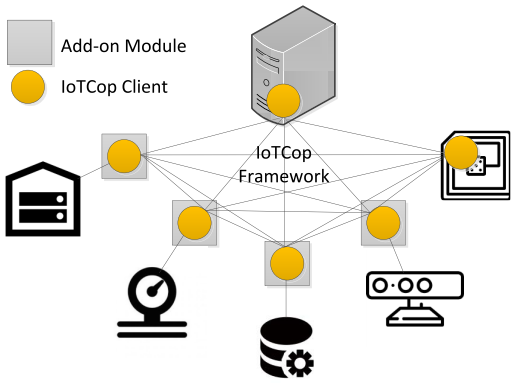


Fig. 4. IoTCop is implemented by IoTCop clients. The client may be placed on the device if the device supports IoTCop. Even if a device does not support it, the device can join the framework by pairing with an add-on hardware module.

comply with the security policies. The framework is also independent of the network and application layers. In other words, the framework can work on top of any network protocols and can monitor any application layer protocols. If the network layer is not present, then it can also work on top of the physical layer.

IoTCop is implemented using blockchain clients, as shown in Fig. 4. A system may consist of different types of devices, which may not directly support the framework. If the device supports the framework, then it means the blockchain client runs on the device. Devices that do not support the framework can join the framework by pairing with an add-on hardware module, where a blockchain client runs.

Pairing with an add-on module does not necessarily mean physical attachment. An add-on module may be physically attached to a device, but it can also be connected to a device remotely over a network. An add-on module has two network interfaces: one is for its paired device and the other is for the blockchain network. The first interface for a paired device should be configured so as to be connected only to its paired device. The network interface may vary with devices. To be applicable to a broader range of devices, the add-on module should be equipped with various network interfaces for its paired device.

C. Protocol

The protocol of IoTCop is illustrated in Fig. 5. All messages of the application layer should go through the framework so that compliance with security policies can be verified. When a message is sent from a source device, it is handled by the sender interface in its IoTCop client. The client may be located

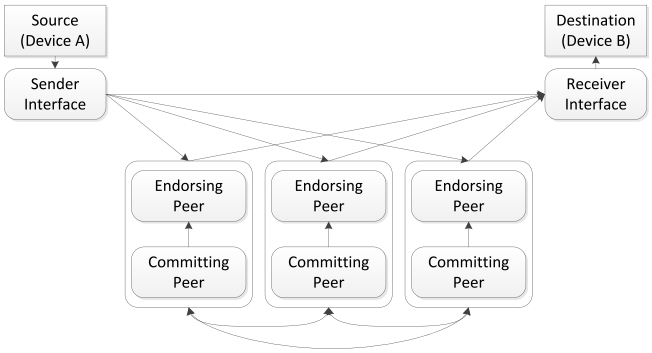


Fig. 5. IoTCop client consists of sender and receiver interfaces, endorsing peer, and committing peer. This figure shows how they interact.

either at the device itself or the paired add-on module. The sender interface sends the message to all endorsing peers. Each endorsing peer determines whether or not the message is compliant based on its endorsement policy. The result is sent to the receiver interface of the destination device. If the message is accepted, then it is delivered to the destination device. Depending on the result, the security policies may need to be updated. The security policies are updated by committing peers through the consensus protocol. The latter ensures the consistency of the shared security policies. The technical details of the protocol are presented in Section V.

D. Allocation

A blockchain client may not have all *logical* components of the blockchain clients (see rounded boxes in Fig. 5) due to resource availability. IoTCop offers flexible allocation of components, but a pair of sender and receiver interfaces should be allocated together, and a pair of endorsing and committing peers should also be allocated together.

In Fig. 6, for example, device A is connected to the framework through an add-on module. The blockchain client running on the add-on module for device A includes all of the sender/receiver interfaces and endorsing/committing peers. The client for device B also has the same components, but the client runs in device B itself. For device C, due to its resource constraints, it runs only the sender/receiver interfaces.

It is also possible to utilize a device external to the system, which does not send or receive any messages to/from devices of the target system. For example, desktops and servers can be utilized as endorsing and committing peers. Since those devices do not exchange messages, the sender/receiver interfaces are not necessary. If such devices are employed, then an adversary will find it harder to locate and compromise such devices. They also mitigate resource requirements for running the blockchain clients.

V. IoTCop Framework

This section presents the technical details of the proposed framework. The primitives of building IoTCop and the protocol definition are given in Sections V-A and V-B, respectively. The security properties are discussed in Section V-C followed by limitations in Section V-D.

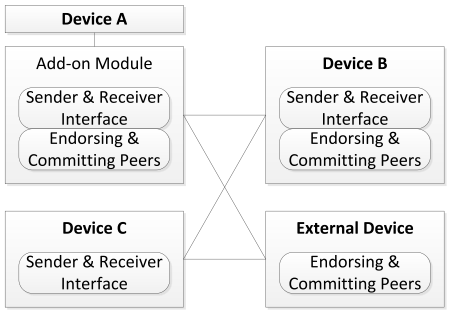


Fig. 6. Example allocation of blockchain clients.

A. Primitives

IoTCop employs a digital signature scheme to validate the authenticity of messages. It consists of the following three algorithms.

1) GEN*(1^λ):* Given as input the security parameter 1^λ, it generates a pair of public and private keys (PK, SK).
2) SIG*(SK, m):* Given as input a private key SK and a message m, it generates a signature σ.
3) VER*(PK, m, σ):* Given as input a public key PK, a message m, and a signature σ, it generates 1 if the signature σ on the message m is successfully verified with the public key PK, and 0 otherwise.

When an IoTCop client joins the framework, it generates (PK, SK) by using GEN and publishes PK as its ID.

An IoTCop client is comprised of a sender interface, receiver interface, endorsing peer, and committing peer. Their algorithms are as follows.

1) SENDERINTERFACE*(m):* For the given input m, it generates $M = (m, \text{PK}_{\text{sender}}, q, \sigma_{\text{sender}})$. $\text{PK}_{\text{sender}}$ is the public key of the IoTCop client, where the sender interface runs. q is a sequence number that increments whenever this algorithm is executed. $\text{PK}_{\text{sender}}$ and q are included in M to uniquely identify the message globally (across the system) and prevent the replay attack. If q is reused by an adversary, the previous message can be sent again. This message can be used for false command/data injection. Thus, it is important to assign a unique q whenever a message is generated. σ_{sender} is $\text{SIG}(\text{SK}_{\text{sender}}, m||q)$.
2) RECEIVERINTERFACE*($\mathbb{R}$, M):* $\mathbb{R}$ is a set of response messages R from all endorsing peers for the same sender and message, where $R = (r, \text{PK}_{\text{sender}}, q, \sigma_{\text{sender}}, \sigma_{\text{end}})$. All R in $\mathbb{R}$ have the same $\text{PK}_{\text{sender}}$, q, and σ_{sender}. M is received from the sender. RECEIVERINTERFACE generates 1 if $R.r$ is 1 (which means accept), $\text{VER}(\text{PK}_{\text{peer}}, R.r||R.\text{PK}_{\text{sender}}||R.q||R.\sigma_{\text{sender}}, R.\sigma_{\text{end}})$ is 1, $R.\text{PK}_{\text{sender}}$ is equal to $M.\text{PK}_{\text{sender}}$, and $R.q$ is equal to $M.q$, for the majority ($> n/2$) of R in $\mathbb{R}$. As mentioned above, the public keys of IoTCop clients are published, which means PK_{peer} is known. The signature of M should also be validated [$\text{VER}(M.\text{PK}_{\text{sender}}, M.m||M.q, M.\sigma_{\text{sender}})$ should be 1]. Otherwise, it generates 0.
3) ENDORSINGPEER*(M):* M is the message generated from the sender interface of the source device. Only if $\text{VER}(M.\text{PK}_{\text{sender}}, M.m||M.q, \sigma_{\text{sender}})$ is 1, it generates R; otherwise, M is ignored. $R = (r, M.\text{PK}_{\text{sender}}, M.q, M.\sigma_{\text{sender}}, \sigma_{\text{end}})$. $R.r = \text{CHECKPOLICY}(M.m, M.\text{PK}_{\text{sender}}, M.q, T)$. CHECKPOLICY is a user-defined function that generates 1 if $M.m$ is accepted, and 0, otherwise. T is the state shared by all peers. Its definition should be given by the user. $R.\sigma_{\text{end}} = \text{SIG}(\text{SK}_{\text{peer}}, R.r||M.\text{PK}_{\text{sender}}||M.q||M.\sigma_{\text{sender}})$. SK_{peer} is the private key of the IoTCop client where the peer runs. If the shared state needs to be updated, it sends U to all peers. $U = (u, M.\text{PK}_{\text{sender}}, M.q, M.\sigma_{\text{sender}}, \sigma_{\text{com}})$. u is a user-defined message to update the shared state. $\sigma_{\text{com}} = \text{SIG}(\text{SK}_{\text{peer}}, u||M.\text{PK}_{\text{sender}}||M.q||M.\sigma_{\text{sender}})$.
4) COMMITTINGPEER*($\mathbb{U}$):* $\mathbb{U}$ is a set of update messages U from all endorsing peers triggered by the same sender and message, where $U = (u, \text{PK}_{\text{sender}}, q, \sigma_{\text{sender}}, \sigma_{\text{com}})$. All u in $\mathbb{U}$ have the same $\text{PK}_{\text{sender}}$, q, and σ_{sender}. It updates the shared state T, if u of U are the same, and $\text{VER}(\text{PK}_{\text{com}}, U.u||U.\text{PK}_{\text{sender}}||U.q||U.\sigma_{\text{sender}}, U.\sigma_{\text{com}})$ is 1, for the majority ($> n/2$) of U in $\mathbb{U}$. PK_{com} is the public key of the peer who sent U. It ignores U, otherwise. It updates the state by calling the user-defined function $\text{UPDATESTATE}(U.u, T)$.

The user provides the definition of the following two functions and two data structures. They may be implemented by using a smart contract, but its latency is order-of-magnitude longer than that of IoTCop. We will evaluate the latency using experiments in Section VII.

1) CHECKPOLICY*(m, PK_{sender}, q, T):* It generates 1 if m is accepted and 0, otherwise. It may check multiple security policies.
2) UPDATESTATE*(u, T):* It updates T processing the request u.
3) *T:* The user defines the data structure of the shared state.
4) *u:* The user defines how to update T.

B. IoTCop Protocol

The sequence diagram of the IoTCop protocol is depicted in Fig. 7. The protocol begins when a source device sends an application layer message, $A = (m)$. The IoTCop protocol is transparent to the application layer. Thus, the source device sends A without being aware of the IoTCop framework.

For A to be delivered to the destination device, its compliance is checked by IoTCop. To do so, the sender interface of the IoTCop client broadcasts M to all other IoTCop clients. When an IoTCop client receives M, it individually determines whether or not A should be accepted. When it is determined, R is sent to the receiver interface of the destination device. The receiver interface sends A to the destination device if RECEIVERINTERFACE returns 1. Otherwise, A is discarded. In this way, the source device is isolated if its message is against the security policy.

An IoTCop client may need to update the state depending on the results of checking security policies. In this case, an IoTCop client broadcasts U to other clients to update the state.

In the PBFT protocol, there is a commander who initiates a message to all lieutenants. Each lieutenant forward the

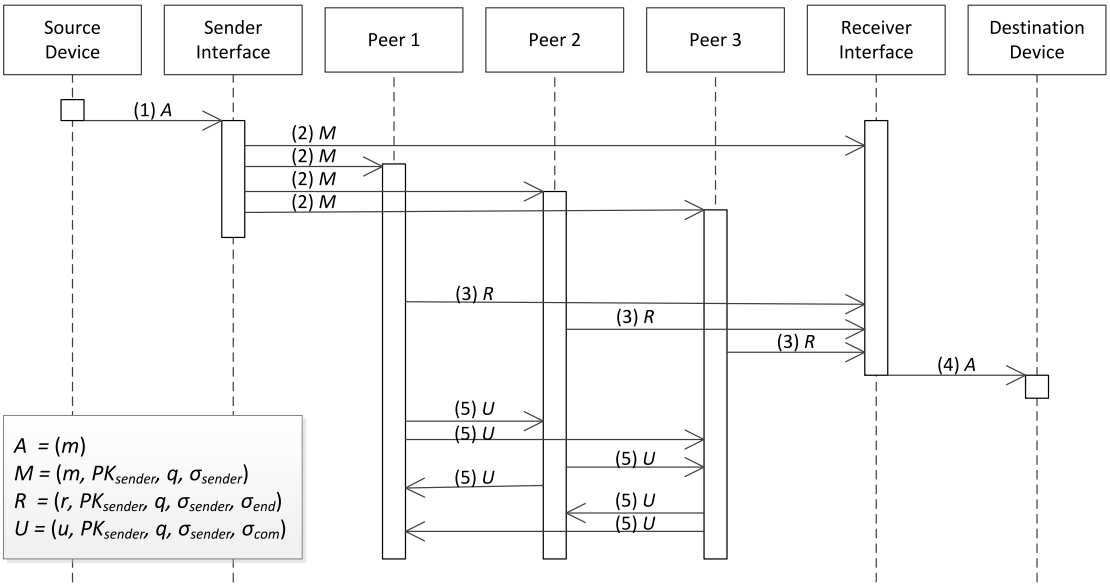


Fig. 7. Sequence diagram of the IoTCop protocol.

received message to others and receives forwarded messages from others. If the majority of the received messages are the same, a consensus is reached and the message is accepted. In Hyperledger Fabric, an orderer plays as a commander, and committing peers do as lieutenants. The orderer creates block collecting transactions and initiates the block to all committing peers. Each committing peer performs the same tasks that the lieutenant does in the PBFT protocol to reach a consensus.

In IoTCop, the sender interface plays as a commander, and other IoTCop clients do as lieutenants. The sender interface triggers the protocol by sending M to all IoTCop clients. Each IoTCop client generates U that is analogous to the forwarded message. The difference is that U is generated only if necessary. U is exchanged among IoTCop clients to reach a consensus.

C. Security Properties

Let us suppose a probabilistic polynomial-time adversary $\mathcal{A}$ and $\mathcal{A}$ succeeds in compromising an IoT device in a system. The goal of $\mathcal{A}$ is to compromise or trigger the malfunction of another device in the system by sending illegitimate messages. $\mathcal{A}$ wins the game if $\mathcal{A}$ can send illegitimate messages that are accepted by healthy devices, whereas the challenger $\mathcal{C}$ wins if the device, which is already compromised by $\mathcal{A}$, is isolated, which means illegitimate messages sent by the compromised device will not be accepted by healthy devices.

$\mathcal{A}$ can try to win the game by compromising endorsing peers, committing peers, the sender interface, or the receiver interface.

Theorem 1: The security policies are enforced as long as the compromised endorsing peers are no more than $n/2$, where n is the total number of endorsing peers.

Proof: $\mathcal{A}$ may try to compromise endorsing peers and convince them to accept illegitimate messages, which is against the policy. To succeed in this attack, $\mathcal{A}$ needs to compromise the majority of endorsing peers because the illegitimate messages are accepted only if the majority of endorsing peers approve. In other words, illegitimate messages are not accepted, if the number of compromised endorsing peers is less than $n/2$. ■

Theorem 2: The shared state is maintained correctly as long as the compromised committing peers are no more than $(n-1)/3$, where n is the total number of committing peers.

Proof: $\mathcal{A}$ may try to compromise committing peers to distort the decision by making illegal changes to the shared state. Since Hyperledger Fabric employs PBFT, this type of attack cannot succeed as long as the number of compromised committing peers is less than $(n-1)/3$. ■

Theorem 3: Compromising the sender interface only results in isolation of the source device connected to the sender interface.

Proof: The compromised sender interface may alter $M = (m, \mathrm{PK}_{\mathrm{sender}}, q, \sigma_{\mathrm{sender}})$. If the sender interface replaces m with an illegitimate one that does not comply with security policies, the illegitimate message is discarded. As long as the sequence number (q) is unique in a predetermined time window (defined by timeout), any number can be used. If the sender interface uses a number overlapping with previous messages, it only causes both messages to be discarded. If the signature does not match with m, $\mathrm{PK}_{\mathrm{sender}}$, or q, M is ignored by endorsing peers. If the sender interface sends M to a wrong destination device, it is discarded if it does not comply with security policies. The sender interface may send different messages to different IoTCop clients with the same sequence number, which only results in discarding the messages. Therefore, even if the sender interface is compromised, the security properties are not affected. ■

If the receiver interface is compromised, illegitimate messages may be accepted by the destination device, which is connected to the compromised receiver interface. This is because it is the receiver interface of the destination device that

makes the final decision of whether a message will be accepted or not. It is a gray area where $\mathcal{A}$ wins, but $\mathcal{C}$ also wins in a sense. $\mathcal{A}$ wins because illegitimate messages are accepted by another device (the destination device whose receiver interface is compromised). However, the impact is confined to only the destination device. It is only the destination device that accepts unauthorized or malicious messages when the receiver interface is compromised. In other words, even if this happens, the source device is still isolated because illegitimate messages are rejected by all other devices, which means $\mathcal{C}$ also wins.

While the shared state is maintained by the blockchain, no messages or user data are stored in the shared state. Specifically, the content stored in the blockchain is not dictated by IoTCop, rather it depends on the policy defined by the user. In our context, only the whitelist is stored in the blockchain.

D. Limitations and Future Works

It is assumed that device authentication and identification are achieved securely by utilizing existing techniques. In other words, the limitation of the current framework is identity theft. If a device manages to steal identity of another device and impersonates this device, the proposed framework may allow the device to perform tasks it had not been authorized to (prior to being stolen). Since the focus of this article is on enforcing security policies, this feature is beyond our scope. However, the proposed framework is compatible with existing device identification techniques [48], [49]. By employing existing techniques, the proposed framework can address identity theft.

If a device sends a message to a destination device that does not belong to the target system, it is out of scope because the goal of the proposed framework is to prevent a malicious device from *affecting the entire system* by sending unauthorized or malicious messages. In other words, if the destination device is unknown, security properties are not enforced. For some applications, it may raise a concern that the proposed framework may not be able to block information leakage to unknown devices. This is because it is the destination device that finally rejects illegitimate messages, in our current implementation. If we allow other network components (such as access points, switches, and routers) to reject illegitimate messages, we can prevent information leakage to unknown devices. To do so, it requires modification to network components.

The current implementation of the framework does not assume messages to be encrypted. Even if messages are encrypted, endorsing peers may obtain all necessary information. For example, for whitelisting, the only required information is the ID of the source device. In contrast, to check the communication policy, the message type is also required. The IoTCop client can work only if the message type is not encrypted. If it is encrypted, then the proposed framework should be extended in the following way. When the source and destination devices exchange a symmetric key for encryption, their IoTCop clients should also have the key. When an encrypted message is sent, the IoTCop client of the source device decrypts the message and extracts the required information for checking security policies. The required information is encrypted with a different key and broadcasted to other IoTCop clients. To do so, all IoTCop clients need to maintain the same key for encryption, but the key for source and destination devices can be kept only in the source and destination IoTCop clients.

The current implementation of the framework does not support strong memory consistency for the shared state. If multiple committing peers initiate conflicting update requests (triggered by different endorsing peers) at the same time, the shared state may become inconsistent. What is guaranteed is that the update information U is processed atomically, and the order from the same committing peer is always maintained. This suffices to implement the security policies of the case study. However, if the order of U from *different* committing peers matters, a stronger consistency model is necessary, which is left as our future work.

If a majority of the devices are of the same type or connected to the framework through the same type of add-on hardware module, then the security of the proposed framework could be weakened. In this situation, if there is a vulnerability that can be exploited remotely, a majority of devices (or add-on hardware modules) can be easily compromised. Therefore, in a real-world environment, it is desirable to run IoTCop clients on a variety of devices. There should also be multiple types of add-on hardware modules so that any device type does not dominate the system.

In IoTCop, the delay increases with the number of peers. Thus, if the number of devices grows, scalability may become an issue. To ensure scalability, we can employ multithreading, hierarchical topology, and early termination. The delay increases linearly because the receiver interface or the committing peer needs to check the signature from all endorsing peers (in case of the receiver interface) and from other committing peers (in case of the committing peer). Instead of checking the signature one by one serially, if we can parallelize it by multithreading; thus, improving scalability. The decision may also be made hierarchically. For example, we can cluster devices into groups, and let the group leader make an intermediate decision, and the receiver interface and committing peer make the final decision from the intermediate decisions. Finally, we may terminate receiving messages once the consensus is reached. Therefore, once the consensus is reached [i.e., if the majority of the response (R) or the update request (U) are the same], the remaining R and U can be ignored.

A device may be isolated not because it is compromised, but because transient errors occurred. To handle this situation, we may extend IoTCop to support an automated recovery mechanism. If the device proves itself to be benign, the IoTCop framework allows it to rejoin the framework. We can use remote attestation, self-healing, or simple rebooting (in case of transient attacks) as a method of proof.

VI. Case Study: Building Automation and Control

For validation, we deploy the proposed framework using the BACnet, a widely used protocol that works on existing IP

TABLE II
EXAMPLES OF BACNET OBJECTS

BACnet Objects	Source	Destination	Uses
Analog Input	Device	Controller	Smart sensor input, inputs of a device (e.g. room temperature, humidity)
Analog Output	Controller	Device	Smart sensor output, physical outputs of a device (e.g. valve, humidifier)
Analog Value	Controller	Device	Used for a setpoint, internal values used in a device's program (e.g. variable, constant)
Binary Input	Device	Controller	Used as a switch, inputs of a device (e.g. a bulb status, occupancy)
Binary Output	Controller	Device	Used as a relay, physical outputs of a device (e.g. bulb's on/off switch)
Binary Value	Controller	Device	Used as a binary control system constraint, internal values used in a device (e.g. variable, constant)
Program	Controller	Device	Allows a program running in a device to be started, stopped, loaded and unloaded, and reports the present status of the program

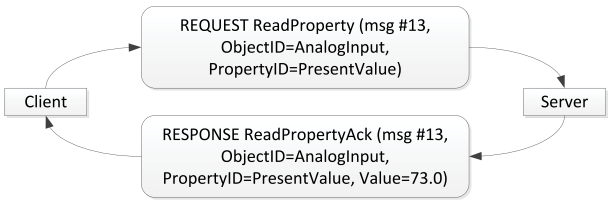


Fig. 8. Example of a BACnet explicit service for reading room temperature.

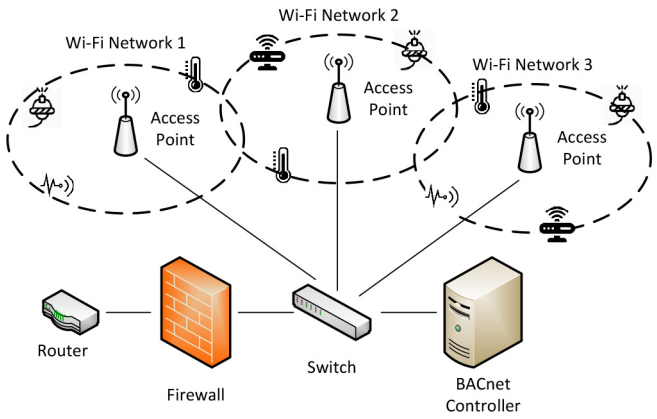


Fig. 9. Network topology assumed for the case study. All devices are connected over Wi-Fi networks and the access points are connected to a network switch. The central controller is connected to the switch, and the system is connected to the public Internet through a firewall and a router.

network infrastructure [50]. The incorporation of IP networks makes it easy and efficient to deploy, but also creates a large attack surface [50]. However, most (if not all) of deployed BACnet systems do not have security features because of their high cost, and there is no known security module for BACnet [51]. Therefore, we apply the proposed framework to demonstrate how to protect a BACnet system.

A. BACnet Background

BACnet is an open data communication protocol to build automation control networks. It provides methods by which computer-based control apparatus from different manufacturers can work together. It allows users to inflate, blend, and match equipment to better fit different needs and design specifications in buildings. BACnet uses an "object model" to represent the functioning of building automation and control systems [52].

BACnet follows a client–server model, where a BACnet client is a device that requests a service and a BACnet server is a device that performs a service. It allows devices to interact with each other using a common communication network for sharing functionality and responsibility of different control functions. Table II summarizes the object examples.

BACnet objects carry different properties. An object is exemplified by a specific name. For instance, the analog input object is exemplified with the name "ReadProperty." ReadProperty service is used by a BACnet device (client) to request another BACnet device (server) to provide the value of properties of an object. The server then responds with the appropriate information for the requested object, indicating the successful completion of an operation. Fig. 8 is an example of the explicit service (request and response) associated with the room temperature.

BACnet has many applications adopted for various network topologies. It is hard to generalize them because they are heavily dependent on applications. For the purpose of latency evaluation, the network topology shown in Fig. 9 is assumed in our case study. The system consists of multiple Wi-Fi networks, and all devices are connected over the Wi-Fi networks. We assume there is a central BACnet controller that communicates with all devices and this central controller is directly connected to the switch. The system is connected to the public Internet through a firewall and a router.

B. Security Policies

In this case study, we implemented two security policies: 1) communication policy and 2) whitelisting. The type of endorsement policies (security policies) depends on the systems and applications. In this article, two types of endorsement are implemented as a proof of concept. They are chosen because they can be applied to any system. Since endorsing and committing peers are highly modular, they can easily be reused for other applications. It should be noted that IoTCop is not limited to these two examples and it can accommodate various types of security policies provided by the system designer.

The communication policy is based on source, destination, and object. Only when the source, destination, and object of a message match, the message is approved. Fig. 10 shows an example. Let us suppose that there are three devices (`D1`, `D2`, and `D3`) and one central controller (`C1`). At a certain point, `D1` sends `AnalogInput` to `C1`. Since the tuple of (`D1, C1, AnalogInput`) is found in the communication policy, it is

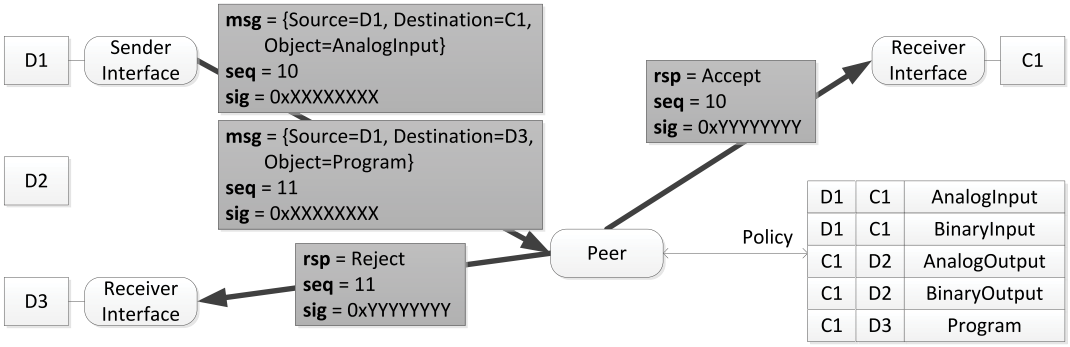


Fig. 10. Example of the communication policy. The first message `AnalogInput` from `D1` to `C1` is accepted but the second message `Program` from `D1` to `D3` is rejected because the second message cannot be found in the policy.

approved. If `D1` tries to send `Program` to another device (`D3`), it is rejected because (`D1, D3, Program`) is not found in the policy. Therefore, any (unauthorized) message from compromised devices or unknown devices will be rejected and compromised or unknown devices are isolated from the rest of the system.

The policy is given at the installation time by an administrator and does not change dynamically during runtime. If necessary (e.g., if a new device is installed), the administrator may change it manually by triggering the update protocol (by using U). The same policy is given to all IoTCop clients.

Recall that security policies CHECKPOLICY, the state of IoTCop clients (T), and the update information UPDATESTATE and u are application specific and given by the user. In this case study, the communication policy is implemented in function CHECKPOLICY, which looks up the policy table and returns true if the message is found in the table. The table is a part of T of IoTCop clients. Since the table is not updated during runtime, u is not used for this policy.

Since the focus of this article is on validation of the framework (how to *enforce* security policies), the above-mentioned simple communication policy is employed. However, an advanced policy is also applicable to the framework. An example is the Bark language [53]. It allows the administrator to specify a complex communication policy, which includes subject, object, action, and conditions. For example, the Bark language can specify "device A is allowed to request data from device B between 8 A.M. and 5 P.M. on weekdays." In the original paper [53], the policy is enforced by a gateway, which is a centralized approach. If it is employed as an endorsement policy of the proposed framework, then the policy is enforced by multiple endorsing peers. Thus, it offers a higher level of assurance than the centralized approach.

The second policy is whitelisting. When a system is installed, a list of trusted devices (whitelist) is given by an administrator. The whitelist is maintained by IoTCop clients, and the initial whitelist is given to all clients. If the source device of the message is found in the list, it is approved. If a message is denied by the other security policies (communication policy in this case study), its source device is removed from the whitelist so that the source device can be isolated permanently. Each IoTCop client updates the whitelist if a consensus is reached by PBFT.

TABLE III
HARDWARE SPECIFICATION OF DEVICES USED FOR PROTOTYPING

Component	Value
Processor	Cortex A53
Number of cores	4
Processor clock	1.2 GHz
Main memory	1 GB
Operating system	Raspbian

The whitelisting is also implemented in function CHECKPOLICY, which approves if the source device of the message is found in the list. The list is a part of T, and u is used when a source device should be removed from the list.

In this case study, we implemented that a suspicious device is removed from the whitelist immediately. It should be noted that it is just an example that is chosen to demonstrate how IoTCop works. The use has the freedom to choose any kinds of security policies and their implementation. As a variant of our implementation, a suspicious device may be removed if it violates security policies more than once (e.g., three times). It is also possible for the administrator to confirm the removal after manual investigation.

VII. EVALUATION

The evaluation of the IoTCop framework is twofold. We first compare the latency of the IoTCop framework with that of Ethereum using a small-scale prototype. The results are given in Section VII-A. To extend the latency evaluation to large-scale systems, we developed an analytical model by characterizing the prototype. The methodology and results are presented in Sections VII-B and VII-C. Finally, use case scenarios are discussed to evaluate the security level of IoTCop compared with existing approaches in Section VII-D.

A. Prototype

The prototype is developed using four Raspberry Pi 3 boards. Their hardware specification is given in Table III. All boards are connected to the same Wi-Fi network.

The latency is significantly affected by the digital signature algorithm (DSA) used because the generation and verification of a signature should be performed multiple times to

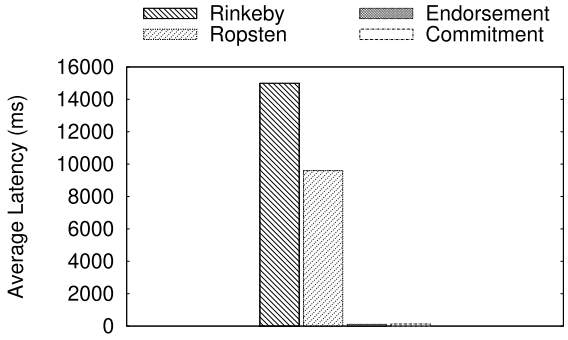


Fig. 11. Average latency of IoTCop and Ethereum on the prototype. The average latency of endorsement and commitment of IoTCop is order-of-magnitude lower than that of Rinkeby and Ropsten Ethereum test networks.

TABLE IV
RESOURCE OVERHEAD INCURRED BY AN IOTCOP CLIENT

Component	CPU load	Memory usage
Sender interface	0.2 %	0.8%
Receiver interface	0.2 %	0.8%
Endorsing peer	0.3 %	1.8%
Committing peer	0.3 %	1.8%

process a message. We employ the popular elliptic curve DSA (ECDSA) with the National Institute of Standards and Technology (NIST) P192 curve as a DSA in this evaluation. However, DSA is orthogonal to our framework. The IoTCop framework can work with any DSA that may offer higher performance and/or security than ECDSA.

We implemented both IoTCop clients and Ethereum clients on the same platform to make a comparison. As a reference, we implemented the same whitelisting technique on Ethereum clients. We measured the average end-to-end latency of message delivery as endorsement latency. As commitment latency, the average of latency between when a message is sent from the sender interface and when all committing peers reach a consensus. In the case of Ethereum, the latency was measured two test networks: 1) Rinkedy and 2) Ropsten.

Fig. 11 shows the results. On average, it takes 110.81 and 128.95 ms for endorsement and commitment, respectively, in IoTCop. In contrast, it takes 15.0 and 9.6 s for the whitelisting in Rinkeby and Ropsten, respectively. It is clear that IoTCop offers order-of-magnitude lower latency compared to Ethereum. Furthermore, the resource demand of IoTCop is much less than Ethereum because IoTCop employs PBFT as a consensus protocol instead of a computationally intensive PoW. The resource demand for IoTCop is quantitatively measured and summarized in Table IV.

B. Modeling Methodology

To evaluate IoTCop in large-scale systems, we developed an analytical model to estimate the latency. The delay components are illustrated in Fig. 12 and their definitions presented in Table V. Since the computational intensity of the two types of endorsement (communication policy and whitelist) is very low, their computational delay is not measured separately.

The delay of endorsing a message D_{end} is calculated using the following equation, where n denotes the total number of

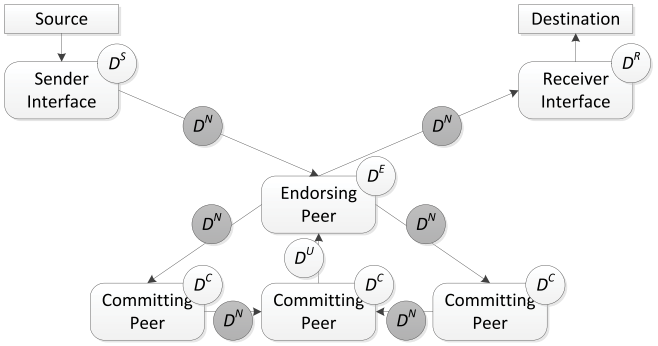


Fig. 12. Delay components of the IoTCop framework.

TABLE V
DEFINITION OF DELAY COMPONENTS

Delay	Definition	Measure
D^S	Delay of a sender interface that includes generation of a digital signature.	21.55 ms
D^E	Delay of an endorsing peer that includes looking up the endorsement policy and generation of its own digital signature.	24.21 ms
D^R	Delay of a receiver interface per endorsing peer that includes verification of the digital signature of the endorsing peer and the sender.	19.89 ms
D^C	Delay of a committing peer that includes generation of its own digital signature to forward the update request to other committing peers.	19.95 ms
D^U	Delay of the policy update by a committing peer that includes verification of the digital signature of the committing peer and the sender.	21.58 ms
D_W^N	Network delay within a Wi-Fi network.	8.49 ms
D_S^N	Switch delay.	0.52 ms

IoTCop clients in the system:

$$D_{\text{end}} = D^S + D^E + n \times D^R + D^N. \tag{1}$$

The delay of committing a request from when a request is sent from the sender interface is as follows:

$$D_{\text{com}} = D^S + D^E + D^C + n \times D^U + D^N. \tag{2}$$

The network delay (D^N) is further broken down into the delay within a Wi-Fi network (D_W^N) and the switch delay (D_S^N). The network topology assumed for our case study is shown in Fig. 9. If both source and destination devices are in the same Wi-Fi network, then the network delay is $2 \times D_W^N$. Otherwise (i.e., not in the same network), a message sent from the source should go through an access point where the source device locates, the network switch, and another access point where the destination device is. The network delay is $2 \times D_W^N + D_S^N$.

We measure the delay components from our prototype as shown in the last column of Table V. We plugged the measured delay components to the analytical model and compared the result with the actual measurement to validate the model. The result is shown in Table VI. It confirms that the analytical model correctly captures most of the major delay components.

TABLE VI
VALIDATION OF THE ANALYTICAL MODEL

Latency	Model	Measurement	Accuracy
D_{end}	102.52 ms	110.81 ms	92.51 %
D_{com}	125.85 ms	128.95 ms	97.59 %

TABLE VII
NETWORK PARAMETERS OF FOUR NETWORKS USED TO MEASURE THE LATENCY OVERHEAD

Parameter	Net1	Net2	Net3	Net4
No. of Wi-Fi networks (w)	2	2	4	5
No. of clients per network (d)	5	10	5	6
No. of IoTCop clients in total (n)	10	20	20	30

C. Large-Scale Evaluation

We measure the latency overhead of four network configurations (see Table VII). The findings are shown in Fig. 13. As a reference, the comparison is made with "no security" and "minimal security." In no security, a message is delivered without checking any security policy. In minimal security, a signature is attached to a message by a sender and verified by a receiver. It represents what is recommended by NIST as minimal security for IoT systems. "Endorsement" means D_{end} and "commitment" means D_{com}, respectively.

The latency of IoTCop increases with the number of total devices (n). Comparing Net2 and Net3, we can see that the network configuration does not have a significant impact on latency. The average endorsement latency (D_{end}) is estimated as 261.77 ms, 460.67 ms, 460.57 ms, and 659.46 ms, for Net1, Net2, Net3, and Net4, respectively. The average commitment latency (D_{com}) is 298.62, 514.42, 514.32, and 730.11 ms for Net1, Net2, Net3, and Net4, respectively. Compared to no security and minimal security, the latency of IoTCop is higher and increasing with n while they remain the same. However, it should be noted that compared to Ethereum, the latency of IoT is still significantly lower (i.e., 730.11 ms versus 9.6 s).

When the number of devices grows, the latency also increases. We can address this issue by employing a faster DSA implementation. We notice that most of the latency is incurred by DSA. For example, it takes 9.96 and 7.49 ms on a mid-end device for signature generation and verification, respectively. It corresponds to 46.21%, 41.14%, 75.31%, 49.92%, and 69.41% of D^S, D^E, D^R, D^C, and D^U, respectively. Therefore, if we use a faster DSA algorithm or reduce its latency by employing a hardware accelerator, we expect that the overall latency can be reduced drastically. We can achieve and/or enhance scalability by using the methods discussed in Section V-D.

D. Security Validation

Here, we will analyze the security of IoTCop.

False Command Injection: By exploiting vulnerabilities of a device, an adversary may compromise its firmware. Consequently, the private key may also be exposed. In other words, using the malicious firmware attack, an adversary may compromise a device to inject false commands to other devices.

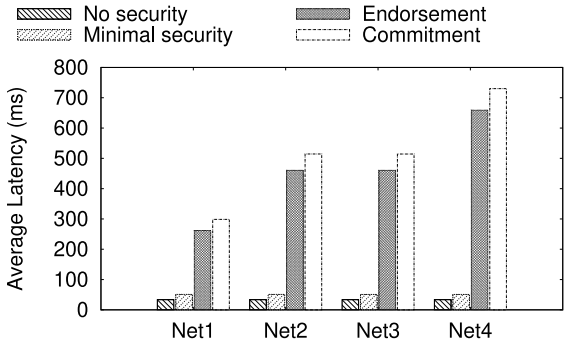


Fig. 13. Average latency of IoTCop. In no security, a message is delivered without checking any security policy. In minimal security, a signature is attached to a message by a sender and verified by a receiver. Endorsement means D_{end} and commitment means D_{com}.

We evaluate this scenario by changing the behavior of a source device in our evaluation. Specifically, the source device sends commands that are against the predetermined security policy and findings from the evaluation demonstrate that IoTCop is able to successfully detect the source device and isolate the attack.

Endorsing Peer Attack: An adversary may attempt to circumvent the security policy by compromising endorsing peers. We evaluate this scenario by changing the behavior of one endorsing peer, which allows it to grant any message without checking the security policy. When the source device injects false commands, findings from the evaluation demonstrate that IoTCop is able to successfully isolate the source device as long as the majority of endorsing peers are not compromised.

Committing Peer Attack: An adversary may attempt to circumvent IoTCop by compromising committing peers. We evaluate this scenario by erasing all security policies in a committing peer. When the source device injects false commands, findings from the evaluation demonstrate that IoTCop is able to successfully enforce the security policies as long as the majority of committing peers are not compromised.

VIII. CONCLUSION

In this article, we proposed IoTCop, a blockchain-based IoT monitoring framework, designed to detect and isolate a device that has been compromised by malicious firmware attacks or physical attacks. A malicious device is detected by checking its behavior against the organization's security policies. To demonstrate how we address the three challenges (latency, applicability, and resource constraints) using the proposed Hyperledger Fabric blockchain framework and add-on hardware modules, we evaluated the proposed framework in a BACnet setting. The findings showed that the latency to deliver a message is order-of-magnitude lower than the Ethereum blockchain framework. For example, the add-on hardware modules enable COTS devices to join IoTCop by pairing a hardware module with a device, and the use of hardware modules offloads the burden of running blockchain clients from the device. We expect that IoTCop is a practical and general IoT monitoring framework that can be applied to various IoT systems by addressing these three challenges.

REFERENCES

[1] Y. Yang, L. Wu, G. Yin, L. Li, and H. Zhao, "A survey on security and privacy issues in Internet-of-Things," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1250–1258, Oct. 2017.

[2] *New Wave of Attacks Aiming to Rope Home Routers Into IoT Botnets*, Trend Micro, Shibuya City, Japan, 2020. [Online]. Available: https://www.helpnetsecurity.com/2020/07/17/home-routers-iot-botnets/

[3] *Mirai Botnet Exploit Weaponized to Attack IoT Devices via CVE-2020-5902*, Trend Micro, Shibuya City, Japan, 2020. [Online]. Available: https://blog.trendmicro.com/trendlabs-security-intelligence/mirai-botnet-exploit-weaponized-to-attack-iot-devices-via-cve-2020-5902/

[4] T. Alladi, V. Chamola, B. Sikdar, and K. R. Choo, "Consumer IoT: Security vulnerability case studies and solutions," *IEEE Consum. Electron. Mag.*, vol. 9, no. 2, pp. 17–25, Mar. 2020.

[5] S. Sridhar and S. Smys, "Intelligent security framework for IoT devices cryptography based end-to-end security architecture," in *Proc. Int. Conf. Inventive Syst. Control (ICISC)*, Coimbatore, India, Jan. 2017, pp. 1–5.

[6] M. Pahl and L. Donini, "Securing IoT microservices with certificates," in *Proc. IEEE/IFIP Netw. Oper. Manag. Symp. (NOMS)*, Taipei, Taiwan, Apr. 2018, pp. 1–5.

[7] J. Kuusijärvi, R. Savola, P. Savolainen, and A. Evesti, "Mitigating IoT security threats with a trusted network element," in *Proc. 11th Int. Conf. Internet Technol. Secured Trans. (ICITST)*, Barcelona, Spain, Dec. 2016, pp. 260–265.

[8] A. Rullo, E. Serra, E. Bertino, and J. Lobo, "Shortfall-based optimal security provisioning for Internet of Things," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Atlanta, GA, USA, Jun. 2017, pp. 2585–2586.

[9] G. Ramachandran and D. Hart, "A P2P intrusion detection system based on mobile agents," in *Proc. 42nd Annu. Southeast Regional Conf.*, 2004, pp. 185–190. [Online]. Available: http://doi.acm.org/10.1145/986537.986581

[10] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2016, pp. 3–16.

[11] S. Babar, A. Stango, N. Prasad, J. Sen, and R. Prasad, "Proposed embedded security framework for Internet of Things (IoT)," in *Proc. 2nd Int. Conf. Wireless Commun. Veh. Technol. Inf. Theory Aerosp. Electron. Syst. Technol. (Wireless VITAE)*, Chennai, India, Feb. 2011, pp. 1–5.

[12] J. Liang *et al.*, "DeepFuzzer: Accelerated deep greybox fuzzing," *IEEE Trans. Depend. Secure Comput.*, early access, Dec. 20, 2019, doi: 10.1109/TDSC.2019.2961339.

[13] J. Gao, X. Yang, Y. Jiang, H. Song, K.-K. R. Choo, and J. Sun, "Semantic learning based cross-platform binary vulnerability search for IoT devices," *IEEE Trans. Ind. Informat.*, early access, Oct. 15, 2019, doi: 10.1109/TII.2019.2947432.

[14] K. Markantonakis, R. N. Akram, and R. Holloway, "A secure and trusted boot process for avionics wireless networks," in *Proc. Integr. Commun. Navig. Surveillance (ICNS)*, Herndon, VA, USA, Apr. 2016, pp. 1–9.

[15] S. S. Vivek and R. Ramasamy, "Forward secure on-device encryption scheme withstanding cold boot attack," in *Proc. IEEE 2nd Int. Conf. Cyber Security Cloud Comput.*, New York, NY, USA, Nov. 2015, pp. 488–493.

[16] R. H. Jhaveri, N. M. Patel, Y. Zhong, and A. K. Sangaiah, "Sensitivity analysis of an attack-pattern discovery based trusted routing scheme for mobile ad-hoc networks in industrial IoT," *IEEE Access*, vol. 6, pp. 20085–20103, 2018.

[17] P. C. Chan and V. K. Wei, "Preemptive distributed intrusion detection using mobile agents," in *Proc. 11th IEEE Int. Workshops Enabling Technol. Infrastruct. Collaborative Enterprises*, Pittsburgh, PA, USA, Jun. 2002, pp. 103–108.

[18] G. Helmer, J. S. K. Wong, V. G. Honavar, L. Miller, and Y. Wang, "Lightweight agents for intrusion detection," *J. Syst. Softw.*, vol. 67, no. 2, pp. 109–122, Aug. 2003.

[19] J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff, E. Spafford, and D. Zamboni, "An architecture for intrusion detection using autonomous agents," in *Proc. 14th Annu. Comput. Security Appl. Conf. (Cat. No.98EX217)*, Phoenix, AZ, USA, Dec. 1998, pp. 13–24.

[20] (2020). *Hyperledger Fabric Project*. [Online]. Available: https://www.hyperledger.org/projects/fabric

[21] B. Lee and J.-H. Lee, "Blockchain-based secure firmware update for embedded devices in an Internet of Things environment," *J. Supercomput.*, vol. 73, no. 3, pp. 1152–1167, Mar. 2017.

[22] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, "Blockchain for IoT security and privacy: The case study of a smart home," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PerCom Workshops)*, Kona, HI, USA, Mar. 2017, pp. 618–623.

[23] S. Huh, S. Cho, and S. Kim, "Managing IoT devices using blockchain platform," in *Proc. 19th Int. Conf. Adv. Commun. Technol. (ICACT)*, Bongpyeong, South Korea, Feb. 2017, pp. 464–467.

[24] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[25] R. Di Pietro, X. Salleras, M. Signorini, and E. Waisbard, "A blockchain-based trust system for the Internet of Things," in *Proc. 23Nd ACM Symp. Access Control Models Technol.*, 2018, pp. 77–83.

[26] C. Lin, D. He, N. Kumar, X. Huang, P. Vijayakumar, and K. R. Choo, "HomeChain: A blockchain-based secure mutual authentication system for smart homes," *IEEE Internet Things J.*, vol. 7, no. 2, pp. 818–829, Feb. 2020.

[27] S. He, W. Ren, T. Zhu, and K. R. Choo, "BoSMoS: A blockchain-based status monitoring system for defending against unauthorized software updating in industrial Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 2, pp. 948–959, Feb. 2020.

[28] J. Wang, L. Wu, K. R. Choo, and D. He, "Blockchain-based anonymous authentication with key management for smart grid edge computing infrastructure," *IEEE Trans. Ind. Informat.*, vol. 16, no. 3, pp. 1984–1992, Mar. 2020.

[29] A. Riahi, E. Natalizio, Y. Challal, N. Mitton, and A. Iera, "A systemic and cognitive approach for IoT security," in *Proc. Int. Conf. Comput. Netw. Commun. (ICNC)*, Honolulu, HI, USA, Feb. 2014, pp. 183–188.

[30] A. Riahi, Y. Challal, E. Natalizio, Z. Chtourou, and A. Bouabdallah, "A systemic approach for IoT security," in *Proc. IEEE Int. Conf. Distrib. Comput. Sens. Syst.*, Cambridge, MA, USA, May 2013, pp. 351–355.

[31] A. F. A. Rahman, M. Daud, and M. Z. Mohamad, "Securing sensor to cloud ecosystem using Internet of Things (IoT) security framework," in *Proc. Int. Conf. Internet Things Cloud Comput.*, 2016, pp. 1–5.

[32] A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla, "SCUBA: Secure code update by attestation in sensor networks," in *Proc. 5th ACM Workshop Wireless Security*, 2006, pp. 85–94.

[33] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla, "Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems," in *Proc. 20th ACM Symp. Oper. Syst Principles*, 2005, pp. 1–16.

[34] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla, "SWATT: Software-based attestation for embedded devices," in *Proc. IEEE Symp. Security Privacy*, Berkeley, CA, USA, May 2004, pp. 272–282.

[35] A. One, "Smashing the stack for fun and profit," *Phrack Mag.*, vol. 7, no. 49, pp. 14–16, 1996.

[36] H. Shacham, "The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)," in *Proc. 14th ACM Conf. Comput. Commun. Security*, 2007, pp. 552–561.

[37] C. Chavda, E. C. Ahn, Y.-S. Chen, Y. Kim, K. Ganesh, and J. Lee, "Vulnerability analysis of on-chip access-control memory," in *Proc. USENIX Workshop Hot Topics Storage File Syst.*, 2017.

[38] C. Sarkar, S. N. A. U. Nambi, R. V. Prasad, A. Rahim, R. Neisse, and G. Baldini, "DIAT: A scalable distributed architecture for IoT," *IEEE Internet Things J.*, vol. 2, no. 3, pp. 230–239, Jun. 2015.

[39] H. M. Kim and M. Laskowski, "Toward an ontology-driven blockchain design for supply-chain provenance," *Intell. Syst. Account. Financ. Manag.*, vol. 25, no. 1, pp. 18–27, 2018.

[40] Z. Li *et al.*, "A blockchain and AutoML approach for open and automated customer service," *IEEE Trans. Ind. Informat.*, vol. 15, no. 6, pp. 3642–3651, Jun. 2019.

[41] Q. Lin, H. Wang, X. Pei, and J. Wang, "Food safety traceability system based on blockchain and EPCIS," *IEEE Access*, vol. 7, pp. 20698–20707, 2019.

[42] E. M. Abou-Nassar, A. M. Iliyasu, P. M. El-Kafrawy, O. Song, A. K. Bashir, and A. A. A. El-Latif, "DITrust chain: Towards blockchain-based trust models for sustainable healthcare iot systems," *IEEE Access*, vol. 8, pp. 111223–111238, 2020.

[43] J. Kang *et al.*, "Blockchain for secure and efficient data sharing in vehicular edge computing and networks," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4660–4670, Jun. 2019.

[44] S. Wang, A. F. Taha, and J. Wang, "Blockchain-assisted crowdsourced energy systems," 2018. [Online]. Available: https://arxiv.org/abs/1802.03099.

[45] E. Münsing, J. Mather, and S. Moura, "Blockchains for decentralized optimization of energy resources in microgrid networks," in *Proc. IEEE Conf. Control Technol. Appl. (CCTA)*, Mauna Lani, HI, USA, Aug. 2017, pp. 2164–2171.

[46] E. Mengelkamp, J. Gärttner, K. Rock, S. Kessler, L. Orsini, and C. Weinhardt, "Designing microgrid energy markets: A case study: The Brooklyn microgrid," *Appl. Energy*, vol. 210, pp. 870–880, Jan. 2018.
[47] Y. Liu, K. Wang, K. Qian, M. Du, and S. Guo, "Tornado: Enabling blockchain in heterogeneous Internet of Things through a space-structured approach," *IEEE Internet Things J.*, vol. 7, no. 2, pp. 1273–1286, Feb. 2020.
[48] H. Noguchi, T. Demizu, N. Hoshikawa, M. Kataoka, and Y. Yamato, "Autonomous device identification architecture for Internet of Things," in *Proc. IEEE 4th World Forum Internet Things (WF-IoT)*, Singapore, Feb. 2018, pp. 407–411.
[49] J. Koo and Y. Kim, "Interoperability of device identification in heterogeneous IoT platforms," in *Proc. 13th Int. Comput. Eng. Conf. (ICENCO)*, Cairo, Egypt, Dec. 2017, pp. 26–29.
[50] Z. Zheng and A. L. N. Reddy, "Safeguarding building automation networks: THE-driven anomaly detector based on traffic analysis," in *Proc. 26th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Vancouver, BC, Canada, Jul. 2017, pp. 1–11.
[51] Z. Pan, S. Hariri, and Y. Al-Nashif, "Anomaly based intrusion detection for building automation and control networks," in *Proc. IEEE/ACS 11th Int. Conf. Comput. Syst. Appl. (AICCSA)*, Doha, Qatar, Nov. 2014, pp. 72–77.
[52] H. Newman, "BACnet explained, part one," *Supplement ASHRAE J.*, vol. 55, no. 11, pp. B2–B7, 2013.
[53] J. Hong, A. Levy, L. Riliskis, and P. Levis, "Don't talk unless I say so! securing the Internet of Things with default-off networking," in *Proc. 3rd ACM/IEEE Int. Conf. Internet Things Design Implement. (IoTDI)*, Orlando, FL, USA, Apr. 2018, pp. 117–128.

Sreenivas Sudarshan Seshadri received the B.E. degree in electronics and communications engineering from Anna University, Chennai, India, in 2013, and the M.S. degree in electrical engineering from the University of Texas at San Antonio, San Antonio, TX, USA, in 2018.

He is currently working as a Power System Engineer with Operation Technology Firm, University of Texas at San Antonio.

David Rodriguez (Member, IEEE) received the B.S. degree in electrical engineering and the M.S. degree in computer engineering from the University of Texas at San Antonio, San Antonio, TX, USA, in 2017 and 2018, respectively, where he is currently pursuing the Ph.D. degree in electrical engineering.

In 2018, he worked as a Graduate Research Assistant in blockchain technology utilizing hyper ledger fabric framework. His research interests include blockchain and adversarial machine learning.

Mukunda Subedi received the B.S. degree in computer engineering from the University of Texas at San Antonio, San Antonio, TX, USA, in December, 2018.

Since then, he has been working as a Software Developer with Tata Consultancy Services, Mumbai, India. Thus far, his client has been American Express and USAA. He is working on the agile development process for a high-end code's defects and bug fixes. His job involves in eclipse, control-M, Jira, SQL, Java, and Javascript.

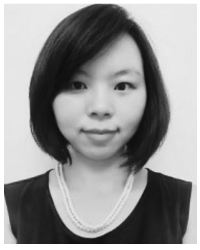
Kim-Kwang Raymond Choo (Senior Member, IEEE) received the Ph.D. degree in information security from the Queensland University of Technology, Brisbane, QLD, Australia, in 2006.

He currently holds the Cloud Technology Endowed Professorship with the University of Texas at San Antonio (UTSA), San Antonio, TX, USA.

Dr. Choo is a recipient of the IEEE Technical Committee on Scalable Computing Award for Excellence in Scalable Computing (Middle Career Researcher) in 2019, the UTSA College of Business Col. Jean Piccione and Lt. Col. Philip Piccione Endowed Research Award for Tenured Faculty in 2018, the Outstanding Associate Editor of IEEE ACCESS in 2018, the British Computer Society's Wilkes Award Runner-up in 2019, the EURASIP JWCN Best Paper Award in 2019, the Korea Information Processing Society's JIPS Survey Paper Award (Gold) in 2019, the IEEE Blockchain Outstanding Paper Award in 2019, the Inscrypt Best Student Paper Award in 2019, the IEEE TrustCom Best Paper Award in 2018, the ESORICS Best Research Paper Award in 2015, the Highly Commended Award by the Australia New Zealand Policing Advisory Agency in 2014, the Fulbright Scholarship in 2009, the Australia Day Achievement Medallion in 2008, and the British Computer Society's Wilkes Award in 2008. He and his team won the Digital Forensics Research Challenge organized by Germany's University of Erlangen–Nuremberg in 2015. He is also a Fellow of the Australian Computer Society, and the Co-Chair of IEEE Multimedia Communications Technical Committee's Digital Rights Management for Multimedia Interest Group.

Sara Ahmed (Member, IEEE) received the bachelor's, master's, and Ph.D. degrees in electrical engineering with a power electronics concentration from the Center for Power Electronics, Virginia Tech, Blacksburg, VA, USA, in 2006, 2007, and 2011, respectively.

She is an Assistant Professor with the Electrical and Computer Engineering Department, University of Texas at San Antonio, San Antonio, TX, USA. She has five years of prior industry experience as a Senior Scientist with ABB's U.S. Corporate Research Center, Raleigh, NC, USA. She holds seven patents, and more than 20 referred publications. Her primary research interests are in the area of modeling, simulation, and analysis of power electronics systems with a focus on control, stability, fault analysis, model prediction, integration of renewables, and hardware-in-the-loop modeling and testing.



Qian Chen (Member, IEEE) received the Ph.D. degree in electrical and computer engineering from Mississippi State University, Mississippi State, MS, USA, in 2014.

She is an Assistant Professor with the Department of Electrical and Computer Engineering, University of Texas at San Antonio, San Antonio, TX, USA. Her primary research area is autonomic computing and cyber security. Her research topics, including human factors and their impacts on cybersecurity, healthcare information system and IoMT security, industrial control systems security (SCADA and IIoT), software vulnerability, and end-to-end security solutions.

Junghee Lee (Member, IEEE) received the B.S. and M.S. degrees in computer engineering from Seoul National University, Seoul, South Korea, in 2000 and 2003, respectively, and the Ph.D. degree in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, GA, USA, in 2013.

He has been with the School of Cyber Security, Korea University, Seoul, since 2019. From 2003 to 2008, he worked with Samsung Electronics, Suwon, South Korea, on electronic system level design of mobile system-on-chip. From 2014 to 2019, he was with the Department of Electrical and Computer Engineering, University of Texas at San Antonio, San Antonio, TX, USA, as an Assistant Professor. His research interests include secure design or hardware-assisted security of processor, nonvolatile memory, storage, and dedicated hardware.