BoSMoS: A Blockchain-Based Status Monitoring System for Defending Against Unauthorized Software Updating in Industrial Internet of Things

Sen He, Wei Ren, *Member, IEEE*, Tianqing Zhu, *Member, IEEE*, and Kim-Kwang Raymond Choo, *Senior Member, IEEE*

***Abstract*—The role of the Industrial Internet of Things (IIoT) in critical infrastructure sectors, such as power, chemistry, and manufacturing, will be increasingly important as we move toward Industry 5.0. For example, IIoT devices are deployed in factories to help the manufacturing companies (e.g., automotive) gain in-depth insight into the various states of production, and thus improving production efficiency and achieving cost reductions. However, malicious code may compromise IIoT devices if either the devices are exposed to outside or unexposed inner devices are updated unauthentically. Due to their limited resources and features, it is challenging to implement strong security solutions for such embedded devices. In this article, we propose a blockchain-based software status monitoring system, called BoSMoS. The system is designed to monitor the software status of IIoT devices to detect and respond to identified malicious behaviors (e.g., intrusions). BoSMoS takes a snapshot of the statue of monitored software and monitors its file system calls. In order to ensure the software integrity information, we use blockchain as the distributed ledger to store a snapshot of software status. The blockchain network of BoSMoS can employ different consensus algorithms. We also evaluate the performance of BoSMoS, in terms of exception response delay, resistance performance to various intrusions, and scalability. The experimental results justify that BoSMoS is practical and sound. In addition, the evaluation of scalability and security demonstrates that the system can carry deployment of large-scale IIoT devices and can guarantee authenticated software updating, as well as detect unauthorized software status.**

***Index Terms*—Blockchain, Industrial Internet of Things (IIoT), software monitoring.**

Manuscript received July 26, 2019; revised September 26, 2019; accepted October 10, 2019. Date of publication October 15, 2019; date of current version February 11, 2020. This work was supported in part by the Major Scientific and Technological Special Project of Guizhou Province under Grant 20183001, in part by the Open Funding of Guizhou Provincial Key Laboratory of Public Big Data under Grant 2018BDKFJJ009 and Grant 2017BDKFJJ006, and in part by the Open Funding of Hubei Provincial Key Laboratory of Intelligent Geo-Information Processing under Grant KLIGIP2016A05. *(Corresponding author: Wei Ren.)*

S. He and W. Ren are with the School of Computer Science, China University of Geosciences, Wuhan 430074, China (e-mail: weirencs@cug.edu.cn).

T. Zhu is with the School of Software, University of Technology Sydney, Sydney, NSW 2007, Australia.

K.-K. R. Choo is with the Department of Information Systems and Cyber Security, University of Texas at San Antonio, San Antonio, TX 78249 USA.

Digital Object Identifier 10.1109/JIOT.2019.2947339

I. Introduction

INDUSTRIAL Internet of Things (IIoT) underpins the evolution of Industry 4.0 in technological advanced countries, such as Germany and the United States [e.g., the national network for manufacturing innovation (NNMI)[1]] and its role will expand in the Industry 5.0 era. Typically in an IIoT network or ecosystem, there are tens to millions of Internet of Things (IoT) devices connected to different systems and networks, providing different services, and performing different functions. Examples of IIoT applications include automated monitoring, control, management, and maintenance [1].

Due to the dedicated nature of most IIoT devices (e.g., designed for specific operations, such as sensing and controlling), there may be customized software running on these embedded devices under strict conditions, and any minor modifications and updates may result in incompatibility with the hardware or operating system. Generally, it is acknowledged that IIoT manufacturers do not implement the strong security solutions to these embedded devices due to the devices' limited computational capabilities, for example in terms of energy, memory and processing capacity. Therefore, such devices are (highly) vulnerable to malicious exploitation, as evidenced by several real-world incidents and the discussion in [2]–[9]. For example, IIoT devices and systems, particularly those in critical industrial facilities, can be targeted by malicious attackers including nation-stated sponsored actors. For the latter group, the attack methods are usually sophisticated and more challenging to defend. IIoT devices generally have insufficient resources to deploy strong security solutions.

Hence, it is not surprising that IIoT/IoT security is an active research area, as evidenced by the numerous and ongoing attempts to map out the threat landscape and design novel security solutions. Sadeghi *et al.* [10], for example, discussed several related security and privacy challenges, as well as potential solutions for IIoT systems. Also as noted by the authors, attacks against IIoT systems can exist on all abstraction layers of the system. In the study of Sicari *et al.* [11], challenges relating to authentication, confidentiality, access

[1]https://www.manufacturing.gov/, last accessed Feb. 1, 2019.

control, privacy, trust, enforcement, secure middleware, and mobile security were highlighted.

One of the several security objectives in IIoT security is to prevent the system or software failure due to unexpected modification(s) that result in physical damage, reduced productivity, or harm to humans. To achieve this objective, the software status of IIoT devices needs to be monitored to ensure integrity and reliability. This is a relatively mature research area in the field of software engineering. Arora *et al.* [12], for example, proposed to observe the program's dynamic execution trace through a hardware monitor in the processor architecture, check whether the software falls within the allowed program behavior, and flag any deviations from expected behavior to trigger appropriate response mechanisms. In a separate work, Munawar and Ward [13] proposed to leverage simple statistical modules to adaptively monitor software systems. Huang *et al.* [14] suggested using software monitoring with controllable overhead (SMCO). However, there are a number of limitations associated with such existing solutions. For example, in the context of IIoT security, limitations include the inability to monitor running software status in a large scale IIoT deployment in real time, and trusted snapshots extracted to check the integrity of software are vulnerable to attacks.

In this article, we propose to deploy software monitoring on the blockchain's peer-to-peer network to mitigate the limitation of monitoring software status in a large scale IIoT deployment. The proposed system can be used in various IIoT environments, such as industrial automation, industrial transportation, etc. [15]. Trusted snapshots are "protected" by the blockchain, in the sense that the snapshots stored on the blockchain are immutable and resilience to single-point failures. In other words, we ensure the integrity and availability of the snapshots and provide consistent, stable, and correct detection references to the monitoring systems. The proposed blockchain network can use different consensus algorithms to meet user needs. Peer-to-peer network architecture significantly reduces the cost of maintaining the network and improves the efficiency of nodes to obtain information. The contribution of this article is as follows.

1) A monitoring method based on block hashing chain over file system call is implemented to monitor the software status of IIoT devices, by leveraging our proposed blockchain network.
2) A blockchain network architecture is proposed to record, guarantee, and verify trusted snapshots instead of unauthorized software updating status.

The remainder of this article is organized as follows. In Sections II and III, we present the relevant background and problem formulation. Our proposed scheme is presented in Section IV, and the performance and security evaluations are presented in Section V. Finally, Section VI concludes this article.

II. Preliminaries

In this section, we briefly introduce the various topics relating to the proposed system.

A. Software Security in IIoT

Software security in engineering software refers to the capability of the software to continue functioning correctly under some malicious attacks [16]. It is a system-level issue that requires consideration of security mechanisms and security-based design. Software security usually focuses on identification and authentication, authorization, auditing, confidentiality, data integrity, etc. Software security is also an important research direction in IIoT. If software security is threatened, it may cause damage to industrial equipment, which may result in loss of asserts and harm to humans.

One objective of software security in IIoT is to protect the availability of IIoT devices, which should prevent any unnecessary delay in production that results in loss of productivity and loss of revenues. In an IIoT environment, special attention must be paid to the prevention of denial-of-service attacks against cyber-physical production equipment.

Another important objective is to preserve the integrity of IIoT software and system, which should prevent any system failure caused by illegal modifications. This particularly includes protection against malicious attacks from saboteurs, which may lead to an unnoticed productivity loss and increased resources occupation.

One of the fundamental objectives is to prevent malicious users from gaining control of the system. Industrial equipment will become a malicious tool if it is controlled by attackers, which may be used to cause physical damage, disrupt the communication of the network or launch distributed denial of service (DDoS) attacks as a bot in botnet.

B. Technical Status of Software

Status monitoring is a concept similar to condition monitoring [17] that we proposed, which monitors the technical status of target software that includes the running conditions, environmental parameters, resource usage, system configurations, etc. All modifications to the related files of target software will result in changes in the technical status of target software. The technical status of a software program may be changed when it is updated, its configure is changed, or it is implanted with malicious code.

During the runtime of a software program, if the files executed or read by the program remains integrity, the technical status of the software will be regarded as normal. Therefore, the information of files accessed by the software program can be used as a feature of the technical status of the software. All files, that target software program has accessed during runtime, with their integrity information can be recorded as a snapshot of the software at the current time. By comparing the obtained snapshot with a trusted snapshot stored in the blockchain, whether the software is in the correct technical status can be determined.

There are three types of files that a software program will access during its runtime: installation files, user profiles, and other dependent libraries.

1) *Installation Files:* These are all the files generated during the software installation process, usually, including the necessary binary files for software execution, dynamic

libraries, static libraries, configuration files, and resource files.

2) *User Profiles:* These are collections of settings and information associated with users. The software provides users with the custom software services through user profiles.
3) *Dependent Libraries:* These are programs, plugins, dynamic libraries, static libraries, and resource files provided by the operating system or other software. By calling the functions implemented by dependent libraries, the amount of work of the software developer can be greatly reduced. However, if the dependent libraries are not installed, the software will not run.

C. Blockchains

A blockchain is a digitized, decentralized, public ledger of all cryptocurrency transactions which was first introduced in 2009 by Nakamoto [18]. It achieves complete decentralization by adopting a peer-to-peer network. Each node of the network maintains a full backup of the blockchain whose blocks can only be added but cannot be modified or deleted. The blockchain was originally created to be used as a public ledger to solve the "double spending" problem in cryptocurrencies [19]. However, the application of blockchain has extended to many fields at present, including IoT, intelligent manufacturing, supply chain management, information sharing, and trading [20]–[24].

The blockchain consist of numerous blocks, and each block consist of two parts. The first part is a block body which stores the important data. For instance, in Bitcoin, blocks store transactions, and in Ethereum, they store transactions or smart contracts [25]. The second part is block header which stores information about itself, such as timestamp, block size, version, etc. Block header of each block except the genesis block contains the hash value of the previous block. Therefore, if a malicious user intends to tamper with a block on blockchain, all blocks behind the block also need to be modified. However, consensus mechanisms are applied to prove the validity of blocks and prevent tampering with blocks. Currently, common consensus mechanisms are proof of work (PoW), proof of stake (PoS), etc. [26].

According to accessing and managing permission, blockchain is divided into three categories: 1) public blockchain; 2) private blockchain; and 3) consortium blockchain [27].

1) *Public Blockchains:* Public blockchains are decentralized; they are open to public, and anyone can participate in as a node. Any node in public blockchains may access the information, submit transactions that would be confirmed, and participate in the consensus procedures therein. Bitcoin and Ethereum are both considered public blockchain.
2) *Private Blockchains:* Private blockchains are the absolute opposite of public blockchains; they are totally centralized. Nodes cannot join a private blockchain unless invited by the network administrators because it is a possession of a single entity or an enterprise which can override or delete commands on a blockchain if needed. Private blockchains are generally applied to database management, audit, and company management.
3) *Consortium Blockchains:* Consortium blockchains are said to be semi-decentralized; they are blockchains with consensus procedures controlled by preset nodes. Consortium blockchains are permissioned as private blockchains, but they are controlled by a number of companies instead of a single organization.

TABLE I
INFLUENTIAL CONSENSUS ALGORITHM

Algorithm	**Application**
PoW	Bitcoin, Litecoin, and the first three stages of Ethereum: Frontier, Homestead and Metroplis
PoS	Peercoin, NXT, and the last stage of Ethereum: Serenity
DPoS	BitShares
Paxos	Google Chubby, Apache ZooKeeper
PBFT	Hyperledger Fabric v0.6
Raft	etcd

D. Consensus Algorithm

The blockchain network is a distributed network. The primary problem of distributed networks is how to solve the problem of consistency, that is, how to reach consensus among multiple independent nodes. There is almost no problem in reaching the agreement in a centralized scenario, but the distributed environment is not so ideal. For example, the communication between nodes may be unreliable, there may be delays and failures, and even the node may be down directly. The consistency problem in multiprocessors and distributed systems is very difficult to solve. The difficulty lies in the following aspects: 1) the distributed system itself may be malfunctioning; 2) communication between the distributed systems can be faulty or have huge delays; 3) distributed systems may run at different speeds; some run very fast, while others are slow; and 4) the Byzantine Generals' problem [28], where there may be malicious nodes in the distributed network.

In order to solve the problem mentioned above, the concept of the consensus algorithm was introduced. Table I lists some influential consensus algorithms and applications that use these consensus algorithms. Differences between IIoT and the Internet environment will result in incompatibility when applying traditional consensus algorithms directly in IIoT environments. Shala *et al.* [29] proposed a trust evaluation model based on the IoT and discussed and evaluated the mainstream consensus algorithm as well as proposed a novel trust consensus protocol.

III. Problem Formulation

A. System Model

The system model of blockchain-based software status monitoring system (BoSMoS) is depicted in Fig. 1. The proposed system consists of blockchain network, blockchain gateway, IIoT devices, and monitoring module, and two special entities: 1) software developer and 2) administrator. The details of these entities are as follows.

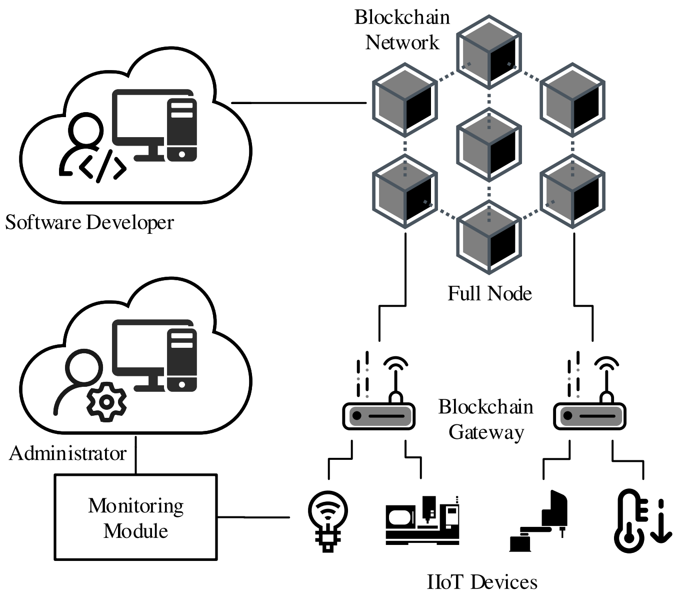

Fig. 1. System model of BoSMoS.

1) Blockchain Network: The blockchain network is used as a trusted decentralized distributed database to store trusted software technical status snapshots. It consists of full nodes and blockchain gateways. Full nodes in the blockchain network are responsible for maintaining the network. They store complete blockchain data and communicate each other via P2P protocol. New blocks can only be generated by full nodes, and all nodes in the network can verify these new blocks by consensus protocol. Using blockchain as a database for storing snapshots can greatly protect the integrity and availability of trust snapshots, preventing attackers from tampering with software snapshots of devices in IIoT. The blockchain makes the snapshots stored therein credible and correct, and these snapshots will be trusted references when monitoring software running on the device.

2) Blockchain Gateway: Blockchain gateways are maintained by the owner of IIoT. They provide blockchain data for IIoT devices and respond to block requests. Limited by computing, storage, and network resources of IIoT devices, they are too weak to be nodes in the blockchain network. Therefore, the blockchain gateway was designed to help IIoT devices to obtain trusted snapshots stored in the blockchain. Each blockchain gateway maintains a local database that stores the complete blockchain data. In the blockchain network, these nodes do not participate in the creation of new blocks, they only receive, verify and store new blocks. On the one hand, blockchain gateways can increase the number of devices that blockchain network can accommodate, on the other hand, they can provide blockchain services for heterogeneous IIoT devices.

3) IIoT Devices: IIoT devices are the main monitoring objects of BoSMoS. All devices, including blockchain gateways and full nodes, run the same monitoring module. However, unlike other nodes, IIoT devices are not part of the blockchain network. Although they also receive and verify blocks, they only store the hash value of these blocks instead of keeping complete blockchain data. IIoT devices will request the corresponding block data from blockchain gateways through blockchain module when a trusted software technical status snapshot is required.

4) Monitoring Module: The monitoring module determines the software status by comparing the software snapshots and monitoring file system calls. The monitoring result will be sent to the administrator terminal. According to the rules set by the administrator, the monitoring module can interrupt the abnormal software by itself.

5) Software Developer: Trusted software technical status snapshots are generated by the software developers and each of them needs to maintain at least one blockchain node. When new software or a new version of software is released, the developer needs to generate the trusted software technical status snapshot. The snapshot will be signed with the private key of the developer and broadcasted in the blockchain network with the software package, signature, and the public key.

6) Administrator: The administrator receives status information of all IIoT devices, blockchain gateways and full nodes maintained by the organization in real time.

B. Adversary Model

Intruders can pretend to be nodes in the blockchain network to start attacks on IIoT, and internal personnel can gain control of important devices to destroy IIoT. Every node is of mutual distrust and can be malicious. Concretely, intruders may have the following malicious behaviors:

1) disguised as blockchain gateways to share incorrect blocks that store tampered snapshot to IIoT devices connected to it;
2) disguised as IIoT devices to carry out DDoS attacks on blockchain gateways;
3) tampering with blockchain data stored in blockchain gateway, causing it to send an erroneous block to IIoT devices;
4) tampering with the block data sent by blockchain gateway to IIoT devices;
5) tampering with the software-related files and local snapshot to disrupt the operating of target software.

C. Design Goals

The goal of BoSMoS is to protect the integrity of software by monitoring the status of it. The software's status information contains software integrity information and software related file information. By comparing trusted software status snapshots, the system can determine the software status and will alert the administrator if the status changes.

In BoSMoS, our design goals mainly include *soundness*, *scalability*, and *security*.

1) *Soundness:* If other nodes communicating with the node are honest, they will respond to the node's request and send trusted block data it needs. In addition, if a device is running normally, the monitoring module can detect the status change of target software and send message to the administrator terminal.

2) *Scalability:* The system can be deployed on a large scale in IIoT. A multitude of nodes can join or leave the network, and their behavior will not have a bad impact on the network.
3) *Security:* There are two aspects to this goal. On the one hand, it is important to ensure the security of the blockchain network, so as to prevent the system from being destroyed by attackers. On the other hand, it is significant to ensure the accuracy of the monitoring module, false negatives, and false positives can have a bad impact on the system.

IV. Proposed Scheme

The proposed scheme provides secure operations to verify an IIoT device's running software technical status and to store the trusted technical status snapshot. If the status is not the same as that stored on the blockchain, the system will issue a warning to the administrator terminal.

A. System Architecture

Model-based design and development of production and manufacturing systems is a critical task and many related studies have been made. In this article, we proposed a new system architecture which is more suitable for monitoring software technical status in IIoT devices.

The system architecture of BoSMoS is illustrated in Fig. 2. According to function, the architecture can be divided into two parts: 1) blockchain network synchronization and 2) software technical status monitor. In the first part, the software developer generates the trusted snapshot and submits it with the software to a full node which will verify the validity of the snapshot. Verified snapshot will be packaged into a block and broadcasted to the blockchain network, and all nodes in the blockchain network that receive the block must determine whether to accept it through consensus protocol. Besides, as the node that packages the block, full nodes that receive the block also need to verify the snapshot. In the second part, monitoring module obtains the trust snapshot of target software from blockchain gateway and generates file access whitelist from it. At each set time, the monitoring module takes a snapshot of target software and compares it with the trusted snapshot to judge the software status. In addition, the monitoring module also monitors file system calls, it will respond promptly when target software accesses files that are not on the file access whitelist. All monitoring results are sent to the administrator, so he can detect system anomalies and take action in time.

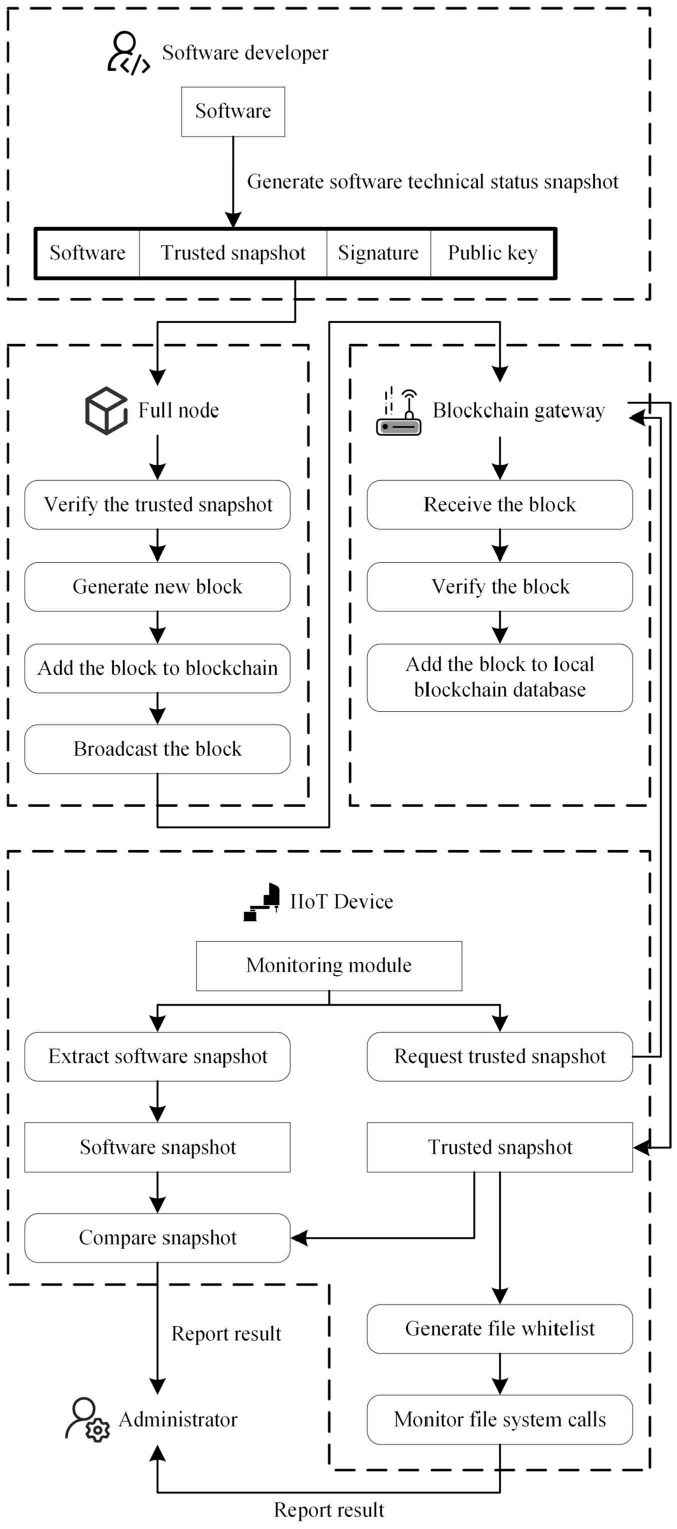

Fig. 2. System architecture of BoSMoS.

B. Blockchain Network Initialization

In a blockchain network, each node generates its own public-private key pair of asymmetric cryptographic algorithm locally, and the private key will be used to generate the digital signature when it initiates a transaction or packs a block.

In the proposed system, all full nodes will generate private key and public key locally, and the keys will play a significant role in the block generation and verification process. The blockchain gateways can also generate keys if they also participates in the block creation process, in which they also acted as full nodes. IIoT devices are not nodes of the blockchain network, so they do not need to generate keys for it. Software developers should belong to a known trusted organization whose keys can be distributed and authenticated by a trusted certification authority. All nodes except software developers generate and store their keys locally.

TABLE II
STRUCTURE OF BLOCK

Item	Description
block header	The data structure of the metadata of this block.
software signature	A digital code generated by the software developer who generate the trusted snapshot. It is obtained by signing software hash and trusted snapshot with software developer's private key.
developer public key	The software developer's public key in asymmetric cryptography. This is used to verify the signature of the software and snapshot.
data num	The number of items in the snapshot
snapshot	The data stored in the block as a list. In file blocks, it stores the file-hash pairs and dependencies information list. In key blocks, it stores key list.

TABLE III
STRUCTURE OF BLOCK HEADER

Item	Description
version	This indicates the version of the protocol used by the block for future expansion.
timestamp	A Unix epoch time when the full node started creating the header.
previous block header hash	A hash of the previous block's header. This ensure no previous block can be changed without also changing the block's header.
data list hash	A hash of the block's data list.
signature	A digital code generated by the full node's private key and can only be authorized by the its public key. This ensure no one except full node can generate a valid block.
public key	The full node's public key in asymmetric cryptography. This is used to verify the signature of the block.
software name	The name of monitored software.
software version	The version of monitored software.

C. Blockchain Network Synchronization

The blockchain network consists of two types of nodes: 1) full nodes and 2) blockchain gateways. Software developer as a special entity accesses the network by owning blockchain nodes. Other nodes can distinguish it by the public key it held. In the proposed scheme, generation of a new block starts from the software developer and the block is gradually synchronized to each node. In this section, we will show how trusted snapshots and new blocks are generated and verified.

1) Trusted Snapshots Generation and Verification: Trusted snapshots are extracted by the software developer through the monitoring module under normal environment configuration, and the process of extracting snapshots is introduced in Section IV-D. After the software developer extracts the trusted snapshot, it will be combined with hash value of the software and they will be signed using the digital signature algorithm with the private key of the software developer. After that, a transaction containing software, trusted snapshot, digital signature, and public key of a software developer will be created and sent to the blockchain network.

After receiving the transaction, the full node first verifies the digital signature. Then it extracts the snapshot information from the software through the monitoring module and compares it with the trusted snapshot in the transaction to verify the snapshot. In this process, if the signature verification fails or the snapshot verification fails, the full node will ignore the transaction.

2) Blocks Generation and Verification: Block generation, verification, and synchronization process must follow a consensus algorithm. The blockchain network we designed can be applied to a variety of existing consensus algorithms, such as PoW, PoS, or PBFT. Consensus algorithms are not the focus of our discussion, so we have omitted the specific consensus process in this section and believe that the blockchain network is secure. How the proposed scheme applies different consensus algorithms will be discussed in Section IV-E.

Full node will generate a new block if the verification of trusted snapshot succeed. The block structure we designed is shown in Table II. Block mainly stores trusted snapshot of software, and the metadata of block is stored in the block header whose structure is depicted in Table III.

The signature and public key in the transaction will be stored in the item "software signature" and "developer public key" of the block structure directly. The block will be broadcasted along with the software after it is generated according to the data structure given in the table.

All full nodes that receive the block first verify the trusted snapshot through the software and the software signature, developer public key, and snapshot in the block. Then full nodes will perform the blockchain consensus process to determine the validity of the block. If any of the verification processes fail, the block will be ignored, otherwise, it will be added to the node's local blockchain database. Each blockchain gateway that received the block only needs to verify the block through the consensus algorithm, and then join the verified block to the local blockchain database.

3) Block Verification of IIoT Device: As discussed earlier, IIoT devices have limited resources to be strict blockchain network nodes. Instead of storing the entire blockchain data, they only cache the required blocks which are obtained from blockchain gateways. Although blockchain gateways are maintained by the owner of the IIoT, it cannot ignore the possibility of these gateways being compromised. Inspired by the design of lightweight node [30] in Bitcoin network, we have designed a low-cost, secure block verification process for IIoT devices.

IIoT devices receive blocks broadcasted by full nodes. Although they do not store these blocks, hash values of these blocks are calculated and stored to form a hash chain whose structure is depicted in Fig. 3. Each time a new block is received, an IIoT device calculates the hash value of the previous data unit and combines it with the hash value of the received block to form a new data unit. Fig. 3(a) depicts that process. The shape of hash chain is consistent with blockchain. When there is a branch in blockchain, hash chain also branches, as shown in Fig. 3(b).

When the monitoring module of an IIoT device needs trusted snapshot which is not stored locally, the blockchain module will request the block from blockchain gateway. After receiving the requested block, the blockchain module not only verifies the validity of the block through consensus protocol but also searches for the hash value of the block on the hash chain to verify the integrity of the block. It indicates that there

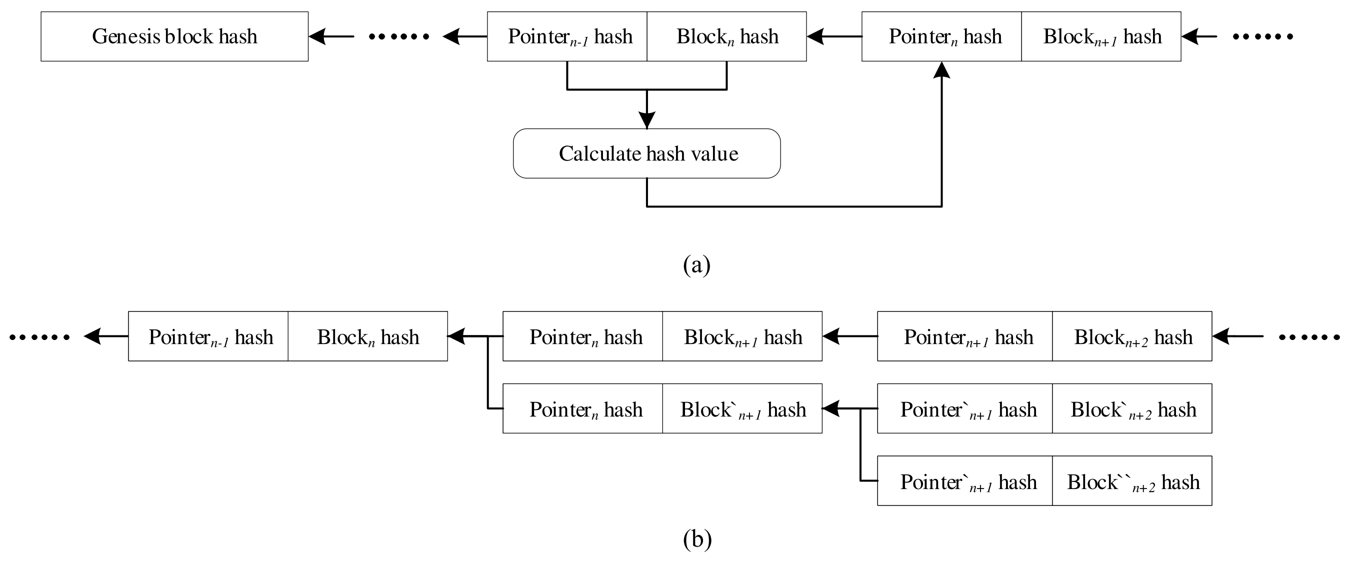

Fig. 3. Hash chain stored in IIoT devices. (a) Generation process of hash pointer and the case of no branching. (b) Case with branches.

may be intrusions in the network if block verification process of IIoT devices fails. In this case, the blockchain module of IIoT devices will warn the administrator.

D. Software Technical Status Monitor

The monitoring module performs two main operations.

1) It obtains information of installation files, user profiles, and dependent libraries of target software to generate software technical status snapshot which is going to be compared with the trusted snapshot stored on the blockchain.
2) It generates a file access whitelist through trusted snapshot and monitors the file system calls requested by the target software. Trusted snapshots are generated by the software developers.

When node n_1 in the network starts running monitoring module, it will perform the first function as the following steps.

1) It will read the information of the target software, including software name, version, dependencies, dependencies version, and absolute path to installation files and user profiles.
2) It will compute the hash value of installation files and user profiles, and store them in the list as (*file_path*, *file_hash*) pair.
3) It will store dependencies and dependencies version requirement at the end of the list as (*dependency*, *version requirement*) pair. The list is the software technical status snapshot at this moment. Table IV shows an example of the structure of the list.
4) If n_1 is an IIoT device, it will send a *req_snapshot* message including public key of n_1, name and version of target software to a blockchain gateway. Otherwise, n_1 will use software name and version to search it on a local blockchain database.
5) Node n_1 will obtain the trusted snapshot of target software from the blockchain gateway or the local blockchain database. If n_1 is an IIoT device, it will verify the block it received from blockchain gateway by checking the hash value of the block.
6) The monitoring module will compare the snapshot generated in steps 2) and 3) with the trusted snapshot to judge if the software is in a normal technical status. Otherwise, the system will report an error and send the information to the administrator terminal.
7) At intervals of t minutes (set by the administrator), the system will repeat steps 1)–6).

When the above steps are completed for the first time, the monitoring module immediately generates a file access whitelist and starts monitoring all file system calls requested by the target software. The process is as follows.

1) Extract the file path from trusted snapshot to generate a file list.
2) Query blockchain gateway or local blockchain database for trusted snapshots of all the dependencies of a target software. The monitoring module will deliver an error if the trusted snapshot of a dependency cannot be searched in the blockchain.
3) Add file information at trusted snapshots obtained in the previous step to the list to finish the whitelist of file accesses.

TABLE IV
EXAMPLE OF SOFTWARE TECHNICAL STATUS SNAPSHOT

/opt/google/chrome/nacl_helper	33b6489a22db159af4aed39ef055512afe2d e8ead852929afc50a688cef5d6ac
/opt/google/chrome/resource.pak	32fdded137dd1d1265fdba14c412cb01847 22d9c5f11c8e4760c1e671c6608a0
/usr/bin/google-chrome-stable	9fceed1cde430e40cad07f3662211122781f 5c6da3e4e23432119ea6c91c4b14
/etc/cron.daily/google-chrome	3b39d48cb80e8b28d58e8ceb9e9ab762cda 330ac674938a20733e9a031e6aa5a
libatk-bridge2.0-0	>= 2.5.3
libexpat1	>= 2.0.1

The first four rows represent a part of the file-hash pairs of the files that generated by the target software.
The last two rows show a part of the dependencies' name and the required version of the package.

4) The monitoring module creates a new thread to monitor file system calls requested by target software. Information about all requested files that are not on the file access whitelist will be recorded in a log file and sent to the administrator terminal with a warning message.

During the operation of the monitoring module, if software's technical status changes, or the software accesses a file that it should not access, the relevant error information will be fed back to the administrator's terminal. Through setting, it can terminate the operation of target software.

Some software needs to execute files provided by dependencies at runtime, and these dependencies also need to be monitored. The structure of the software technical status snapshot that we proposed contains the information of dependencies. Nodes in the network can index the blocks which store the trusted snapshot of dependencies by its name and version. The newly executed program will trigger the monitoring module and perform all monitoring steps from the beginning.

E. Discussion

As described above, the power of our proposed approach relies on the use of blockchain technology which makes it possible to monitor large-scale IIoT devices (*scalability*). The blockchain network model we proposed is universal for different consensus algorithms. Each consensus algorithm has different strengths and weaknesses. Different consensus algorithms should be applied for different industrial scenarios to meet specific needs. We use PoW and PBFT algorithms as examples to explain how our blockchain network applies these consensus algorithms.

1) BoSMoS With PoW: In addition to verifying trusted snapshots, full nodes that create new blocks in the proposed blockchain network also need to calculate the nonce based on the difficulty. Other nodes need to verify the nonce when receiving the block [18]. Blockchain gateways do not participate in the generation of blocks, nor do they verify trusted snapshots. They only store and validate blocks. When the blockchain branches, all nodes trust the longest branch and full nodes only generate new blocks on it. Therefore, blockchain gateways do not need to care about the security of trusted snapshots stored in the blocks of the longest branch, because they have all been verified by other full nodes.

2) BoSMoS With PBFT: All full nodes in the proposed blockchain network participate in the consensus process of the PBFT algorithm [28]. Blockchain gateways do not participate in the consensus process, but they will track the process and store confirmed blocks. Blockchain gateways also do not need to verify trusted snapshots stored in the block, because at least 2/3 of full nodes have already confirmed these snapshots according to the PBFT algorithm.

Our proposed blockchain network is also designed to meet the *security* requirements. Trusted snapshots stored on the blockchain are immutable and resistant to single point of failures (SPOFs), which ensures the integrity and availability of the snapshots and provides consistent, stable, and trusted references for the monitoring module.

TABLE V
EXPERIMENTAL COMPUTER FEATURES

System information	**Features**
CPU architecture	x86_64
CPU operation mode	64-bit
CPU max speed	3300 MHz
RAM	2 GB
Operation system	Ubuntu 18.04.1

V. EVALUATION AND DISCUSSION

A. Evaluation Framework

According to our scheme, we implemented the proposed system in Python3.6 and used BigchainDB as our blockchain network platform [31]. BigchainDB is a blockchain platform that has blockchain properties and database properties which uses Tendermint for all networking and consensus [32]. BigchainDB is used because it provides many database features to facilitate data storage.

We simulated a network of ten IIoT devices on a computer in function assessment whose features are listed in Table V. These ten nodes are composed of four full nodes (one software developer, one administrator, and two normal full nodes), two blockchain gateways, and four IIoT devices. Additionally, in the scalability assessment, we expanded the number of IIoT devices to 2000 and the number of blockchain gateways to 100 and tested the maximum number of devices that a single blockchain gateway can connect stably. The experimental network architecture is shown in Fig. 4.

Tendermint is a consensus algorithm based on PBFT. However, consensus algorithm is not the focus of our discussion, so we do not discuss specific consensus process and assume that the blockchain network is secure and trustworthy. Therefore, the main evaluation targets are blockchain gateways and IIoT nodes. Under this premise, software status monitoring performance, intrusion resistance performance, and computation performance of BoSMoS have been tested by the experiment. In addition, scalability and security are also analyzed.

Message queuing telemetry transport (MQTT) is an IoT connectivity protocol which is designed as an extremely lightweight publish/subscribe messaging transport.[2] It can well meet the requirements of the proposed system, so we choose to apply this to implement the system. The docker version emqx[3] tool was used to deploy full nodes and blockchain gateways. In the scalability assessment, we use the emqtt_bench[4] tool to simulate IIoT devices to evaluate network.

B. Evaluation Results

1) Software Status Monitoring Performance: There are two cases where the integrity of target software has been tampered with: 1) the software has been tampered with before it runs and 2) relevant files have been maliciously modified during the software runtime. We define *mt* as the time the software

[2]http://mqtt.org, last accessed Sep. 20, 2019.
[3]https://github.com/emqx/emqx, last accessed Sep. 20, 2019.
[4]https://github.com/emqx/emqtt-bench, last accessed Sep. 20, 2019.

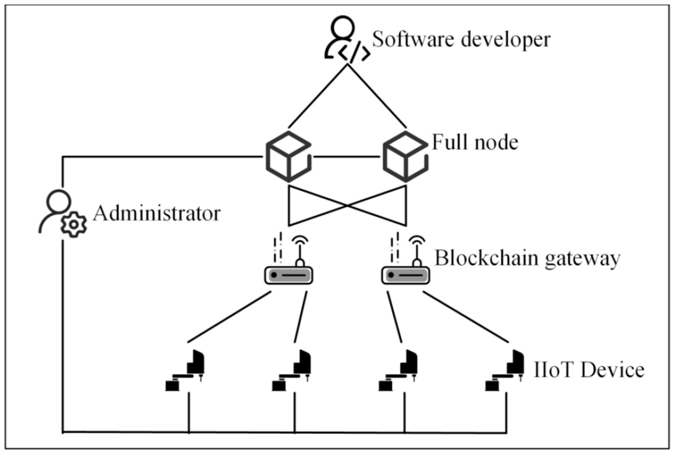

Fig. 4. Experimental network architecture.

TABLE VI
INTRUSION RESISTANCE PERFORMANCE

Intrusion type	**Whether it is detected**
Blockchain gateway camouflage	Yes
IIoT device camouflage and DDoS	No
Blockchain data tampering	Yes
Block message tampering	Yes

is modified and dt as the time the modification is discovered. Then exception response delay ed satisfies the formula $ed = dt - mt$. Installation files, user profiles, and dependent libraries described in Section II-B have been modified, respectively, in both cases to test the response delay to the software status exception.

In the first case, BoSMoS checks the software integrity by comparing the software status snapshot with Ts before it starts running. In this case, dt can be considered as the startup time of target software, and mt is any time less than dt. Therefore, ed is related to the time the software was started after it was modified.

In the second case, BoSMoS takes a software status snapshot at regular intervals to check software integrity. In this experiment, the time interval is set to 300 s. The startup time of target software is taken as the initial time of the experiment to compute mt and dt.

By modifying the software files at different times, we obtain the experimental results as shown in Fig. 5. As can be seen from the relationship between mt and dt reflected in Fig. 5(a), dt is related to the time interval set by the user. The results shown in Fig. 5(b) depict that the closer mt is to the next detection, the smaller ed will be.

2) Intrusion Resistance Performance: Intrusions listed in Section III-B were tested except software files tampering which has been discussed in the previous section. Experimental results are shown in Table VI.

It can be concluded from the experimental results that the IIoT device can detect incorrect block data received from the blockchain gateway. In spite of the fact that this kind of detection is coarse-grained and the proposed system does not know how the intruder specifically implements the intrusion, the system can still warn administrators to take further action.

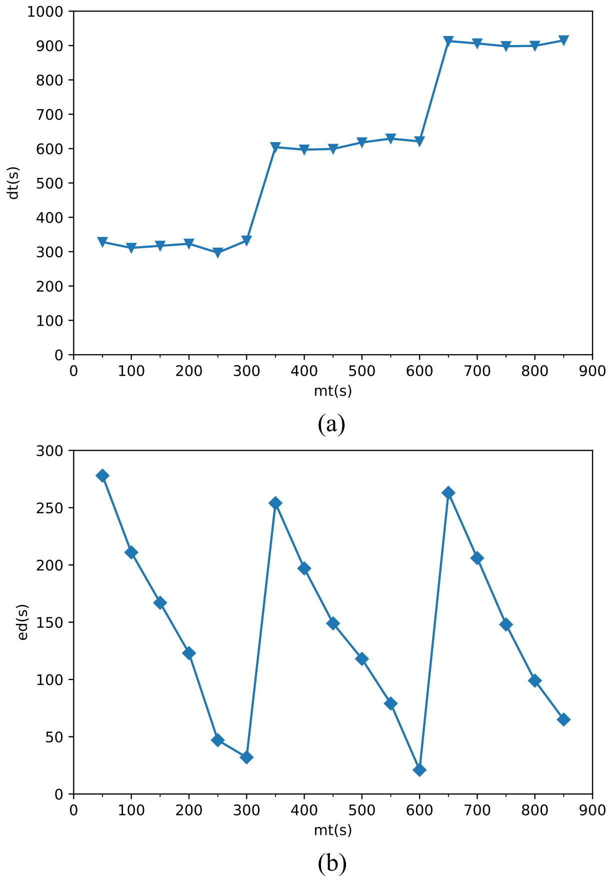

Fig. 5. Exception response delay for monitoring module. (a) Schematic of mt and dt. (b) Schematic of mt and ed.

BoSMoS is not optimized for DDoS attacks. The detection and protection of DDoS attacks can be designed in the communication protocol, which is not the main design goal of the proposed system. In addition, the decentralized network architecture of blockchain allows a few nodes to fail, resists a certain degree of DDoS attacks, and gives administrators more time to respond.

3) Computation Performance: Computing resource consumption of BoSMoS for IIoT devices should be as small as possible. Excessive computing resources may affect the operation of IIoT device. In the proposed system, IIoT devices mainly consume a large amount of computing resources when extracting software status snapshots. This means that the computation performance of the system can be calculated by evaluating the performance of the IIoT device to calculate hash values. Pereira *et al.* [33] have evaluated performance of the cryptographic algorithms over various mainstream IoT platforms and the operating systems in their work. They have evaluated two kinds of hash algorithms: Blake2 and Keccak. From their work, we can conclude that consumption of computing resources is related to the total size of software files and the resource consumption is affordable for IoT devices especially when using the Blake2s algorithm. Consequently, we can claim that BoSMoS has great computation performance on IIoT devices.

4) Scalability Evaluation: Before determining the number of nodes to simulate, we first tested the maximum number

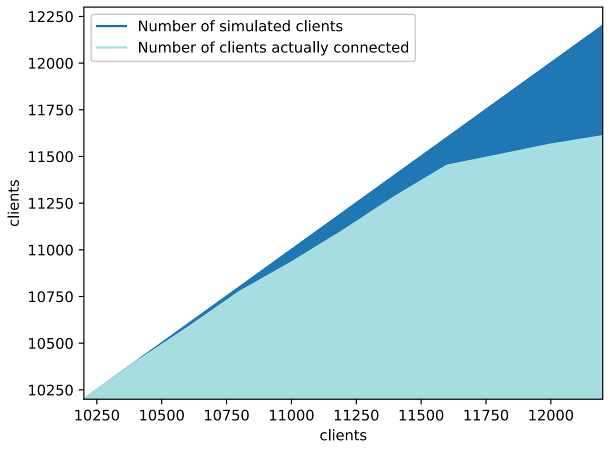

Fig. 6. Number of IIoT devices connected by a single blockchain gateway.

TABLE VII
NETWORK SITUATION

Payload Size (bytes)	msg/sec	Total throughput
256	10,013	2,563,328
512	10,004	5,122,048
1k	10,050	10,291,200
2k	10,044	20,570,112
3k	8,216	25,239,552
4k	6,998	28,663,808
5k	5,385	27,571,200
6k	4,633	28,265,152

of devices that the experimental virtual machine can carry. Two hundred simulated full nodes were set up to continuously send messages in the blockchain network. In addition, 200 simulated IIoT device nodes were added each time to test the maximum load of a single blockchain gateway. The results were shown in Fig. 6. After the total number of nodes exceeded 10 400, some IIoT nodes started to drop which indicated that a blockchain gateway can stably connect more than 10 000 IIoT device nodes in the test environment. Moreover, the console of the tested blockchain gateway shows that with so many nodes connected, only nearly 300 MB of memory was occupied and CPU usage did not exceed 20%.

The results on network payload and throughput for 100 blockchain gateways and 2000 IIoT devices are shown in Table VII. From the table we can conclude that the throughput of the test network is around 28 000 000 B/s $\approx$ 26.7 MB/s, and a larger evaluation can refer to the test done by the Xmeter team.[5]

Devices in the blockchain network can freely join or leave the network without affecting other nodes. Depending on the consensus algorithm used by the blockchain network, the number of full nodes that the network can accommodate will vary. However, the experiment proved that the blockchain gateways can greatly expand the number of devices that a blockchain network can accommodate, thereby enhancing its scalability.

In terms of storage of IIoT devices, hash chain allows them to store only 1024 bits data per block in the case of using sha-512 algorithm, and 1-MB storage space can store 8192 blocks. In addition, the release and update of software will not be very frequent, so the number of blocks will not expand too fast. Even if the number of blocks grows to the current number of blocks in Bitcoin (less than 589 824 blocks), IIoT devices only need 72 MB storage space to store the hash values of these blocks. Therefore, the limited storage resource of IIoT devices will not become an obstacle to the scalability of the system.

In a nutshell, the proposed system can achieve strong scalability.

5) *Security Evaluation:*

a) Software integrity: BoSMoS uses the blockchain network to store software integrity information. It periodically checks software integrity by comparing the software status snapshots and monitoring file system calls. From the experimental results, the response of BoSMoS to software status exceptions is related to the time interval set by the administrator. The shorter the monitoring interval, the faster the exceptions can be discovered, but the resource consumption of device will be greater.

b) Data integrity: As nodes in the blockchain network, full nodes and blockchain gateways maintain block data consistency through the consensus algorithms. IIoT devices synchronizes the integrity information of the entire blockchain by maintaining hash chain to verify the integrity of the requested block received from blockchain gateways. Any changes to the block data can be found and reported to the administrator. Data integrity is well protected by BoSMoS.

c) Intrusion prevention: BoSMoS can detect attacks from intruders in a coarse-grained manner and send the exception information to administrator according to the experimental results. In addition, IIoT devices that detect network anomalies can change the blockchain gateway they communicate with and request the block again.

d) Transmission security: It is a tough task to ensure transmission security in a low-reliability IIoT environment. As the case stands, there is already a lot of work devoted to this field, such as MQTT, Coap, AMQP, etc. [34]–[36]. These communication protocols also provide some encryption methods to deal with eavesdropping. Transmission security can be protected by applying these protocols in communication between the blockchain network nodes and IIoT device nodes.

VI. CONCLUSION

In this article, we proposed BoSMoS that can be deployed in a large scale IIoT setting. We also evaluated the security and performance of the proposed system (e.g., software status monitoring performance, intrusion resistance performance, and computation performance). Findings from the evaluation indicated that the exception response delay is related to the system monitoring period set by the user. In order to detect the software status exception in a more timely manner, BoSMoS generates a file access whitelist through trusted snapshot and monitors file system calls. Security analysis of software integrity and experimental results of computation performance indicate that the system can effectively protect

[5]https://emq-xmeter-benchmark-cn.readthedocs.io/en/latest/throughput.html, last accessed Sep. 25, 2019.

software integrity and achieve *soundness*. Scalability evaluation indicates that although the number of full nodes that BoSMoS can accommodate varies according to the consensus algorithm used, the design of blockchain gateway can greatly increase the IIoT devices that BoSMoS can accommodate. Furthermore, the experimental results show that under the conditions of this article, a single blockchain gateway can stably connect more than 10 000 IIoT device nodes and provide effective blockchain data services without consuming too much resources. In a nutshell, BoSMoS has strong *scalability*. Intrusion resistance experiments prove that BoSMoS can resist the attack of intruders to a certain extent. Experiment results and security analysis indicate that BoSMoS achieves *security* under the assumption that the consensus algorithm used by the blockchain network is secure.

Future extensions include designing a suitable consensus algorithm for our scheme and deploying the extended system in a real-world setting.

References

[1] L. Da Xu, W. He, and S. Li, "Internet of Things in industries: A survey," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 2233–2243, Nov. 2014.

[2] J. Vijayan, "Stuxnet renews power grid security concerns," Computerworld, Jul. 2010. Accessed: Oct. 26, 2019. [Online]. Available: https://www.computerworld.com/article/2519574/stuxnet-renews-power-grid-security-concerns.html

[3] J. Pollet and J. Cummins, "Electricity for free? The dirty underbelly of SCADA and smart meters," in *Proc. Black Hat USA*, vol. 2010, 2010, pp. 1–24.

[4] K. Koscher *et al.*, "Experimental security analysis of a modern automobile," in *Proc. IEEE Symp. Security Privacy (SP)*, 2010, pp. 447–462.

[5] B. Zhu, A. D. Joseph, and S. Sastry, "A taxonomy of cyber attacks on SCADA systems," in *Proc. IEEE Int. Conf. Internet Things Cyber Phys. Soc. Comput.*, 2011, pp. 380–388.

[6] B. Miller and D. C. Rowe, "A survey SCADA of and critical infrastructure incidents," in *Proc. ACM 1st Annu. Conf. Res. Inf. Technol.*, 2012, pp. 51–56.

[7] C. Miller and C. Valasek, "A survey of remote automotive attack surfaces," in *Proc. Black Hat USA*, vol. 2014, 2014, pp. 1–94.

[8] A. Illera and J. Vidal, "Lights off! The darkness of the smart meters," in *Proc. Block Hat Europe*, Oct. 16, 2014. Accessed: Oct. 26, 2019. [Online]. Available: https://www.blackhat.com/eu-14/briefings.html

[9] G. Hernandez, O. Arias, D. Buentello, and Y. Jin, "Smart nest thermostat: A smart spy in your home," in *Proc. Black Hat USA*, 2014, pp. 1–8.

[10] A.-R. Sadeghi, C. Wachsmann, and M. Waidner, "Security and privacy challenges in industrial Internet of Things," in *Proc. 52nd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, 2015, pp. 1–6.

[11] S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini, "Security, privacy and trust in Internet of Things: The road ahead," *Comput. Netw.*, vol. 76, pp. 146–164, Jan. 2015.

[12] D. Arora, S. Ravi, A. Raghunathan, and N. K. Jha, "Hardware-assisted run-time monitoring for secure program execution on embedded processors," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 12, pp. 1295–1308, Dec. 2006.

[13] M. A. Munawar and P. A. S. Ward, "Leveraging many simple statistical models to adaptively monitor software systems," in *Proc. Int. Symp. Parallel Distrib. Process. Appl.*, 2007, pp. 457–470.

[14] X. Huang *et al.*, "Software monitoring with controllable overhead," *Int. J. Softw. Tools Technol. Transfer*, vol. 14, no. 3, pp. 327–347, 2012.

[15] D. Minoli and B. Occhiogrosso, "Blockchain mechanisms for IoT security," in *Proc. Internet Things*, vol. 1, 2018, pp. 1–13.

[16] G. McGraw, "Software security," *IEEE Security Privacy*, vol. 2, no. 2, pp. 80–83, Mar./Apr. 2004.

[17] B. Rao, *Handbook of Condition Monitoring*. Oxford, U.K.: Elsevier, 1996.

[18] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Accessed: Mar. 2009. [Online]. Available: https://metzdowd.com

[19] D. Pointcheval and J. Stern, "Security arguments for digital signatures and blind signatures," *J. Cryptol.*, vol. 13, no. 3, pp. 361–396, 2000.

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

[21] Z. Li, A. V. Barenji, and G. Q. Huang, "Toward a blockchain cloud manufacturing system as a peer to peer distributed network platform," *Robot. Comput. Integr. Manuf.*, vol. 54, pp. 133–144, Dec. 2018.

[22] F. Tian, "An Agri-food supply chain traceability system for China based on RFID & blockchain technology," in *Proc. IEEE 13th Int. Conf. Service Syst. Service Manag. (ICSSSM)*, 2016, pp. 1–6.

[23] S. Ølnes, J. Ubacht, and M. Janssen, "Blockchain in government: Benefits and implications of distributed ledger technology for information sharing," *Govt. Inf. Quart.*, vol. 34, no. 3, pp. 355–364, 2017.

[24] J. Kang, R. Yu, X. Huang, S. Maharjan, Y. Zhang, and E. Hossain, "Enabling localized peer-to-peer electricity trading among plug-in hybrid electric vehicles using consortium blockchains," *IEEE Trans. Ind. Informat.*, vol. 13, no. 6, pp. 3154–3164, Dec. 2017.

[25] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," vol. 151, Zug, Switzerland, Ethereum Project, Yellow Paper, pp. 1–32, 2014.

[26] Z. Zheng, S. Xie, H.-N. Dai, and H. Wang, "Blockchain challenges and opportunities: A survey," *Int. J. Web Grid Services*, vol. 14, no. 4, pp. 352–375, 2018.

[27] M. Swan, *Blockchain: Blueprint for a New Economy*. Beijing, China: O'Reilly Media, 2015.

[28] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, 1982.

[29] B. Shala, U. Trick, A. Lehmann, B. V. Ghita, and S. Shiaeles, "Novel trust consensus protocol and blockchain-based trust evaluation system for m2m application services," *Internet Things*, vol. 7, Sep. 2019, Art. no. 100058.

[30] A. Gervais, S. Capkun, G. O. Karame, and D. Gruber, "On the privacy provisions of bloom filters in lightweight bitcoin clients," in *Proc. 30th Annu. Comput. Security Appl. Conf.*, 2014, pp. 326–335.

[31] T. McConaghy *et al.*, "BigchainDB: A scalable blockchain database," Berlin, Germany, BigChainDB, White Paper, 2016.

[32] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," Ph.D. dissertation, Eng. Syst. Comput., Univ. Guelph, Guelph, ON, Canada, 2016.

[33] G. C. Pereira *et al.*, "Performance evaluation of cryptographic algorithms over IoT platforms and operating systems," *Security Commun. Netw.*, vol. 2017, Aug. 2017, Art. no. 2046735.

[34] A. Luoto and K. Systä, "Fighting network restrictions of request-response pattern with MQTT," *IET Softw.*, vol. 12, no. 5, pp. 410–417, Oct. 2018.

[35] D. Garcia-Carrillo and R. Marin-Lopez, "Multihop bootstrapping with EAP through COAP intermediaries for IoT," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 4003–4017, Oct. 2018.

[36] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, COAP, AMQP and HTTP," in *Proc. IEEE Int. Syst. Eng. Symp. (ISSE)*, 2017, pp. 1–7.

Sen He is currently pursuing the master's degree with the School of Computer Science, China University of Geosciences, Wuhan, China.

His current research interests include Internet of Things and blockchain.

Wei Ren (M'06) received the Ph.D. degree in computer science from the Huazhong University of Science and Technology, Wuhan, China.

He is currently a Full Professor with the School of Computer Science, China University of Geosciences, Wuhan, China. He was with the Department of Electrical and Computer Engineering, Illinois Institute of Technology, Chicago, IL, USA, in 2007 and 2008; the School of Computer Science, University of Nevada Las Vegas, Las Vegas, NV, USA, in 2006 and 2007; and the Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, in 2004 and 2005. He has published over 70 refereed papers, one monograph, and four textbooks.

Prof. Ren was a recipient of the ten patents and five innovation awards. He is a Senior Member of the China Computer Federation.

Tianqing Zhu (M'11) received the B.Eng. and M.Eng. degrees from Wuhan University, Wuhan, China, in 2000 and 2004, respectively, and the Ph.D. degree in computer science from Deakin University, Geelong, VIC, Australia, in 2014.

She was a Lecturer with the School of Information Technology, Deakin University from 2014 to 2018. She is currently a Senior Lecturer with the School of Software, University of Technology Sydney, Sydney, NSW, Australia. Her current research interests include privacy preservation, data mining, and network security.

Kim-Kwang Raymond Choo (SM'15) 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. In 2016, he was named the Cybersecurity Educator of the Year—APAC (Cybersecurity Excellence Awards are produced in cooperation with the Information Security Community on LinkedIn).

Dr. Choo was a recipient of the 2018 UTSA College of Business Col. Jean Piccione and Lt. Col. Philip Piccione Endowed Research Award for Tenured Faculty, the IEEE TrustCom 2018 Best Paper Award, the ESORICS 2015 Best Research Paper Award, the 2014 Highly Commended Award by the Australia New Zealand Policing Advisory Agency, Fulbright Scholarship in 2009, 2008 Australia Day Achievement Medallion, and British Computer Society's Wilkes Award in 2008. In 2015, he and his team won the Digital Forensics Research Challenge organized by Germany's University of Erlangen–Nuremberg, Erlangen, Germany. He is also a Fellow of the Australian Computer Society.