IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 19 March 2023, accepted 25 April 2023, date of publication 4 May 2023, date of current version 11 May 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3272899

RESEARCH ARTICLE

A Secure and Cost-Efficient Blockchain Facilitated IoT Software Update Framework

GABRIEL SOLOMON[1], PENG ZHANG[2], RACHAEL BROOKS[1], AND YUHONG LIU[1], (Senior Member, IEEE)

[1]Department of Computer Science and Engineering, Santa Clara University, Santa Clara, CA 95053, USA
[2]College of Electronics and Information Engineering, Shenzhen University, Shenzhen, Guangdong 518060, China

Corresponding author: Gabriel Solomon (gsolomon@scu.edu)

ABSTRACT As resource-constrained Internet-of-Things (IoT) devices become popular targets of various malicious attacks, frequent updates to keep their software up to date are essential to their security. However, state-of-the-art software delivery and payment systems incorporate multiple services in a client-server structure requiring multiple transits of information between client and server, while also creating a wide attack surface. We propose a blockchain-based end-to-end secure software update delivery framework for Internet of Things (IoT) devices, which aims to ensure confidentiality, integrity, availability, efficiency, and audit-ability for verified software delivery, while offloading the cryptographic computation from resource-constrained IoT devices to a decentralized blockchain system. In particular, we leverage Ciphertext-Policy Attribute-Based Encryption (CP-ABE) and design a customized authorization policy to not only ensure that software updates can only be decrypted and installed on authorized IoT devices but also significantly reduce the computational overhead for key generation and key delivery on the manufacturer side. Furthermore, secure and atomic software delivery and payments between IoT devices and the manufacturer are assured through smart contracts. The authenticity of the delivered software is guaranteed by offloading the computation-based signature validation to smart contracts. Compliance audits are satisfied through immutable records on the blockchain's public ledger, and the smart contracts efficiently guarantee the delivery of software updates in exchange for payment. Security analysis and experiments are performed to compare the proposed framework with state-of-the-art studies and validate its effectiveness.

INDEX TERMS Blockchain, Internet of Things, software update, secure software update, computer security, CP-ABE, smart contract.

I. INTRODUCTION

Internet of Things (IoT) devices are devices with sensors and processors that communicate over the Internet to perform specific tasks [1], [2]. Common examples of IoT devices include wearable medical devices that monitor a patient, smart home devices that control home systems, sensors in farming that report temperature and weather conditions, and many more. Due to broad consumer acceptance, in 2020, 749 billion USD was spent worldwide on IoT devices. Spending will surge to over 1,100 billion USD in 2023 [3], [4], [5]. There were 22 billion connected IoT devices in 2018. The number of IoT devices will double to over 50 billion by 2030 [5], [6].

However, due to their prevalent adoptions, while lacking sufficient computing resources for sophisticated security mechanisms, IoT devices can be easily compromised. One of the most well-known attacks is the Mirai botnet attack of 2016 which brought down large portions of the Internet through a DDoS attack waged by IoT devices [7], [8], [9], [10].

One essential protection approach to preventing such attacks is to patch the software of IoT devices frequently to ensure they are up to date. However, malicious attackers can also launch attacks against the software update process itself by, for example, providing manipulated software updates,

The associate editor coordinating the review of this manuscript and approving it for publication was Biju Issac.

or retrieving software updates without payment. An example is an attack in 2016 that targeted the electric power grid in Ukraine disabling power for 30 substations. The outage impacted approximately 230,000 residents by updating firmware on IoT devices that controlled power systems with malicious software [11], [12]. There have also been software repository state attacks against package managers [13] that provide software libraries and components. Therefore, ensuring the confidentiality, integrity, and availability of the software update process is critical.

To ensure confidentiality and integrity, current cryptography-based solutions require that the manufacturer can (1) encrypt their software updates and (2) efficiently generate numerous decryption keys for all of the authorized IoT devices to retrieve the software update files, which results in heavy overhead on the manufacturer side. In addition, the current industry standard for software delivery is achieved through client-server-based architectures that by design create a single point of failure [14]. On the other hand, IoT devices need to perform authentication to ensure that the received software is from the real manufacturer. The authentication process is often computationally heavy and therefore not suitable to be implemented on resource-constrained IoT devices [14]. Furthermore, existing research assumes the manufacturer is honest by default and thus does not provide a guarantee of a valid software delivery transaction. In other words, the payment can occur without software delivery or vice-versa. Last but not least, in compliance-driven industries such as health care, government, energy, and automotive, having proof of software update and installation for auditors is critical to the manufacturer keeping its business license by maintaining compliance with regulations. However, current software update records are kept in mutable databases that can be manipulated intentionally during an attack or unintentionally during normal business operations [15], [16].

The rest of this paper is organized as follows. We outline the contributions in Section II, discuss the related work in Section III, present the threat model and design goals in Section IV and V, respectively. We further present the proposed solution in Section VI, followed by evaluations in Section VII, security analysis in Section VIII, and conclusions in Section IX.

II. CONTRIBUTIONS OF THIS ARTICLE

A. RESEARCH PROBLEM

IoT devices need to receive current software updates to maintain confidentiality, integrity, and availability. Current technologies rely on a client-server-based software distribution with mutable database records that can be changed. Maintaining confidentiality and security for software distribution systems is critical to system adoption.

Some recent research has proposed using blockchain for software distribution but left several challenges unresolved such as efficient key generation, realistic threat conditions, and guaranteed schemes for payment. In addition, some research uses on-chain solution which is not practical for large software updates since blockchain data storage is limited by the block size. The proposed framework addresses these research problems and makes the following contributions.

B. NOVEL CONTRIBUTIONS

To address these challenges, in this work, we propose a novel blockchain-based distributed software update framework with the following major contributions:

- We propose a distributed blockchain-based framework to facilitate resource-constrained IoT devices to outsource heavy computation-based authentications while providing high availability and immutable records for software updates.
- We adopt Ciphertext-Policy Attribute-Based Encryption (CP-ABE) and design a customized authorization policy to provide efficient and scalable key generation for large numbers of authorized IoT devices.
- We design a cost-efficient smart contract that can guarantee atomic payment in exchange for the delivery of software updates.

C. ADVANTAGES OVER EXISTING TECHNOLOGIES

Compared to existing client-server-based software update delivery frameworks, the proposed framework provides three distinct advantages.

1) DECENTRALIZED FRAMEWORK

Current centralized client-server architecture for software update delivery is often vulnerable to single point failures. Through blockchain, the proposed framework creates a distributed framework that provides resilience and high availability. In addition, compared to a regular database that can be altered, blockchain records are immutable, producing a record that can be audited, and not changed.

2) EFFICIENT KEY GENERATION

With existing technologies, the manufacturer needs to generate encryption keys and perform encryption per device and per software update to maintain confidentiality. In addition, the manufacturer needs to communicate with each device separately for notifications. Through CP-ABE the proposed framework significantly reduces the overhead of key generation and communications from $O(n)$ to $O(1)$.

3) PAYMENT WITHOUT THIRD-PARTY INVOLVEMENT

Current technologies use ''webhooks'' which are secondary reverse channel communications to servers sent from third parties that verify and report that payment has occurred. The introduction of a third party requires additional trust assumptions. Through smart contract, the proposed framework ensures atomicity of the payment and delivery without introducing a trusted third party.

III. RELATED WORK

Recently, blockchain-related techniques have been proposed for IoT software updates. Many of these studies focus on some specific aspects of the software update process. For example, some studies focus on leveraging blockchain as a public ledger to ensure non-repudiation. In Huy's research [15], blockchain is adopted as a public ledger to synchronize configurations across IoT devices. Samaniego & Deters [16], [17] propose the use of blockchain to move software-defined IoT virtualized resources from the device to the fog. Some other works focus on the integrity of IoT software updates. In [18], the authors propose Meta-key, which ensures the integrity of software through blockchain-based key exchange and management. Huh et al propose a blockchain system to manage keys for IoT device configuration [15]. Dorri, et al work focuses on smart-home IoT device utilizing a trust based architecture focusing on trust to reduce block validation time [19]. Placho et al's research illustrates the importance of managing software updates for a large number of IoT devices since software updates far outnumber software development version in the automotive industry [20]. He et al. proposed to use a blockchain to validate the integrity of firmware updates for IoT devices [21]. In addition, many studies focus on incentivizing the participation of blockchain nodes. Leiba et al. propose an incentive-driven blockchain framework for software distribution [22]. Arbari and Shajari [23] propose to leverage a Nash equilibrium micropayment mechanism to motivate nodes to participate in the blockchain platform for IoT software updates. Fukuda and Omote's framework mainly focuses on incentives and access control to reduce the computation complexity of IoT devices [24]. Although these studies advanced the adoption of blockchain techniques in the IoT software update scenario from diverse aspects, they do not provide a holistic view of the entire software update process and thus do not handle some critical security issues, such as authenticity, secure delivery, ensured payment, etc.

There is an existing body of research that explores the idea of applying blockchain for the distribution of software [25]. One category of such studies proposes to store the software update on-chain, which may not be practical due to the limited block space on blockchains. For example, Boudguiga et al. [26] propose innocuous nodes to facilitate software update verification. Lee & Lee provide an on-chain solution for updating firmware on IoT devices [27]. Zhao et al. preserves privacy for participants and provides software updates for IoT devices with proof of delivery [28]. Similarly, Yohan & Lo [29], [30] propose an on-chain blockchain framework for firmware updates on IoT devices. Baza et al.'s framework rely on autonomous vehicles to deliver the software updates and adopts a trust-based system for authentication and integrity of the firmware under a weak assumption of vehicle proximity, storage, and bandwidth to deliver software updates [31]. On-chain software updates using blockchain are further analyzed in a proposal to move IETF SUIT (Software Updates for Internet of Things) [32].

Therefore, some other studies propose off-chain solutions by taking advantage of other storage mechanisms. Yohan et al. [33] present an off-chain solution to distribute software updates and has node operations off-loaded to IoT fog gateways. But it lacks authenticity verification and therefore cannot handle malicious attacks injecting false software updates. Pillai et al. [34] design an off-chain blockchain software update solution using Ethereum [35] and InterPlanetary File System (IPFS) [36] for digital goods. This solution, however, requires the creation and deployment of a smart contract for each update, which can lead to significant overhead and thus does not scale for a large number of IoT devices.

More importantly, most existing studies, regardless of on-chain or off-chain solutions, control the authorized access to the software/firmware updates while ignoring the overhead. As a result, for every IoT device, a new key must be generated and managed by the manufacturer, leading to a linearly increasing overhead when the number of IoT devices is growing, which significantly limits the scalability of the system.

In summary, while the related research covers different aspects of the software update process, there is a lack of a holistic framework that jointly considers confidentiality, integrity, availability, authenticity, the atomic exchange of payment for the value of digital assets, and the non-repudiation of auditing records. In addition, due to the adoption of CP-ABE and the design of a customized authorization policy, the proposed framework in this work solves the scalability issue when a massive amount of IoT devices require software updates.

IV. THREAT MODEL

The proposed framework can defend against a wide variety of malicious attacks. In this section, we discuss our base assumptions, the roles of adversaries, the attack vectors, and describe potential attacks against the proposed framework.

A. ASSUMPTIONS

In this study, we make the following assumptions for the IoT software update scenario.

First, we assume that an IoT owner possesses one or multiple IoT devices and can connect to them through a private and secure network.

Next, the inherent properties of the blockchain are assumed to exist, such as non-repudiation and non-malicious majority of blockchain nodes [37].

Third, softlifting is the piracy process whereby a legitimate single license is purchased and then installed on multiple devices. We assume that the manufacturer requires that only authorized IoT devices can install software updates, which aligns with their licensing agreements to prevent softlifting.

Lastly, data stored in the IPFS cloud storage system is assumed to be secure from attackers.

B. THREAT TYPES

In this work, we mainly consider possible attacks launched by three different parties: (1) a general malicious attacker, (2) a malicious IoT owner, who possesses one or multiple IoT devices, or (3) a malicious manufacturer. In particular, the major types of attacks considered in this study are summarized below.

First, attacks from a general malicious attacker (MA):

- Software update notification attack: In this attack, a malicious attacker aims to interrupt the software notification sent by the manufacturer to IoT owners, causing delays/failures of software updates.
- Confidentiality attack: In this attack, a malicious attacker attempts to retrieve the contents of a software update during its delivery process with the aim of receiving a software update it should not have while also avoiding payment.
- Invalid update attack: In this attack, a malicious attacker attempts to send an invalid/damaged software update to IoT devices aiming to damage the functionality of the IoT device.
- Roll-back attack: In this attack, a malicious attacker aims to send a valid old software with a known pre-existing vulnerability that the attacker can exploit to damage the functionality of the IoT device [38].

Second, attacks from a malicious manufacturer (MM):

- Non-delivery attack: A dishonest manufacturer receives payment while not delivering the required software updates aiming to receive payment without providing a service.
- False update-record attack: A dishonest manufacturer aims to manipulate the record of his/her software update delivery services to avoid taxes, legal responsibilities, etc.

Third, attacks from a malicious owner (MO):

- Payment-free attack: In this attack, a malicious owner of IoT devices attempts to receive a software update without payment.

V. DESIGN GOALS

The design goals of the framework are to prevent attacks in the threat model to reduce the attack surface for software updates through a secure distributed blockchain system that provides:

- Confidentiality, integrity, and authenticity of software updates for authorized IoT devices
- Efficient and scalable key-generation
- High availability
- Guaranteed payment and delivery of software updates
- Immutable software update records for auditing

A. CONFIDENTIALITY, INTEGRITY, AND AUTHENTICITY OF SOFTWARE UPDATES FOR AUTHORIZED DEVICES

This design goal aims to provide end-to-end security for the software update process. This is motivated by the aforementioned software update attacks. In particular, we aim to keep the software update confidential and only available for authorized devices by encrypting the software update via attribute-based encryption algorithms (i.e. more details in Section IV-B). The integrity of the software update is maintained by verification using a SHA-3 cryptographic hash, which ensures that any changes to the software update will be detected [39]. We choose SHA-3 because it is a highly configurable hash algorithm that is resistant to numerous attacks [40]. The framework is designed to provide authenticity by leveraging smart contracts and Elliptic Curve Digital Signature Algorithm (ECDSA) to validate the manufacturer's identity [41]. ECDSA provides a space-saving small key size for signatures that is suitable for IoT devices.

It is standard practice for software updates to be encrypted before delivery to maintain confidentiality. This process is commonly called encryption-at-rest. The encryption process requires a key to be generated per software update per IoT device. This key generation, however, can create a large burden for the manufacturing side, especially when thousands of IoT devices need to be updated frequently. Therefore, it motivates the next design goal.

B. EFFICIENT AND SCALABLE KEY GENERATION

To enable numerous authorized IoT devices to decrypt software updates, the manufacturer has to generate a large number of keys, which will take significant computational resources over time. In addition, the key management and delivery scheme, an essential part of secure software delivery systems, will be very complex for a large amount of IoT devices with diverse properties. Therefore, an efficient and scalable key generation process is critical.

To enable efficient access to a software update from only authorized IoT devices, we propose to adopt attribute-based encryption algorithms in which the decryption of a ciphertext requires the set of attributes of the user key to match the attributes of the ciphertext. One core component is the access tree (i.e., authorization policy), which defines what types of attributes are required for decryption.

In particular, Key-Policy Attribute-Based Encryption (KP-ABE) and Ciphertext-Policy Attribute-Based Encryption (CP-ABE) are the two main attribute-based encryption algorithms. In both cryptosystems, an authorization policy is a set of rules created by the sender that must be satisfied by the receiver's attributes. The major difference is how they generate the ciphertext and decryption keys (users' secret keys). In CP-ABE, the ciphertext is generated based on the access tree, while the decryption key is generated based on the attributes. KP-ABE [42] performs oppositely by generating ciphertext over the attributes and decryption key based on the access tree.

In the IoT software update scenario, the physical attributes of IoT devices are not changing, while the access tree will change for each new software licensing policy change from the manufacturer. Therefore, the proposed framework adopts

CP-ABE, as the decryption keys generated based on device attributes do not need to change frequently. This property can significantly reduce not only the key generation and delivery overhead from the manufacturer side but also the key storage space required on the resource-constrained IoT device side.

C. HIGH AVAILABILITY THROUGH A DISTRIBUTED SOFTWARE UPDATE FRAMEWORK

The sheer amount of IoT devices requires a software update platform to have high availability, which can be a challenging issue for the current client-server model. To improve the resiliency and availability of the software update process, we propose a distributed platform, so that the encrypted software update and the ciphertext of the corresponding key can be delivered with high availability.

In the proposed framework, we propose to leverage blockchain as the basis of the distributed software update delivery platform. Furthermore, due to the high costs and limited storage to store large data on a blockchain, the proposed design seeks to offload the encrypted software updates to the IPFS, a peer-to-peer distributed file storage system, which provides high availability and scalability [36].

D. GUARANTEED PAYMENT AND DELIVERY OF SOFTWARE UPDATES

In addition to a manipulated software update, an attack can be waged against payment. For example, a dishonest IoT owner may try to retrieve the software updates without payment (i.e., a payment-free attack). Or on the other hand, a malicious manufacturer may receive the payment while not delivering the requested software update (i.e., a non-delivery attack). In the current client-server-based structure, information is often processed by RESTful communication, which cannot guarantee the automaticity of transactions [43]. Therefore, to guarantee both parties honestly execute the transaction, we propose to execute a binding contract to ensure atomicity of payment in exchange for delivery of the software update.

To achieve this goal, we leverage a smart contract, which is a program that can be deployed on blockchains to ensure that predefined protocols/actions can only be executed when certain conditions are met. The adoption of a smart contract can guarantee the atomicity of the transactions, meaning that both the actions from the two parties (i.e., payment from the IoT devices and the delivery of software updates from the manufacturers) are successfully completed, or none of them are completed.

Furthermore, the design of the smart contract can significantly influence the cost of transactions. Therefore, we aim to propose a cost-efficient design of a smart contract that can achieve the goal of atomicity while minimizing transaction costs without requiring a trusted intermediary.

E. IMMUTABLE SOFTWARE UPDATE RECORDS FOR AUDITING

The design goal of auditing is motivated by two major needs. (1) Current payment systems (PayPal, Stripe for example) rely on "webhooks" to report auditing information back to the manufacturer that is separate and distinct from the transaction. The webhooks also typically work on a RESTful framework and require transmission and periodic retransmissions to ensure records are updated or received by the server to settle payment balances. (2) The permanence of data for auditing in compliance-driven industries is important for the business to keep its licenses. Though true, software payment systems that store data in databases can be manipulated and changed by anyone with write permissions. In contrast, the blockchain's public ledger is immutable, providing an accurate and permanent record for auditors.

VI. PROPOSED SCHEME

A. PRELIMINARY

The core technologies adopted in the proposed framework are blockchain and CP-ABE. The working mechanisms of these technologies are briefly summarized in this section.

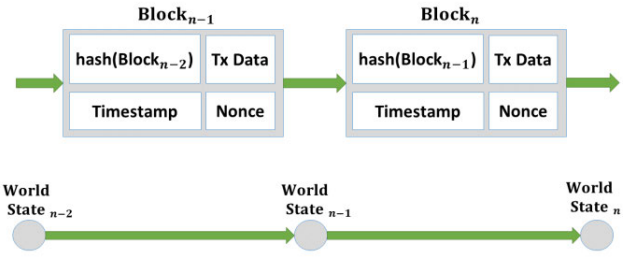


FIGURE 1. Blockchain data structure.

1) BLOCKCHAIN

Established from BitCoin [44], blockchain-related technologies have attracted broad attention from the general public. In this work, we propose to adopt a blockchain platform and design a cost-efficient smart contract to offload the comprehensive cryptographic computations from the resource-constrained IoT devices, ensure atomic payment in exchange for delivery of software updates, and maintain an immutable software update record for auditing [45], [46], [47].

In particular, each block contains a series of ordered transactions, where each transaction represents certain interactions among different parties. Both a manufacturer and an IoT owner can obtain a unique ID and a pair of public/private keys, and therefore, all their actions performed on the blockchain (e.g., delivery of software, payment) will be authenticated and reflected in transactions, which significantly reduces the authentication overhead on the IoT device. In addition, different blocks are linked together by including the cryptographic hash value of the previous block in the next block as seen in Fig. 1. In this way, no prior blocks can be changed without altering all later blocks. This immutability property enables the blockchain to serve as a public ledger

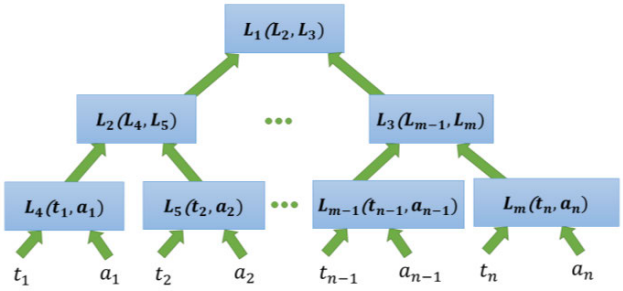


FIGURE 2. CP-ABE authorization policy.

providing an auditable list of software updates. In addition, the distributed nature of the blockchain provides availability for software update encryption keys. The linked list structure is common to many blockchains. Though true, Tangle provides a directed acyclic graph structure for its blockchain that is compatible with the proposed framework [48].

2) CIPHERTEXT-POLICY ATTRIBUTE-BASED ENCRYPTION

Ciphertext-policy Attribute-based Encryption (CP-ABE) is a cryptosystem that encrypts plaintext with a policy such that the attributes that satisfy the policy are required to decrypt the ciphertext [49]. CP-ABE ensures that devices with the correct attributes are singularly able to decrypt and thus install the software update.

In particular, authorization policy is determined by the party generating the ciphertext. As shown in Fig. 2., CP-ABE constructs the authorization policy as an access tree, which is a binary tree of boolean functions defined as follows:

$$\begin{aligned} L_i\left(t_j, a_j\right) &\in \left\{t_j < a_j, t_j > a_j, t_j = a_j\right\} \\ 0 &< i \leq m, \\ a_j &\in \mathbb{Z}, \quad a_j > 0, \\ 0 &< j \leq n \end{aligned} \tag{1}$$

In equation (1), t_j represents an attribute type, a_j represents the attribute value, and $L_i\left(t_j, a_j\right)$ is a boolean function comparing the value of t_j with a_j.

Next, we use an example to explain the authorization policy. For example, the authorization policy might be that the receiver must be over 20 years old and be taller than 5 feet so that he/she can decrypt the ciphertext.

$$authorization\ policy = (age > 20) \wedge (height > 5)$$

where $\wedge$ represents the "and" boolean function.

This means any party that has attributes satisfying the authorization policy can decrypt the message by using its own secret key. Using equation (1), we have

$$\begin{aligned} L_1 &= L_2 \wedge L_3 \\ f_2 &= L_2\left(age, 20\right) = age > 20 \\ L_3\left(height, 5\right) &= height > 5 \end{aligned}$$

where L_2 and L_3 are the ">" (i.e. greater-than boolean operator).

In this study, we will propose a customized CP-ABE access policy to ensure that only authorized IoT devices can install legitimate software updates.

B. SYSTEM OVERVIEW

In this section, we will provide an overview of the proposed framework, and explain different roles and responsibilities as shown in Fig. 3.

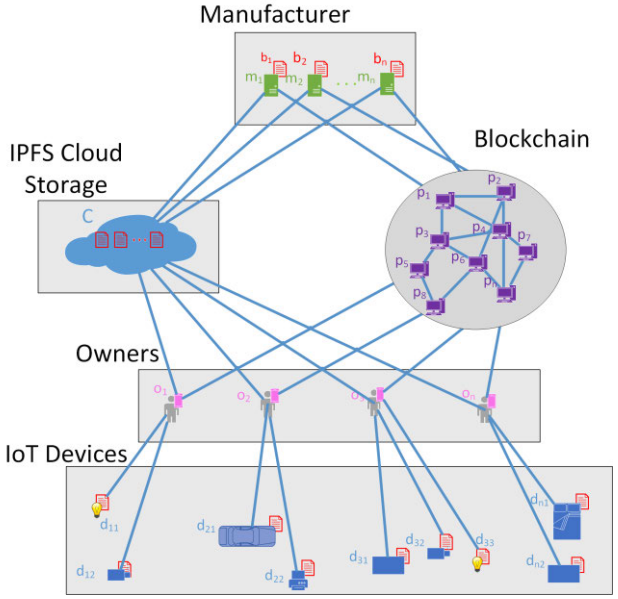


FIGURE 3. Overview of proposed architecture.

The first role is the manufacturer m_i, who produces software updates, posts notifications regarding the availability of new updates, uploads the software update to the cloud data storage [50], and delivers software update keys to IoT devices through smart contracts.

An IoT owner manages a system or network that possesses multiple pieces of IoT devices, for example, a university campus or a smart home network. The owner o_l manages his/her IoT devices, decides if any device needs to install a specific software update, and makes payment to the manufacturer regarding the software update services. The purpose of introducing the IoT owner is to aggregate the interactions between the manufacturer and a set of IoT devices to further reduce the overhead.

The interactions between manufacturers and IoT owners will be facilitated by a distributed blockchain network. We model these interactions (e.g., notification, payment, et cetera.) between the IoT owner and manufacturer as transactions that are stored on the blockchain to maintain an immutable and irrefutable record. For example, the notifications of software updates are sent to the blockchain by the manufacturer. Once the IoT owner decides to make a purchase, he/she can participate in the smart contract

initiated by the manufacturer. The smart contracts are executed by blockchain nodes, which includes verifying the validity of the update, delivering the update keys to the IoT devices, and enforcing the payments. In addition, the final installation transaction provides an immutable record of software installation. This is an especially important feature for manufacturers in regulated industries, which are required to meet industry compliance obligations. The blockchain public ledger records all the transactions and guarantees they are immutable. A copy of the records is stored by each blockchain node p_k in a decentralized way to ensure non-repudiation.

The IPFS Cloud storage is used to store the encrypted software update file (i.e., b_j) because blockchain storage is limited by the block size and storing software updates on blockchains can be very expensive. By adopting IPFS as the off-chain storage, the proposed framework can significantly reduce the overall cost.

C. MAJOR MODULES

As shown in Fig. 4, the proposed framework contains the following major modules: (A) Key generation, (B) notification that a software update exists, (C) encryption and decryption of the software, and (D) distribution and download of the software update. In this section, we give a further analysis of these components of the proposed framework. The notation and terminology used to define each module are explained in Table 1.

1) KEY GENERATION

To reduce the number of device-specific key generations needed for current software update techniques, the proposed framework uses CP-ABE. Instead of per-device key generations, only one key generation is needed for all devices with attributes to satisfy the decryption CP-ABE policy.

The overhead related to using CP-ABE is very low compared to the common technique of generating a unique key for each IoT device. CP-ABE decryption runs in near-linear time relative to the number of attributes as explained in Bethencourt's research [49]. Since in our framework, we only run CP-ABE operations once for a group of g devices as determined by the manufacturer's authorization policy, we reduce the asymptotic time to generate keys from $O(g)$ to $O(1)$. In addition to the reduction in cpu-time to generate keys, we also reduce the space to store and manage keys from $O(g)$ to $O(1)$.

In this work, we propose three key attribute types t_j as {model, serial number, and version}. In particular, the model indicates the product type; and the serial number can be used to uniquely identify each specific IoT device, so that only the device with the specified serial number can perform decryption. The physical attributes of the IoT device are inherent to the device's hardware itself and are defined when the device was manufactured. Last but not least, the version indicates the version number of the software updates. The version attribute provides fine-grained control so that only devices with the correct attributes as described by the authorization policy can access the specific version of the software updates for which they have paid. This provides a further guarantee that a software update cannot be copied and used on another device since the decryption occurs during the installation process. The values of these attributes are determined by the manufacturer by constructing the authorization policy for each piece of software update as shown in Fig. 2.

In addition, the proposed attributes can be easily extended to handle a group of IoT devices with similar attributes. For example, a manufacturer could potentially specify two models of a device with a specific range of serial numbers and version range to update. Only these devices would be able to decrypt the software update. Thereby, the private key is only generated once for a large set of devices. This also

TABLE 1. Terminology and definitions.

Object	Notation	Definition
Manufacturer	$M = \{m_1, m_2, \ldots, m_n\}$ where $m_i \in M, i \in Z^+, n \geq 1$	The company, vendor, or business that manufacture and produce the IoT device. The manufacturers range in size from large to small and may or may not be independent of each other
Software Update	$B = \{b_1, b_2, \ldots, b_n\}$ where $b_j \in B, j \in Z^+, n \geq 1$	Software update payload (often a binary file) that is transferred to the device
Header	h_j is part of b_j	First n-bytes of the software update b_j that contains information on the manufacturer, version, and other meta-deta related to the software release
Blockchain Payload Nodes	$P = \{p_1, p_2, \ldots, p_n\}$ where $p_k \in P, k \in Z^+, n \geq 1$	Payload nodes in the blockchain service update requests and are paid a fee for hosting and serving the payload binary. These nodes detect and verify the software update version. They maintain the block chain public ledger. They are paid a fee for their services.
Transaction	$T_{x_j} where\ j \in Z^+$	Transaction record j in the public ledger of the blockchain specifying a specific software update transaction.
Owner	$O = \{o_1, o_2, \ldots, o_n\}$ where $o_l \in O, l \in Z^+, n \geq 1$	The individual/person who owns and/or manages 1,2, ..., n IoT devices
IoT Device	$D = \{d_{11}, d_{12}, \ldots, d_n$ where $d_{il} \in D, i \in Z^+, l \in Z^+, n \geq 1, m \geq 1$ with $i \in \{manufacture$ and $l \in \{owners\}$	An IoT device ranging from a smart-light bulb, smart-TV, phone, to an IoT device in a vehicle. The set of devices are explicitly heterogeneous, and resource-constrained.
Cloud Data Storage	C	Cloud data storage that holds encrypted binary software update files b_j

has the advantage of applying to single-user license purchase scenarios.

2) NOTIFICATION

The process starts with the manufacturer m_i sending a notification message to the blockchain. The notification message contains version information and a description of the software update [51]. As part of the description, in addition to version numbers, we provide distinct categories of security, bug fixes, features where each field contains a boolean variable for existence and a short text description. This enables the manufacturer to use a template for software update information that clearly explains the purpose of the update for an owner's review and approval process.

The key that is needed to decrypt the software update is encrypted with CP-ABE and signed using ECDSA by the manufacturer. This key becomes part of the update message that is stored on the blockchain by the manufacturer. Once the update message with the key is received by the smart contract, the identity of the manufacturer is verified via the manufacturer's signature, as shown in Fig. 4. Confidentiality is maintained by preventing other parties from accessing the decryption key by encrypting it through CP-ABE. The integrity of the software update is maintained by the SHA-3 hash, which ensures that altering the software update will be detected and prevented by the framework.

By notifying the blockchain, a permanent record of the availability of the update is noted in the public ledger. This prevents attacks targeting notifications and ensures even if the device is temporarily unavailable, the notification will be received once it is available.

In addition, the notification occurring through blockchain provides a notification system that is both distributed and resilient to software update notification attacks. Prevention of notification abuse and notification attacks is an important aspect of software delivery. Since the notification is encrypted and delivered through a distributed, and resilient blockchain framework, the proposed scheme is robust against software update notification attacks that manipulate notification information or carry malicious data.

D. KEY DISTRIBUTION, ENCRYPTION, AND DECRYPTION PROCESS

The asymmetric and symmetric cryptosystems with key generation, encryption, and decryption functions are defined as follows:

$$Asymmetric = \{KeyGen_a(\lambda), E_a(\chi, k), D_a(c, k)$$
$$Symmetric = \{KeyGen_s(\lambda), E_s(\chi, k), D_s(c, k)$$

where $KeyGen, E, D$ represent the key generation, encryption, and decryption algorithms respectively. λ is the security parameter, χ is the message (the software update b_j or other binary data), c is the encrypted message, and k is a key.

We define UID as a universal identifier for a unique software version released by a manufacturer. $UID = \{url \| version\}$, where url is a valid URL, such as "https://www.manufacturer.com/", and $version$ follows the unique version numbering scheme determined by the specific manufacturer [52].

In particular, the key setup distribution, encryption, and decryption steps are as follows:

1) KEY SETUP

The manufacturer m_i runs $KeyGen_a(\lambda)$ to generate a public key PK_{m_i} and private key SK_{m_i} pair. The owner o_l also generates a public key and private key pair. The key pair for m_i is (PK_{m_i}, SK_{m_i}). The key pair for o_l is (PK_{o_l}, SK_{o_l}). The IoT owner and manufacturer are now set up to communicate.

2) ENCRYPTION OF THE PAYLOAD BINARY

The manufacturer m_i runs CP-ABE $Setup_c(\lambda)$ to generate the public parameters PK_c and master key MK_c, and then runs CP-ABE encryption $E_c(PK_c, b_j, A)$, where A is the access policy.

The access policy describes the terms, by which the file can be decrypted. In the proposed scheme, the policy defines a boolean formula containing the following attributes.

- serial number of the device
- model of the device
- version of software

For example, the access policy may target updating a specific range of serial numbers for a set of devices to update a specific version of the software. In this way, only the specific devices with the attributes that satisfy the boolean formula-based authorization policy can decrypt that software update.

3) UPLOADING SOFTWARE UPDATES TO THE CLOUD

The software updates $E_c(PK_c, b_j, A)$ is uploaded to the IPFS cloud C. Storing the software update on the blockchain in an "on-chain" solution can introduce avoidable inefficiencies such as creating multiple blockchain transactions or making a transaction more expensive in gas.

4) UPLOADING OF THE KEY TO BLOCKCHAIN

The manufacturer m_i creates a SHA-3 based [39], [53] cryptographic hash (i.e., h_{Eb_j}) for the encrypted binary update file. A software update message um is defined as $um = \{UID \| h_{Eb_j} \| E_a(k_{b_j}, PK_{o_l})\}$. The um is signed by using ECDSA [41], which provides a secure and space-efficient key size for delivery through the blockchain. The signed um is notated as σ. Both the um and σ are uploaded to the blockchain. The following data is then posted on the blockchain.

- UID - Unique identifier of the update, which is used by p_k as an identifier to compare versions
- h_{Eb_j} - Hash of the encrypted binary, which is stored on the cloud
- $E_a(k_{b_j}, PK_{o_l})$ – the encrypted key which can be used by o_l to decrypt the binary
- σ – signature of the um for delivery to owner o_l

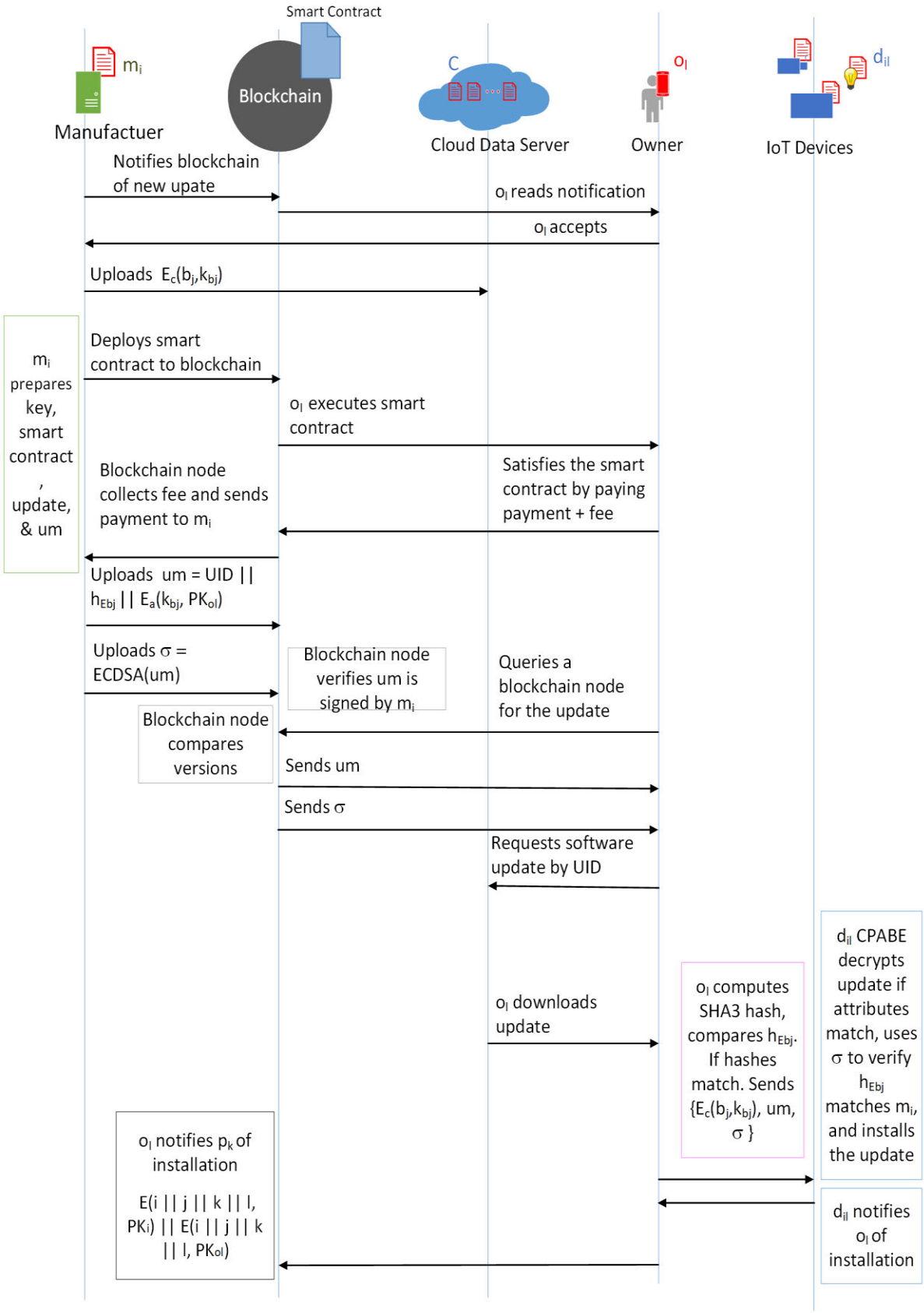


FIGURE 4. Framework process diagram.

E. DISTRIBUTION AND DOWNLOAD OF THE UPDATE

In this section, we will discuss the distribution of the smart contract, delivery of the software update, and its final installation on the target IoT devices as shown in Fig. 4.

The manufacturer creates a smart contract for its software update services and deploys it on the blockchain. Due to its openness, the smart contract can be verified by any node on the blockchain to ensure its correctness. The owner o_l who purchases his/her IoT devices from the manufacturer can participate in the smart contract to interact with the manufacturer.

1) SETUP OF A SOFTWARE UPDATE FOR DELIVERY TO THE OWNER

The smart contract can receive the software update message um, and the signed message σ submitted by the manufacturer, verify the signature, and retrieve the url that stores the encrypted software update. Then, the IoT owner can download the encrypted software updates $E_s\left(b_j, k_{b_j}\right)$ from the cloud data storage C provided by the url.

2) DOWNLOADING THE UPDATE TO THE IOT DEVICE

Once downloaded the encrypted software update $E_s\left(b_j, k_{b_j}\right)$, the owner o_l computes the corresponding hash $h_1{}'$, and verify if it is identical with h_{Eb_j}.

If verified, the IoT device decrypts the software update by performing $b_j \leftarrow D_c\left(PK_c, E_s\left(b_j, k_{b_j}\right), SK_d\right)$ where SK_d is a secret key encrypted with the attributes, including the model, serial number of the device, and version of the software. The attributes are specific to the IoT device and installed in the device's hardware when it is manufactured and are thus considered unforgeable.

3) INSTALLATION OF THE UPDATE

The device then installs the software and upon completion notifies the blockchain that the software has been installed with a message $E_a\left(i \parallel j \parallel k \parallel l \parallel um, PK_i\right) \parallel E_a\left(i \parallel j \parallel k \parallel l \parallel um, PK_{o_l}\right)$ specifying information about who participated in the transaction. This message can be used later by both the manufacturer and the owner for auditing purposes.

F. DESIGN OF THE COST-EFFICIENT BUYER–SELLER SMART CONTRACT

In this section, we discuss the proposed cost-efficient smart contract, which aims to guarantee the delivery of software in exchange for payment.

Typically, when creating a smart contract, the transactions that take place are between two parties. In the proposed buyer-seller smart contract, we design the capability for groups of IoT owners to receive the software update through the same smart contract. Although each IoT owner still needs to have a transaction with the manufacturer such that no party acts before a previous step has been completed, performing the transactions by group instead of individually by owner or device reduces the overall transaction cost to the manufacturer. As a result, the proposed solution conserves gas and increases the performance of the smart contract.

Specifically, while the assignment of IoT owners to the group, where they will receive their data should be done off the smart contract, the smart contract has a central array used to cache the CP-ABE-encrypted keys for the software update and the signature from the manufacturer. The owners of the devices can see which element of the central array holds their software information. Furthermore, there is a hash of the transaction used by the manufacturer to put that data on the smart contract. When a miss occurs in the cache, the blockchain's immutable ledger records all transactions and provides the software update information. Once the owner accesses the data and updates his/her devices, the device sends proof of update to the blockchain as a final confirmation for auditing purposes. By putting the CP-ABE encrypted keys on the blockchain, the framework is designed to make the common case fast. It is less performant to individually encrypt keys for every device which is the traditional and current practice. In addition, using the smart contract combined with CP-ABE encryption allows for these transactions to be confidential, available, and auditable.

Below we outline the high-level algorithms for the Buyer-Seller smart contract. In the implementation, the Buyer-Seller smart contract is organized into a class structure with two primary algorithms of read and write as described below.

Algorithm 1 Buyer-Seller Write

```
1: G : d_i → g_j                                  > initialization
2: procedure Write(g_j, c_k, p_l, f_m)
3:     A(g_j) → (c_k, p_l, f_m)
```

Algorithm 2 Buyer-Seller Read

```
1: G : d_i → g_j                                  > initialization
2: procedure Read(d_i, m_n, e)
3:     g_j = G(d_i)
4:     if g_j ∄ A then
5:         A(g_j) ← readKeys(m_n)
6:     if e = P(A(g_j)) +F(A(g_j)
7:         (p_l, f_m) ↔ C(A(g_j))
8:         pay f_m
9:         collect p_l
```

In Algorithm 1 and 2, G is the preassigned mapping of groups to device identifiers. A is the array used to cache CP-ABE keys by group, d_i is a device, g_j is a group, c_k is a cpabe key, p_l is a price, and f_m a fee predetermined by the manufacturer. We define C, P, and F as utility functions that return a specific element of the tuple. Specifically, $C\left(c_k, p_l, f_m\right)$ returns the first element c_k. $P\left(c_k, p_l, f_m\right)$ returns p_l and $F\left(c_k, p_l, f_m\right)$ returns f_m. Furthermore, $readKeys()$ reads all transactions which allows the manufacturer to update the array A when a cache miss occurs.

VII. PERFORMANCE EVALUATION

In this section, we describe the proposed scheme on the Ethereum blockchain and evaluate gas usage caused by the proposed scheme. Gas is Ethereum that is consumed and paid by nodes to execute a smart contract and/or make any transactions on the Ethereum blockchain. Gas is a metric used to measure both CPU and memory usage for smart contracts that run on the Ethereum Virtual Machine (EVM). Thus, it is the primary means of evaluating the performance of Ethereum blockchain systems. The proposed framework leverages the Ethereum blockchain [35] for implementation because it has smart contract functionality implemented in a Turing-complete programming language (Solidity) [54].

The simulations ran under Windows 10 in Ubuntu-based Docker containers designed to simulate IoT device configurations. For physical hardware tests, we ran on Raspberry pi3 Model B Broadcom BCM2711, Quad-core Cortex-A72 2GB RAM, and Raspberry pi4 Raspberry Pi 3 (RPi3) Model B Quad-Core 1.2 GHz 4 GB RAM Quad Core devices.

Since gas usage measures both CPU and memory space usage on the Ethereum blockchain, we analyze the framework in the context of the amount of gas used and its performance and scalability [55].

A. SIMULATION PARAMETERS

The input parameters for the simulation results are as follows:

- 1 manufacturer
- 5 software products
- 6 hardware models of IoT devices
- 5,000 IoT devices

From the simulation, Fig 5, Fig 6, and Fig 7, the histograms describe the overall usage of gas for the proposed smart contract to analyze performance resulting in 45,271 smart contract transactions. As seen in these figures, the framework demonstrates both consistent and efficient gas usage due to how the proposed smart contract is designed to group transactions.

B. RESULT ANALYSIS

For the notifications that are posted on the blockchain, gas is used to send the notification that a new update exists, to send the update message, and to send the manufacturer's digital signature of the update message on the blockchain. Notably, using Ethereum-based smart contracts, the contract creation and data writing operations cost gas while reading data from the blockchain does not. The simulations showed stable notification gas usage under 50,000 wei, as shown in Fig. 5. While 50,000 wei may seem like a lot, with 1 ether being $1{\cdot}10^{18}$ wei. At the time of writing this paper, the current market value of 1 ether is $3027. Thus, the total cost for all notification transactions for 5,000 devices for all products and all models is much less than one cent ($\$1.5 \times 10^{-10}$), thereby making the proposed framework practical for manufacturers with millions of IoT devices.

Similarly, the experiment results also demonstrate efficient usage of gas for the update message at under 30,000 wei, as shown in Fig. 6. In addition, the simulation shows efficient usage of gas for the signature message signature under 30,000 wei, as shown in Fig. 7.

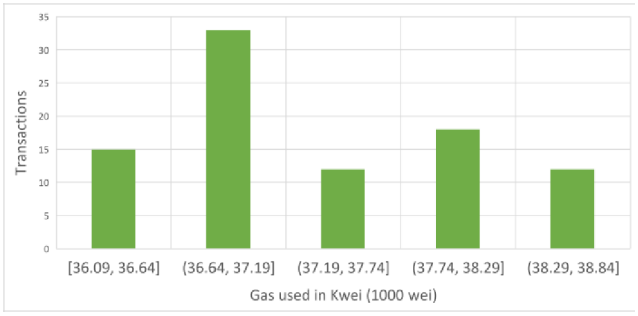


FIGURE 5. Notification gas usage.

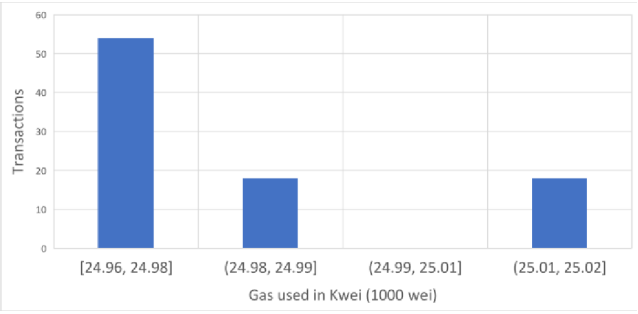


FIGURE 6. Update message gas usage.

C. COMPARISON

This section compares the proposed framework with existing frameworks as seen in Table 2. The proposed framework provides several particularly important advantages when compared to previous research.

Specifically, several researchers have used on-chain storage of the software update. On-chain solutions are limited by the block size of the blockchain technology [26], [27]. Even with variable block-size technologies like Etherium, storing large software updates on the blockchain costs additional fees in gas to store large software updates. The block size limit or/and the additional cost are avoided with off-chain solutions. Prior research focuses on the notification and distribution of the software key while not improving the availability of the software update binary file itself [26], [31], [33]. In the proposed framework, we leverage IPFS to provide a highly available and distributed encrypted software update binary file. The proposed framework offloads several computationally expensive operations to the blockchain platform to reduce computational complexity requirements for IoT devices. As seen in Table 2, previous research has overburdened the IoT devices with these operations creating high IoT computational complexity [26], [27], [31]. The proposed

TABLE 2. Comparison.

	Lee & Lee [28]	Boudguiga [27]	Yohan and Lo [34]	Baza [32]	Proposed Framework
On or Off-Chain Software Update	On-chain	On-chain	Off-chain	Off-chain	Off-Chain
Software Update File Availability	High	Low	Low	Low	High
IoT Computational Complexity	High	High	Low	High	Low
Smart contract	No	No	No	Yes	Yes
Owner-Controlled	No	No	No	No	Yes
Support of Multiple Manufacturers	Single	Multiple	Multiple	Multiple	Multiple
Key Generation Complexity	O(n)	O(n)	O(n)	O(1)	O(1)

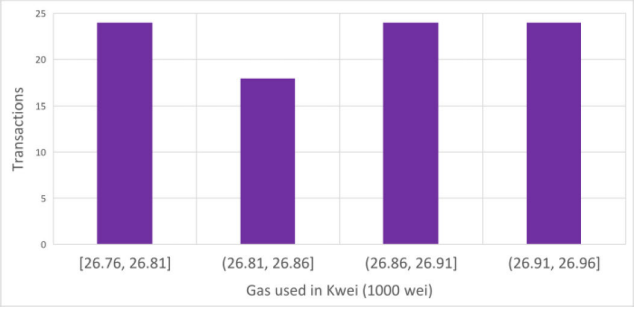


FIGURE 7. Update message signature gas usage.

framework takes advantage of CP-ABE to provide $O(1)$ key generation complexity for the manufacturers. In addition, previous analysis shows that CP-ABE performs well on constrained IoT devices when the authorization policy is dependent on three attributes only and the data being CP-ABE-encrypted is small as we use CP-ABE to encrypt the key, not the software update itself [56].

D. SCALABILITY VERSUS SECURITY TRADE-OFF

Within the proposed framework for software delivery, there are trade-offs between scalability and security. First, the adoption of blockchain and smart contracts ensures the authenticity and integrity of the software update, as well as the atomic software delivery and payments between IoT devices and the manufacturer. Although blockchain techniques are often criticized for its scalability and efficiency, many recent studies have shown significant improvement of the scalability and efficiency through alternative consensus mechanisms [57], [58], and side-chains [59] or layer 2 solutions [60], [61]. Therefore, the scalability of blockchain itself is not the focus of this proposed study. Instead, we mainly focus on the scalability comparison between the proposed framework and the traditional client-server framework.

Table 4 illustrates the scalability of the proposed framework compared to the traditional client-server model by asymptotically evaluating n IoT devices from the manufacturer's viewpoint. As shown in Table 4, due to the adoption of CP-ABE, the manufacturer only needs to encrypt the software for each authorization policy, instead of encrypting it for each device. From the security aspect, it requires a secure design of the CP-ABE authorization policy, so that only the IoT devices with a completed payment can satisfy the authorization policy. To address this issue, we add the devices' serial number as one important attribute to avoid softlift attacks, an attack through which a paid device can share the encrypted software update with another device without payment. From the scalability aspect, since the number of authorization policy terms is a small constant value (i.e., 4 attributes in the proposed system), the encryption costs at the manufacture end remain constant when the number of IoT devices n grows. This significantly improves the scalability of the proposed system.

To notify the client, the manufacturer only needs to post one single transaction (i.e., $O(1)$) on the blockchain, instead of sending notifications to each of the n IoT devices, which significantly reduces the communication overhead at the manufacturer end.

In the conventional client-server model, the server must send the decryption key to each client separately through a secured channel. In the proposed blockchain based framework, the decryption key is sent in exchange for payment by the smart contract. This exchange is accomplished via the smart contract and is publicly available. This public nature emphasizes the importance confidentiality of the transaction to be maintained. As the proposed framework adopts CP-ABE to encrypt the key, the confidentiality is ensured as long as the CP-ABE scheme is not compromised as shown in the Security Analysis section. In addition, the framework uses the blockchain to deliver the key to each device in a distributed manner, which provides additional reliability. The adoption of blockchain may introduce additional risks if the majority blockchain nodes are dishonest. But as discussed

TABLE 3. Safeguards against threat model attacks.

Attack	Adversary	CP-ABE	ECDSA	SHA3	Smart Contract	Blockchain	IPFS
Notification Attack	Malicious Attacker (MA)					X	
Confidentiality Attack	Malicious Attacker (MA)	X					
Invalid Update Attack	Malicious Attacker (MA)		X	X			
Roll-back Attack	Malicious Attacker (MA)		X		X	X	
Non-delivery Attack	Malicious Manufacturer (MM)				X		X
False Update Record Attack	Malicious Manufacturer (MM)					X	
Payment-free Attack	Malicious Owner (MO)		X				

TABLE 4. Scalability from manufacturer's viewpoint.

Process	Client-Server Framework	Proposed Framework
Manufacturer encrypts software update	$O(n)$	$O(1)$
Manufacturer notifies clients	$O(n)$	$O(1)$
Manufacturer sends decryption-key	$O(n)$	$O(1)$
Manufacturer sends software update to client	$O(n)$	$O(1)$

in the Security Analysis section, we show compromising the blockchain is exponentially difficult.

Last but not least, in the proposed framework, the manufacturer only needs to send the encrypted software update to IPFS once, rather than sending it to each IoT device, which greatly improves the scalability. Since the software update is stored and maintained by a third party (i.e., the IPFS), there are additional confidentiality and integrity risks involved if the IPFS is compromised. However, as a distributed cloud storage, it is very difficult for the IPFS to be completely compromised. In addition, the manufacturer only uploads the encrypted software update to IPFS, and the hash value of the encrypted software is posted on the blockchain. As a result, we can assume the confidentiality is ensured if the encryption scheme is not compromised. In addition, as long as the blockchain record is not manipulated, by checking the hash value of the encrypted software, we can ensure the integrity is not compromised.

VIII. SECURITY ANALYSIS

In this section, we define the cryptographic lemmas needed to support proof of the framework's security. Then, we define theorems and proofs of the safeguards against the specific threats defined in the threat model which was provided in Section V. Table 3 organizes and explains which safeguards protect against various threats.

A. SECURITY FOUNDATIONS

The security of the framework is founded on the cryptographic security of CP-ABE, hash, and properties of the blockchain. Herein we define the security foundations of the framework as lemmas to support the proofs of the security of the framework.

Lemma 1: Given a blockchain with n blocks, the probability P for malicious nodes to revert a transaction by 'catching up' is

$$P = 1 - \sum_{k=0}^{n} \frac{\lambda^k e^{-\lambda}}{k!}(1 - (\frac{q}{p_h})^{(n-k)}) \tag{2}$$

where p_h is the probability that an honest node finds the next block, $q = 1 - p_h$ is the probability that a malicious node finds the next block. Therefore, as n grows, the probability P of malicious nodes reverting a transaction decreases exponentially [44].

Lemma 2: CP-ABE cryptosystem for device d_{ij} with the attributes $\{a_1, a_2, \ldots, a_k, a_{k+1}, \ldots, a_m\}$ *where* $m \in \mathbb{Z}$, m is the message, $E_c(m)$ encrypts m, and, D_c decrypts such that $D_c(E_c(m)) = m$. An IoT device with an attribute a_k that does not satisfy authorization policy P is unable to decrypt an encrypted message $E_c(m)$. The CP-ABE cryptosystem ensures the security so that when the authorization policy key is not satisfied the ciphertext cannot be decrypted [49].

Lemma 3: We assume a random oracle model. Within the random oracle model, a hash H will deterministically produce a random value h' for a given input, $H(x_0) = h'$ [62].

Lemma 4: We define a function $G(\lambda)$ that generates a key pair of $\{sk, pk\}$ where sk is a private key, pk is a public key, and λ is an initialization parameter. Given a message m, there exists a function $E(m, sk)$ that produces a signature σ. $V(\sigma, pk)$ produces true if and only if m is signed by the private key sk. This cryptographic construct is formally called a digital signature [63]

B. MALICIOUS ATTACKER (MA) SAFEGUARDS

Theorem 1: A Malicious Attacker (MA) attempts to prevent an honest manufacturer from notifying devices of a new

software update. The MA notification attack is exponentially difficult to succeed.

Proof 1: The manufacturer commits notification transaction $T_\propto$ on the blockchain such that $1 \leq \propto \leq t$ where t is the number of transactions. IoT device owner can read any node on the blockchain to receive the notification. To prevent the notification, the MA attempts to produce a separate chain long enough to invalidate $T_\propto$. By Lemma 1, the ability to revert the notification $T_\propto$ decreases exponentially. Thereby, the ability to successfully prevent the software update notification increases exponentially as the length n of the blockchain grows. All IoT devices will be able to read the notification since the blockchain provides availability through its distributed nodes.

Theorem 2: If we assume a MA can become a man-in-the-middle (MITM) and download a software, the MA is not able to decrypt the software update. The confidentiality of the software update is maintained.

Proof 2: Given the attributes $\{a_1, a_2, \ldots, a_j, \}$ *where* $j \in \mathbb{Z}$ that satisfy the customized CP-ABE authorization policy. A MA with an attribute $a_\propto \in \{a_1, a_2, \ldots, a_j, \}$ such that $a_\propto$ does not satisfy the authorization policy. MA is unable to decrypt a software update due to Lemma 2. Confidentiality is further secured since MA does not have the private key without performing exchange of the software update through the Buyer-Seller smart contract. Thus, software update confidentiality is maintained.

Theorem 3: Given a software update message um, it is impossible for a MA to successfully execute an invalid update attack.

Proof 3: MA attempts to provide a damaged software update message um_{ma} to a device with a forged signature σ_{ma}. The hash of um_f is compared to the hash of the valid um given in the software update notification. Due to Lemma 3, $H(um_{ma}) \neq H(um)$, $H(um_{ma})$ *is* rejected. In the proposed framework as seen in Table 3 and Figure 4, this is implemented with a SHA3 hash. In addition, the device will run $V(\sigma_{ma}, pk)$ from Lemma 4 using ECDSA and detect the inauthenticity of the message.

Theorem 4: Given a software update, a MA should not be able to perform a rollback attack in which a software update is reverted to a previous insecure version of the software update.

Proof 4: The proof for the rollback attack safeguard is a corollary of Proof 1. By Proof 1, the software update notification is immutable. Thereby, the defense against the attack is accomplished by the owner reading the software version number, um, and $H(um)$ from the most-recent notification. With this information the owner can compare the version numbers to detect, and reject the previous insecure version of the software update.

C. MALICIOUS MANUFACTURER SAFEGUARDS

Theorem 5: A malicious manufacturer (MM) must not complete a non-delivery attack whereby the MM receives payment without delivering the software update.

Proof 5: A non-delivery attack attempt by the malicious manufacturer (MM) is prevented by the proposed smart contract and the properties of IPFS. The execution of the smart contract ensures that a valid exchange is executed in an atomic way, so that neither the manufacturer nor the IoT owner can take advantage of the other party. IPFS provides distributed and immutable storage of the software update. In addition, when the MA attempts to receive payment, the unique identifier of the software update in IPFS is sent to the IoT owner to download the software update. This unique identifier (UID) of IPFS contains a hash of the data which guarantees that data has not been manipulated. Availability is guaranteed because IPFS is a distributed storage system [36].

Theorem 6: It is exponentially difficult for a MM to perform a false update records attack aimed to create incorrect transaction records.

Proof 6: As a corollary of Proof 1, when a MM attempts a false update record attack, it aims to insert a forged transaction into the blockchain. Due to Lemma 1, such manipulations of records are exponentially difficult to achieve.

D. MALICIOUS OWNER SECURITY SAFEGUARDS

Theorem 7: A malicious owner (MO) is not able to conduct a payment free attack in which they receive a software update without payment.

Proof 7: The smart contract protects against the malicious owner's (MO) payment-free attacks via an atomic transaction guaranteed by the blockchain smart contract for payment and software key delivery. The execution of the smart contract is distributed, and the resulting transactions are recorded on the blockchain in an immutable way that guarantees payment.

Through these safeguards, the framework successfully provides confidentiality, integrity, availability, and authenticity against threats, and achieves the designed security goals.

IX. CONCLUSION AND FUTURE WORK

In summary, in this work, we propose a distributed blockchain-based framework to facilitate resource-constrained IoT devices to perform secure and efficient software updates while providing high-availability and immutable records for software updates. An authorization policy is designed for CP-ABE to ensure that large numbers of authorized IoT devices can efficiently retrieve the plaintext of software updates. A cost-efficient smart contract is designed to guarantee atomic payment in exchange for the delivery of software updates in an efficient way.

We also are currently researching similar problems in the area of autonomous vehicles which hold opportunity for our framework to address challenges unique to autonomous vehicles research [64], [65], [66].

In future work, the proposed framework can be further applied in other application domains, such as software supply chain pipelines or intelligent vehicle software updates over-the-air (OTA). Integration of smart contract and CP-ABE may lead to atomic safeguards and efficient key generation while also protecting against software piracy techniques from a

malicious owner such as softlifting where the owner attempts to install the same license on multiple valid devices they own.

REFERENCES

[1] S. Poslad, "Ubiquitous computing: Basics and vision," in *Ubiquitous Computing: Smart Devices, Environments and Interactions*. Hoboken, NJ, USA: Wiley, 2011, pp. 1–40.

[2] F. Wortmann and K. Fluchter, "Internet of Things," *Bus. Inf. Syst. Eng.*, vol. 57, no. 3, pp. 221–224, 2015.

[3] (Jun. 2019). *Global Internet of Things (IoT) Market Size and Forecast To 2026*. [Online]. Available: https://www.verifiedmarketresearch.com/product/global-internet-of-things-iot-market-size-and-forecast-to-2026/

[4] (Dec. 2020). *Global Internet of Things (IoT) Market By Software Solution, Report ID 6403*. [Online]. Available: https://www.verifiedmarketresearch.com/product/global-internet-of-things-iot-market-size-and-forecast-to-2026/

[5] (Jan. 2020). *Internet of Things (IoT) in the US*. [Online]. Available: https://www-statista-com.libproxy.scu.edu/study/61733/internet-of-things-iot-in-the-us/

[6] (Jan. 2020). *Size of the Internet of Things (IoT) in Retail Market in the United States From 2014 to 2025*. [Online]. Available: https://www-statista-com.libproxy.scu.edu/statistics/688756/iot-in-retail-market-in-the-us/

[7] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other Botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[8] L. H. Newman. (2016). *The Botnet That Broke the Internet Isn't Going Away*. [Online]. Available: https://www.wired.com/2016/12/botnet-broke-internet-isnt-going-away/

[9] C. Zhang and R. Green, "Communication security in Internet of Thing: Preventive measure and avoid DDoS attack over IoT network," in *Proc. 18th Symp. Commun. Netw.*, 2015, pp. 8–15.

[10] Newman. (Oct. 2016). *What We Know About Friday's Massive East Coast Internet Outage*. [Online]. Available: https://www.wired.com/2016/10/internet-outage-ddos-dns-dyn/

[11] (Mar. 2016). *Inside the Cunning, Unprecedented Hack of Ukraine's Power Grid*. [Online]. Available: https://www.wired.com/2016/03/inside-cunning-unprecedented-hack-ukraines-power-grid/

[12] (Mar. 2016). *Analysis of the Cyber Attack on the Ukrainian Power Grid: Defense Use Case*. [Online]. Available: https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2016/05/20081514/E-ISAC_SANS_Ukraine_DUC_5.pdf

[13] J. Cappos, J. Samuel, S. Baker, and J. H. Hartman, "A look in the mirror: Attacks on package managers," in *Proc. 15th ACM Conf. Comput. Commun. Secur.*, New York, NY, USA, Oct. 2008, pp. 565–574.

[14] E. Alsaadi and A. Tubaishat, "Internet of Things: Features, challenges, and vulnerabilities," *Int. J. Adv. Comput. Sci. Inf. Technol.*, vol. 4, no. 1, pp. 1–13, 2015.

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

[16] M. Samaniego and R. Deters, "Blockchain as a service for IoT," in *Proc. IEEE Int. Conf. Internet Things (iThings) IEEE Green Comput. Commun. (GreenCom) IEEE Cyber, Phys. Social Comput. (CPSCom) IEEE Smart Data (SmartData)*, 2016, pp. 433–436.

[17] M. Samaniego and R. Deters, "Using blockchain to push software-defined IoT components onto edge hosts," in *Proc. Int. Conf. Big Data Adv. Wireless Technol.*, Nov. 2016, pp. 110–119.

[18] D. Li, R. Du, Y. Fu, and M. H. Au, "Meta-key: A secure data-sharing protocol under blockchain-based decentralized storage architecture," *IEEE Netw. Lett.*, vol. 1, no. 1, pp. 30–33, Mar. 2019.

[19] A. Dorri, S. S. Kanhere, and R. Jurdak, "Towards an optimized BlockChain for IoT," in *Proc. 2nd Int. Conf. Internet-Things Design Implement.*, Apr. 2017, pp. 173–178.

[20] T. Placho, C. Schmittner, A. Bonitz, and O. Wana, "Management of automotive software updates," *Microprocessors Microsyst.*, vol. 78, Oct. 2020, Art. no. 103257.

[21] X. He, S. Alqahtani, R. Gamble, and M. Papa, "Securing over-the-air IoT firmware updates using blockchain," in *Proc. Int. Conf. Omni-Layer Intell. Syst.*, New York, NY, USA, May 2019, pp. 1–4.

[22] O. Leiba, Y. Yitzchak, R. Bitton, A. Nadler, and A. Shabtai, "Incentivized delivery network of IoT software updates based on trustless proof-of-distribution," in *Proc. IEEE Eur. Symp. Secur. Privacy Workshops*, Apr. 2018, pp. 1–18.

[23] M. S. Arbabi and M. Shajari, "Decentralized and secure delivery network of IoT update files based on Ethereum smart contracts and blockchain technology," in *Proc. 29th Annu. Int. Conf. Comput. Sci. Softw. Eng.*, Toronto, ON, Canada, 2019, pp. 110–119.

[24] T. Fukuda and K. Omote, "Efficient blockchain-based IoT firmware update considering distribution incentives," in *Proc. IEEE Conf. Dependable Secure Comput. (DSC)*, Jan. 2021, pp. 1–8.

[25] X. Xu, C. Pautasso, L. Zhu, V. Gramoli, A. Ponomarev, A. B. Tran, and S. Chen, "The blockchain as a software connector," in *Proc. 13th Work. IEEE/IFIP Conf. Softw. Archit. (WICSA)*, Apr. 2016, pp. 1–6.

[26] A. Boudguiga, N. Bouzerna, L. Granboulan, A. Olivereau, F. Quesnel, A. Roger, and R. Sirdey, "Towards better availability and accountability for IoT updates by means of a blockchain," in *Proc. IEEE Eur. Symp. Secur. Privacy Workshops*, Apr. 2017, pp. 50–58.

[27] B. Lee and H.-J. Lee, "Blockchain-based secure firmware update for embedded devices in an Internet of Things environment," *J. Supercomput.*, vol. 73, no. 3, pp. 1152–1167, Sep. 2016.

[28] Y. Zhao, Y. Liu, Y. Yu, and Y. Li, "Blockchain based privacy-preserving software updates with proof-of-delivery for Internet of Things," 2019, *arXiv:1902.03712*.

[29] A. Yohan and N.-W. Lo, "FOTB: A secure blockchain-based firmware update framework for IoT environment," *Int. J. Inf. Secur.*, vol. 19, no. 3, pp. 257–278, Jun. 2020.

[30] A. Yohan and N.-W. Lo, "An over-the-blockchain firmware update framework for IoT devices," in *Proc. IEEE Conf. Dependable Secure Comput. (DSC)*, Dec. 2018, pp. 1–4.

[31] M. Baza, M. Nabil, N. Lasla, K. Fidan, M. Mahmoud, and M. Abdallah, "Blockchain-based firmware update scheme tailored for autonomous vehicles," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2019, pp. 1–7.

[32] S. Choi and J.-H. Lee, "Blockchain-based distributed firmware update architecture for IoT devices," *IEEE Access*, vol. 8, pp. 37518–37525, 2020.

[33] A. Yohan, N.-W. Lo, and S. Achawapong, "Blockchain-based firmware update framework for Internet-of-Things environment," in *Proc. Int. Conf. Inf. Knowl. Eng. (IKE)*, 2018, pp. 1–13.

[34] A. Pillai, M. Sindhu, and K. V. Lakshmy, "Securing firmware in Internet of Things using blockchain," in *Proc. 5th Int. Conf. Adv. Comput. Commun. Syst. (ICACCS)*, Mar. 2019, pp. 329–334.

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

[36] J. Benet, "IPFS—Content addressed, versioned, P2P file system," 2014, *arXiv:1407.3561*.

[37] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," *Commun. ACM*, vol. 61, no. 7, pp. 95–102, Jul. 2018.

[38] M. Nawir, A. Amir, N. Yaakob, and O. B. Lynn, "Internet of Things (IoT): Taxonomy of security attacks," in *Proc. 3rd Int. Conf. Electron. Design (ICED)*, Aug. 2016, pp. 321–326.

[39] M. J. Dworkin, "SHA-3 standard: Permutation-based hash and extendable-output functions," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. FIPS 202, 2015.

[40] I. F. Alshaikhli, M. A. Alahmad, and K. Munthir, "Comparison and analysis study of SHA-3 finalists," in *Proc. Int. Conf. Adv. Comput. Sci. Appl. Technol. (ACSAT)*, Nov. 2012, pp. 366–371.

[41] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ECDSA)," *Int. J. Inf. Secur.*, vol. 1, no. 1, pp. 36–63, Aug. 2001.

[42] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, 2006, pp. 89–98.

[43] L. Richardson and S. Ruby, *Restful Web Services*. Newton, MA, USA: O'Reilly Media, Inc., 2008.

[44] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Bus. Rev.*, vol. 31, p. 21260, Oct. 2008.

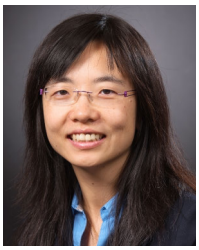
[45] X. Chen, S. Xu, T. Qin, Y. Cui, S. Gao, and W. Kong, "AQ–ABS: Anti-quantum attribute-based signature for EMRs sharing with blockchain," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2022, pp. 1176–1181.

[46] T. Alladi, V. Chamola, N. Sahu, V. Venkatesh, A. Goyal, and M. Guizani, "A comprehensive survey on the applications of blockchain for securing vehicular networks," *IEEE Commun. Surveys Tuts.*, vol. 24, no. 2, pp. 1212–1239, 2nd Quart., 2022.
[47] 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*, Mar. 2017, pp. 618–623.
[48] S. Popov, "The tangle," *White Paper*, vol. 1, p. 30, Jan. 2018.
[49] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2007, pp. 1–15.
[50] K. Gai, J. Guo, L. Zhu, and S. Yu, "Blockchain meets cloud computing: A survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 2009–2030, 3rd Quart., 2020.
[51] M. Fagan, M. M. H. Khan, and R. Buck, "A study of users' experiences and beliefs about software update messages," *Comput. Hum. Behav.*, vol. 51, pp. 504–519, Oct. 2015.
[52] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform resource identifiers (URI): Generic syntax," IETF, Tech. Rep., 1998.
[53] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "KECCAK specifications," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. NISTIR 7764, 2011.
[54] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, pp. 1–10, Sep. 1997.
[55] N. Grech, M. Kong, A. Jurisevic, L. Brent, B. Scholz, and Y. Smaragdakis, "MadMax: Surviving out-of-gas conditions in Ethereum smart contracts," *Proc. ACM Program. Lang.*, vol. 2, pp. 1–27, Oct. 2018.
[56] M. B. Taha, C. Talhi, and H. Ould-Slimane, "Performance evaluation of CP-ABE schemes under constrained devices," *Proc. Comput. Sci.*, vol. 155, pp. 425–432, Jan. 2019.
[57] F. Saleh, "Blockchain without waste: Proof-of-stake," *Rev. Financial Stud.*, vol. 34, no. 3, pp. 1156–1190, 2021.
[58] Y. Wu, P. Song, and F. Wang, "Hybrid consensus algorithm optimization: A mathematical method based on POS and PBFT and its application in blockchain," *Math. Problems Eng.*, vol. 2020, pp. 1–13, Apr. 2020.
[59] A. Singh, K. Click, R. M. Parizi, Q. Zhang, A. Dehghantanha, and K.-K.-R. Choo, "Sidechain technologies in blockchain networks: An examination and state-of-the-art review," *J. Netw. Comput. Appl.*, vol. 149, Jan. 2020, Art. no. 102471.
[60] C. Sguanci, R. Spatafora, and A. Mario Vergani, "Layer 2 blockchain scaling: A survey," 2021, *arXiv:2107.10881*.
[61] M. Jourenko, K. Kurazumi, M. Larangeira, and K. Tanaka, "SoK: A taxonomy for layer-2 scalability related protocols for cryptocurrencies," *Cryptol. ePrint Arch.*, vol. 10, pp. 1–19, Jan. 2019.
[62] R. Canetti, O. Goldreich, and S. Halevi, "The random Oracle methodology, revisited," *J. ACM*, vol. 51, no. 4, pp. 557–594, 2004.
[63] J. Chia, J.-J. Chin, and S.-C. Yip, "Digital signature schemes with strong existential unforgeability," *FResearch*, vol. 10, p. 931, Sep. 2021.
[64] S. Xu, X. Chen, Y. He, Y. Cao, and S. Gao, "VMT: Secure VANETs message transmission scheme with encryption and blockchain," in *Proc. Int. Conf. Wireless Algorithms, Syst., Appl.*, 2022, pp. 244–257.
[65] S. Xu, X. Chen, and Y. He, "EVchain: An anonymous blockchain-based system for charging-connected electric vehicles," *Tsinghua Sci. Technol.*, vol. 26, no. 6, pp. 845–856, Dec. 2021.
[66] A. Dorri, M. Steger, S. S. Kanhere, and R. Jurdak, "BlockChain: A distributed solution to automotive security and privacy," *IEEE Commun. Mag.*, vol. 55, no. 12, pp. 119–125, Dec. 2017.

GABRIEL SOLOMON received the Bachelor of Science degree in computer engineering from UCLA and the Master of Science degree in computer science from the Georgia Institute of Technology. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, Santa Clara University. He is also a researcher in cyber security. He has 17 years of industry experience. His current research interest includes blockchain technologies. He is a recipient of the Silicon Valley NAACP Circle of Friends Award.

PENG ZHANG received the Ph.D. degree in signal and information processing from Shenzhen University, China, in 2011. She is currently an Associate Professor with the College of Electronics and Information Engineering, Shenzhen University. She has published more than 30 academic journals and conference papers. Her current research interests include cryptography technology and security in the blockchain, cloud computing, and the IoT.

RACHAEL BROOKS received the B.S. degree in computer science and engineering from Santa Clara University, in 2020, where she is currently pursuing the M.S. degree in computer science and engineering. Her research interests include smart contract development and image animation using deep learning.



YUHONG LIU (Senior Member, IEEE) received the B.S. and M.S. degrees from the Beijing University of Posts and Telecommunications, in 2004 and 2007, respectively, and the Ph.D. degree from The University of Rhode Island, in 2012. She is currently an Associate Professor with the Department of Computer Science and Engineering, Santa Clara University. Her research interests include trustworthy computing on blockchain, the Internet of Things, and online social media. She is also serving as an IEEE Computer Society Distinguished Visitor (2022–2023) and a Distinguished Lecturer for Asia–Pacific Signal and Information Processing Association (APSIPA) (2021–2022).