A New Scalable and Secure Access Control Scheme using Blockchain Technology for IoT

Sivaselvan N, Vivekananda Bhat K, *Senior Member, IEEE,* Muttukrishnan Rajarajan, *Senior Member, IEEE* and Ashok Kumar Das, *Senior Member, IEEE*

***Abstract*—The growth of IoT devices is so rapid that several billions of such devices would be in use in a span of four-year period. Essential security mechanisms need to be put in place to curb several security attacks prevalent in IoT. Access control is an important security mechanism that ensures legitimate and controlled access to critical and limited resources in IoT. The current access control schemes for IoT could not handle burgeoning number of IoT devices, while meeting the necessary level of security. Consequently, in this paper, we propose a new scalable and secure access control scheme for IoT. With blockchain as the root-of-trust, the proposed scheme performs access control for the IoT devices without having the resource-constrained IoT devices to be part of the blockchain network and to possess substantial amount of blockchain data. Blockchain's tamper-proof property makes it an ideal candidate to be chosen as the root-of-trust. The scheme is secure against various security attacks prevalent in IoT. A proof-of-concept implementation for the scheme is developed and deployed in Ethereum Mainnet. The transaction costs of the different operations in the scheme are fairly below USD 3. Furthermore, scalability of the proposed scheme in different scenarios is investigated.**

***Index Terms*—Internet of Things (IoT), Blockchain, Authentication, Access control.**

I. Introduction

Internet of Things (IoT) has turned out to be one of the most prominent paradigms for several applications like smart home, smart health care, smart city, smart grid, smart transportation, smart farming and so on. It is foreseen that, by the year 2025, there would be 75 billion IoT devices [1] spanning many application areas. The other side of IoT is that several IoT environments are experiencing various security threats [2]. Executing the essential security mechanism(s) to maintain the necessary level of security for the increasing number of IoT devices becomes challenging.

Among the different security mechanisms, access control is essential to ensure that only legitimate IoT devices are allowed access to critical and limited resources in the system. Access control schemes that are devised following the centralized design philosophy do not meet the needs of advanced and dynamic IoT scenarios. Realizing the need for a new way of approaching the problem, researchers moved towards developing decentralized access control solutions. Blockchain technology [3] has the potential to provide robust decentralized solutions. Many decentralized access control solutions for IoT using blockchain technology [4–11] have been proposed. However, these blockchain-based access control solutions could not manage rapidly growing number of IoT devices, while maintaining the required security level.

Sivaselvan N is with the Department of Electrical and Electronic Engineering, City University of London, UK & Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal 576104, India (e-mail: sivaselvan.natarajan@city.ac.uk, siva.selvan@manipal.edu).

Vivekananda Bhat K is with the Department of Computer Science and Engineering & Center for Cryptography, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal 576104, India (e-mail: kv.bhat@manipal.edu).

Muttukrishnan Rajarajan is with the Department of Electrical and Electronic Engineering, City University of London, UK (e-mail: r.muttukrishnan@city.ac.uk).

Ashok Kumar Das is with the Center for Security, Theory and Algorithmic Research, International Institute of Information Technology, Hyderabad 500 032, India (e-mail: iitkgp.akdas@gmail.com, ashok.das@iiit.ac.in).

A. Motivation and Contributions

With the rapid growth of IoT devices and various security attacks reported in IoT, there is a critical requirement to enforce the necessary security mechanisms, especially, access control for controlled access to pivotal and limited resources in IoT. The current blockchain-based access control solutions [4–11] that are designed to fulfill the needs of dynamic IoT scenarios could not handle burgeoning number of IoT devices, while maintaining the required security level. In particular, the schemes [4–7] [11] scale reasonably better. But, they could only partially offer security against the attack vectors identified and enumerated in the classical threat models [12, 13] and other attacks prevalent in IoT. The scheme [9] is comparatively secure but scalability is not investigated. On the other hand, the access control technique introduced in [8] is neither scalable nor secure. The scheme [10] is secure. However, scalability is not investigated. Therefore, we made the following contributions in this paper.

- A new scalable and secure access control scheme using blockchain technology for IoT is proposed. With blockchain as the root-of-trust, the proposed scheme carries out access control for the IoT devices without the need for them to be part of the blockchain network and to hold substantial volume of blockchain data. Blockchain's tamper-proof property that ensures data integrity makes it the right candidate to be chosen as the root of trust. In the proposed scheme, any blockchain node can register as a device manager on-demand to handle the rapidly growing number of IoT devices. The proposed scheme utilizes smart contracts to store the necessary information for access control in the blockchain and manage them.
- The proposed scheme is analyzed for the security attacks enumerated in the classical threat models [12, 13] such

as repudiation, information disclosure, impersonation, Denial-of-Service (DoS), and other conventional attacks like traceability, private-key compromise, collusion.
- A proof-of-concept implementation for the proposed scheme is developed and deployed in Ethereum Mainnet to obtain the real transaction costs of different contract operations in the scheme.
- To ascertain the proposed scheme's performance with the increased and diversified workload, scalability of the proposed scheme in four well identified and formulated scenarios is examined.
- The storage overhead for blockchain transactions in the proposed scheme is studied. The scheme's computational and communication overheads are also evaluated.

B. Paper Structure

The rest of the paper is structured as follows: Section II reviews the related work of decentralized access control schemes in IoT. Section III presents the blockchain-based architecture for the proposed scheme. The proposed blockchain-based access control scheme for IoT is discussed thoroughly in Section IV. Section V presents the security analysis of the scheme. The transaction costs of different contract operations in the proposed scheme are assessed by a proof-of-concept implementation in Section VI. Besides, the study of scalability of the proposed scheme is carried out. Also, the storage, computational and communication overhead in the scheme are studied. Furthermore, the closely related existing schemes are compared with the features of the proposed scheme in Section VI. Section VII concludes the paper.

II. Related Work

This section discusses the recent and closely related decentralized blockchain-based access control schemes in IoT. The classical threat models [12, 13] are employed in identifying the security weaknesses in the related work. The attack vectors considered in accordance with these models for the aforesaid purpose are "repudiation", "information disclosure", "impersonation", "DoS", "traceability", "private-key compromise", and "collusion".

"A framework based on blockchain technology to enable secure mutual authentication, so as to enforce fine-grained access control policies for industry 4.0 environment" was proposed in [4]. The framework is usable and scalable. But, the framework is not secure against "traceability", "information disclosure", and "collusion" attacks.

Xu *et al.* [5] devised a "blockchain-based federated access control system based on capability for IoT" considering two IoT domains. The system is scalable. However, the system is vulnerable to "traceability" attacks. Besides, the system is less usable.

Novo [6, 7] introduced a "scalable access control scheme based on blockchain technology for IoT". The scheme is based around smart contracts of blockchain technology. The scheme is usable and scalable. However, it has limitations in security aspect. The scheme is not secure against "repudiation", "information disclosure", "impersonation", "DoS", and "collusion" attacks.

Zhang *et al.* [8] presented a "framework based on smart contracts for decentralized access control in IoT environment". The framework is usable. But, it is vulnerable to "traceability", "information disclosure", "impersonation", "DoS", and "collusion" attacks. Moreover, the framework is less scalable since the number of contracts in the system is equivalent to the number of resource requesting entities.

"A blockchain-based access control protocol for IoT-enabled smart-grid system" was devised in [9]. The protocol is secure against most of the security attacks considered. However, in the protocol, a registration authority generates the identities, public and private keys for the smart meters and service providers. This may result in "private-key compromise" and "key-escrow" attacks. Furthermore, scalability of the protocol is not examined.

We proposed a "blockchain-based scheme for authentication and capability-based access control in IoT" in [10]. The scheme is usable. In the design of this scheme, scalability requirement is not realized and considered. Henceforth, the ability of the scheme to scale with increasing IoT devices is not investigated. Moreover, the scheme's security analysis does not consider sufficient threat models. We addressed these substantial requirements in the present work.

"A blockchain-based access control framework for IoT endpoint" was presented in [11]. The framework is scalable. However, operation compatibility between the IoT network and blockchain network is achieved by integrating blockchain technology into the gateway nodes in the IoT network. This makes the framework less usable. Besides, the scheme is vulnerable to "traceability", "private-key compromise", and "information disclosure" attacks. This is because device authentication is based around the pre-defined secrets embedded into the IoT devices at the time of manufacturing.

We discuss some closely related existing blockchain-based public-key infrastructure (PKI) approaches as we propose a blockchain-based PKI for IoT-device access control in this paper. In [14], the authors devised "An automated, resilient, and transparent public-key infrastructure" called "BlockPKI". Kubilay *et al.* [15] proposed "A new PKI model with certificate transparency based on blockchain" called "CertLedger". In both the approaches, a dedicated group of certificate authorities that belongs to one organization issues and revokes certificates. Hence, the approaches are prone to "collusion" attacks among the certificate authorities and compromising the overall network.

"A blockchain-based decentralized public-key infrastructure for information-centric networks" was presented in [16]. The framework sets up a decentralized PKI by combining the smart contracts of blockchain and optimized zero-knowledge proof-verifiable presentations. The framework realizes the management of public-key certificates through blockchain and ensures the authenticity and availability of public keys in decentralized infrastructure.

Table I summarizes the merits and limitations of the current blockchain-based access control schemes and proposed scheme.

TABLE I: Summary of the Related Work.

Decentralized access control scheme	Merit	Limitations
[4]	• Scalable • Usable	• Vulnerable to traceability, information disclosure, and collusion attacks
[5]	• Scalable	• Vulnerable to traceability attack • Less usable
[6, 7]	• Scalable • Usable	• Vulnerable to repudiation, information disclosure, impersonation, DoS, and collusion attacks
[8]	• Usable	• Not scalable • Vulnerable to traceability, information disclosure, impersonation, DoS, and collusion attacks
[9]	• Secure	• Scalability is not examined • Usability is not examined
[10]	• Usable	• Scalability is not examined • Resistance to private-key compromise attack is not analyzed
[11]	• Scalable	• Vulnerable to traceability, private-key compromise, and information disclosure attacks • Less usable
Proposed	• Scalable • Secure • Usable	• Need to conduct usability study as future work

III. Blockchain-based Architecture

With blockchain as the root-of-trust, the proposed scheme carries out authentication and access control for the IoT devices without the need for them to be part of the blockchain network and to keep enormous amount of blockchain data. In the proposed scheme, any blockchain node can register as a device manager on-demand to handle the rapidly growing number of IoT devices. The proposed scheme uses smart contract to store the information needed for access control in the blockchain and manage them. The blockchain-based architecture for the proposed scheme is presented in Fig. 1. The different components in Fig. 1 are described in the following section. The different stages labeled *"a - f"* are outlined in Section III-B.

A. Components in the architecture

The blockchain-based architecture has the following components:

1. IoT network: The IoT networks consist of resource-constrained IoT devices. These different networks are connected to the blockchain network through the interfaces.

2. Interface: An interface acts as an intermediary between the IoT and blockchain network that obtains Constrained Application Protocol (CoAP) messages from the IoT network, translates into blockchain-intelligible JSON - Remote Procedure Call (JSON-RPC) messages and vice-versa. The interface has abundant computing power, memory, and energy to handle the messages from IoT and blockchain network. There are many such interfaces in the architecture so that multiple and simultaneous messages can be effectively handled.

The blockchain network consists of a smart contract deploy node, device managers, miners, and a smart contract.

3. Smart contract deploy node: The Smart Contract Deploy Node (SCDN) is a special, privileged, and trusted blockchain node that deploys the smart contract on the blockchain network. It owns the smart contract. SCDN may be elected using the suitable trust metrics produced by an appropriate trust evaluation mechanism whose evaluation is based on the evidences (past behaviours). In this connection, the past behaviours of the blockchain nodes may be recorded in the blockchain. SCDN maintains the blockchain data to ensure data availability in the system in the event that all the blockchain nodes went offline.

4. Device manager: A device manager is a blockchain node which controls one or more devices in the IoT network. Only the device manager interacts with the smart contract to store the necessary information in the blockchain and manage them. It pays the fees for the blockchain transactions initiated to invoke the smart contract functions. However, it does not store blockchain information or validate transactions. Furthermore, the device manager performs only limited and lightweight offchain cryptographic operations. Therefore, even a resource-constrained device can enroll as a device manager. There are many device managers in the setup to ensure that i) the increasing number of IoT devices are managed effectively and ii) an IoT device enrolled under one or more device managers is operating under the control of at least a device manager in case the other managers go offline. Each manager can define access control policies for the IoT device. Thus, the multiple managers act as policy administration points for the IoT device resulting in decentralized policy administration.

5. Miner: The miners control the consensus process in the blockchain network. They validate the blockchain transactions. They execute the consensus algorithm for the blockchain transactions so that this new block of transactions can be added to the blockchain. Also, they store the blockchain information and therefore act as blockchain data repositories. The blockchain data can be queried from the miners whenever needed.

6. Smart contract: The smart contract has functions to store the essential information for access control in the blockchain and manage them. Only the device managers can invoke these functions by initiating blockchain transactions. The miners will keep a record of these transactions.

B. Outline of the different stages in the system

This section presents an outline of the different stages in the system.

a) The SCDN initiates the blockchain transaction to deploy the smart contract on the blockchain network. If the transaction is successful, the blockchain network returns the blockchain address of the contract to SCDN. SCDN holds this address to present it to those blockchain nodes willing to enroll as device managers. The steps in a device manager enrollment are explained in Section IV-A.

b) The interface acts as intermediary between the IoT and blockchain network. It is crucial that the authenticity of the interface is verified before any essential communication is made. The authenticity is verified using the interface's certificate. For this purpose, the device manager registers the

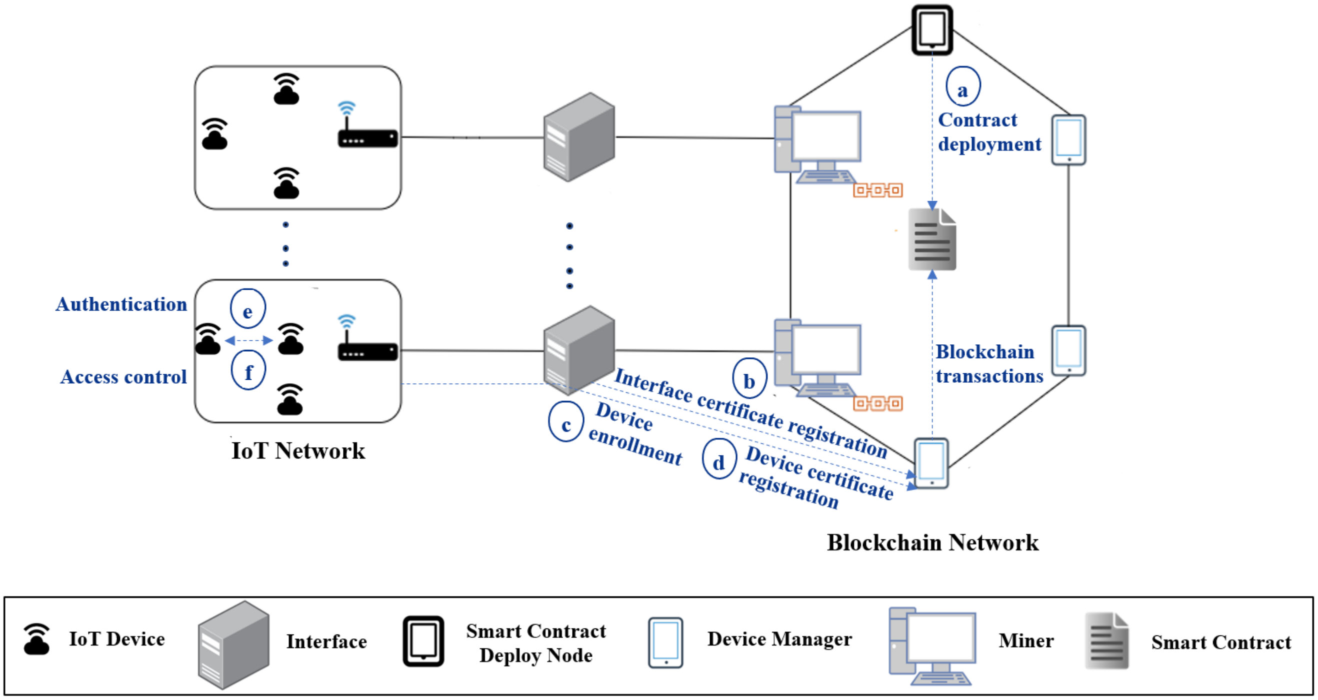

Fig. 1: Blockchain-based access control architecture.

interface's certificate in the blockchain. The device manager revokes the interface's certificate in case the interface exhibits any malicious behaviour. This stage is elaborated in Section IV-B.

c) Each IoT device operates under the control of at least one device manager in the system. Hence, it is essential that the IoT device enrolls under one or more device managers. The IoT device maintains the count of its device managers locally to avoid getting enrolled under too many managers. During an enrollment, the IoT device verifies interface's authenticity and then sends its consent to enroll under the device manager. The manager enrolls the device by initiating a blockchain transaction. The device manager disenrolls the IoT device if the device shows any signs of malicious behaviour. Thereafter, the information about the disenrollment is broadcasted to all other device managers so that those managers which have enrolled this malicious device can do disenrollment of this device. The enrollment of an IoT device under a device manager is detailed in Section IV-C.

d) It is critical that the IoT device is authenticated before it requests access to a resource in another IoT device. The authentication is performed based on the device's certificate. In this connection, the device's certificate is registered in the blockchain with the help of the device manager. Prior to this registration, the manager verifies if the device is enrolled. The manager can revoke the device's certificate in case any misbehaviour is observed. Thus, this setting presents a decentralized PKI based on blockchain technology. It does not require huge infrastructure and cumbersome certificate management unlike conventional PKI. Therefore, blockchain-based PKI is cost-effective compared to the conventional PKI. The steps in the registration of a device certificate are explained in Section IV-D.

e) The IoT devices are mutually authenticated using the blockchain-based PKI (decentralized PKI). The devices' certificates are queried from the miner and are used in the mutual authentication process. The authentication process is elaborated in Section IV-E.

f) When an IoT device requests access to a resource in another IoT device, it is absolutely necessary to verify the access rights of the requesting device. For this purpose, the device manager adds an access token containing the context and access rights for the requesting device in the blockchain by consulting the requested device. The access control process is carried out based on this token. The device manager revokes the device's token if any malicious behaviour is witnessed. Following this, the device is evicted from the current session. As a result, the device would be required to go through the authentication process once again. The access control process is detailed in Section IV-F.

IV. Proposed Scheme

In this section, we present our blockchain-based access control scheme for IoT. In the scheme, we first perform authentication and then access control to ensure legitimate and controlled access to critical and limited resources in IoT. Authentication is performed using the decentralized PKI based on blockchain technology presented by the scheme. Access control is carried out based on access token containing the context and access rights of an IoT device to a particular resource in another IoT device.

The proposed scheme is split into 6 stages: 1) Device manager enrollment (offline), 2) Registration of interface certificate (offline), 3) Enrollment of IoT device (offline), 4) Registration of device certificate (offline), 5) Authentication (online), and

6) Access control (online). Table II introduces the meanings for the symbols used in the scheme.

TABLE II: Symbols and their Meanings

Symbol	Meaning
b_addr_{DM}	Blockchain address of device manager $'DM'$
pu_k_{DM}, pr_k_{DM}	Public, private keys of DM
$msg_{register}$, msg_{enroll}, $msg_{consent}$, msg_{auth}	Messages
σ_1, σ_2, σ_3, σ_4	Digital signatures
id_{Int}	Identity of interface $'Int'$
pu_k_{Int}, pr_k_{Int}	Public, private keys of Int
exp	Expiration time
$uuid_{D1}$, $uuid_{D2}$	Universally unique identifiers of IoT devices $'D1'$ and $'D2'$
$suuid_{D1}, suuid_{D2}$	Secondary universally unique identifiers of $D1$ and $D2$
$count_{DM}$	Count of DMs controlling a device
C	Cipher text
rid	Registration identifier
rnd	Cryptographically strong random number
$Cert_{Int}$, $Cert_{D1}$, $Cert_{D2}$	Self-signed X.509 certificates of Int, $D1$ and $D2$
T	Timestamp
$\sigma5, \sigma6$	Digitally signed timestamps
$ctxt$	Context-awareness parameter
AR	Access rights
Cap_{D1}, Cap_{D2}	Capabilities of $D1$ and $D2$
cid	Random unique capability identifier
a_R	Requested access
$DS(.)$	Digital signature primitive
$E(.)$	Public-key encryption
$D(.)$	Private-key decryption
$h(.)$	Cryptographic hash
$CSPRNG(.)$	Cryptographically strong pseudo random number generator
$\|, \oplus$	Concatenation and bitwise XOR operations

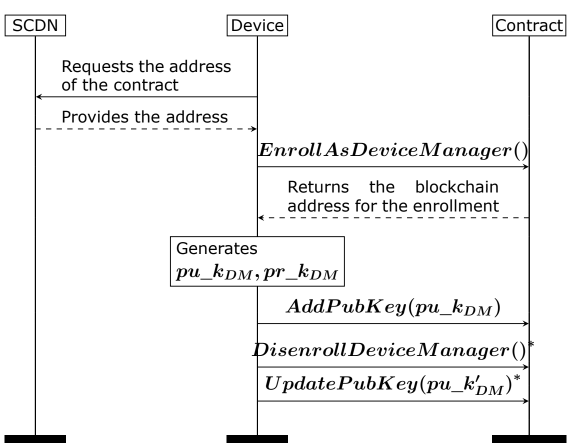

Fig. 2: Device manager enrollment.

A. Stage 1: Device manager enrollment

Fig. 2 presents the steps executed in the device manager enrollment stage. They are explained below:

Step 1. A device in the blockchain network which is willing to enroll as Device Manager (DM) requests SCDN for the blockchain address of the contract. SCDN provides the device with the address. Subsequently, the device initiates a blockchain transaction to invoke $EnrollAsDeviceManager()$ contract function. If the transaction is completed successfully, the device is returned a blockchain address b_addr_{DM} that identifies this device as DM. Thus, the device is enrolled as DM.

Step 2. DM generates Elliptic Curve Cryptography (ECC) or RSA-based public/private keys pu_k_{DM}, pr_k_{DM}. It then adds its public key to the blockchain by invoking $AddPubKey(pu\_k_{DM})$ contract function through a blockchain transaction. This is done to enable entities to query pu_k_{DM} from the miner in case if they wish to send any necessary information to DM in encrypted form.

Step 3 (Optional). A DM may disenroll itself by invoking $DisenrollDeviceManager()$ contract function. This results in revocation of all the capability tokens (access control policies) added by the DM through the $RevokeCapability()$ function. However, if this is the only DM for a particular IoT device, the function would not allow it to disenroll. In this manner, the scheme ensures that an IoT device has at least one DM at any point of time. A DM updates its key-pair periodically to avoid pr_k_{DM} leakage due to cyber attacks. Consequently, DM will have to perform periodic updation of its public key in the blockchain by invoking $UpdatePubKey(pu\_k'_{DM})$ contract function.

B. Stage 2: Registration of interface certificate

Fig. 3 represents the steps in the registration of interface certificate stage. These steps are described below:

Step 1. An Interface (Int) sends a request to register its certificate in the blockchain to DM. Following this, DM prepares register interface certificate message $msg_{register}$ and the corresponding signature $\sigma_1 = DS(msg_{register}, pr\_k_{DM})$. It decides the expiration time $'exp'$ which would be used by the interface Int in the following step in the preparation of its certificate. DM sends $b\_addr_{DM}, msg_{register}, \sigma_1$, and exp to Int.

Step 2. Int queries the public key of DM from the miner by calling $Query()$ method with suitable identity parameter b_addr_{DM}. It is to be noted that a call to $Query()$ method does not require a blockchain transaction. Thus, blockchain transactions are avoided in the context of accessing data from the miner. With pu_k_{DM}, Int validates DM's signature by checking $DS(\sigma_1, pu\_k_{DM}) == msg_{register}$ If the verification succeeds, Int generates id_{Int}, its public/private keys pu_k_{Int} and pr_k_{Int}. Thereafter, it prepares signature $\sigma_2 = DS(msg_{register}, pr\_k_{Int})$ and self-signed X.509 certificate $Cert_{Int} = (id_{Int} \| pu\_k_{Int} \| exp \| \sigma_2)$. It sends the encrypted certificate $C = E(Cert_{Int}, pu\_k_{DM})$ to DM.

Step 3. DM decrypts C using pr_k_{DM}, obtains $Cert_{Int}$ and then pu_k_{Int}. Subsequently, it validates interface signature by checking $DS(\sigma_2, pu\_k_{Int}) == msg_{register}$. It also checks $'exp'$ for integrity. If the verification is successful, DM computes $h(Cert_{Int})$, registers the certificate for Int by invoking $RegisterCertificate(id_{Int}, h(Cert_{Int}))$ contract function through a blockchain transaction. This function does not permit further registrations using the same id_{Int}. Once $'exp'$ is reached, an Int will have to register certificate again but using a different id_{Int}.

Step 4 (Optional). An Int updates its key-pair periodically to prevent pr_k_{Int} leakage owing to cyber attacks. Accordingly, Int prepares a new certificate $Cert'_{Int}$ using its new public key $pu\_k'_{Int}$ and sends it to DM.

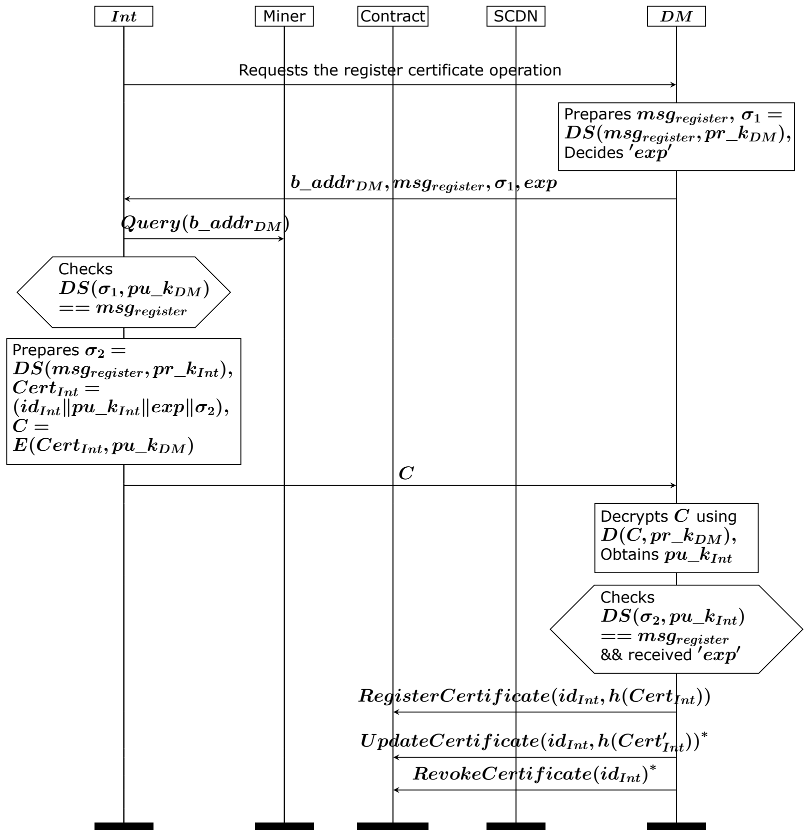

Fig. 3: Registration of interface certificate.

Then, DM updates the certificate of Int by invoking $UpdateCertificate(id_{Int}, h(Cert'_{Int}))$ contract function. This function does not allow a new/fresh certificate registration. If an Int turns malicious at any time before the expiration time $'exp'$ of its certificate, the DM revokes the certificate by invoking the $RevokeCertificate(id_{Int})$ contract function.

C. Stage 3: Enrollment of IoT device

The steps executed in enrollment of IoT device are elaborated as follows:

Step 1. An IoT device, say $D1$ verifies the authenticity of Int and DM before it sends its consent to enroll under DM. For this purpose, DM prepares enroll IoT device message msg_{enroll} and the corresponding signature $\sigma_3 = DS(msg_{enroll}, pr\_k_{DM})$. It sends msg_{enroll} and σ_3 to Int. Int forwards these parameters along with $Cert_{Int}$ to $D1$.

Step 2. $D1$ verifies if the count of its DMs is less than a predefined threshold by checking $count_{DM} < threshold$. It maintains $count_{DM}$ locally. If the condition is tested false, $D1$ aborts the procedure to prevent too many DMs from controlling it. Else, $D1$ queries $h(Cert_{Int})$ and pu_k_{DM} from miner by calling $Query()$ method with suitable parameters id_{Int} and b_addr_{DM} respectively. This call to $Query()$ does not incur a blockchain transaction. With the parameters received from miner, $D1$ verifies the authenticity of Int and DM by checking computed $h(Cert_{Int}) ==$ queried $h(Cert_{Int})$ && $DS(\sigma_3, pu\_k_{DM}) == msg_{enroll}$. If the verification succeeds, $D1$ sends its consent message $msg_{consent}$ and encrypted identity $C = E(uuid_{D1} \| suuid_{D1}, pu\_k_{DM})$ to DM. We chose to use universally unique identifiers for identifying the IoT devices globally since they are unique with almost zero probability of getting duplicated.

Step 3. DM decrypts C using pr_k_{DM} and obtains $uuid_{D1}$. Subsequently, it enrolls $D1$ by initiating a blockchain transaction that invokes $EnrollIoTDevice(uuid_{D1})$ contract function. If this transaction is completed successfully, $uuid_{D1}$ will be stored in the blockchain. Also, DM inserts $suuid_{D1}$ into the list of enrolled devices maintained and managed locally. The preceding function enables IoT devices to get enrolled under multiple DMs.

Step 4 (Optional). At any point of time, if a DM sees an IoT device (say $D1$) under its control, turning malicious, DM disenrolls $D1$ and revokes all the capabilities created for $D1$ by invoking $DisenrollIoTDevice(uuid_{D1})$ contract function. This information can be broadcasted to all other DMs so that those DMs which have enrolled this malicious $D1$ can do disenrollment and revocation of all of its capabilities by invoking the same function. These steps are illustrated in Fig. 4.

D. Stage 4: Registration of device certificate

The steps carried out in the registration of device certificate stage are explained below:

Step 1. As the initial step, $D1$ sends certificate registration request to DM. Subsequently, DM decides the expiration time $'exp'$ for the device ($D1$) certificate and sends it to

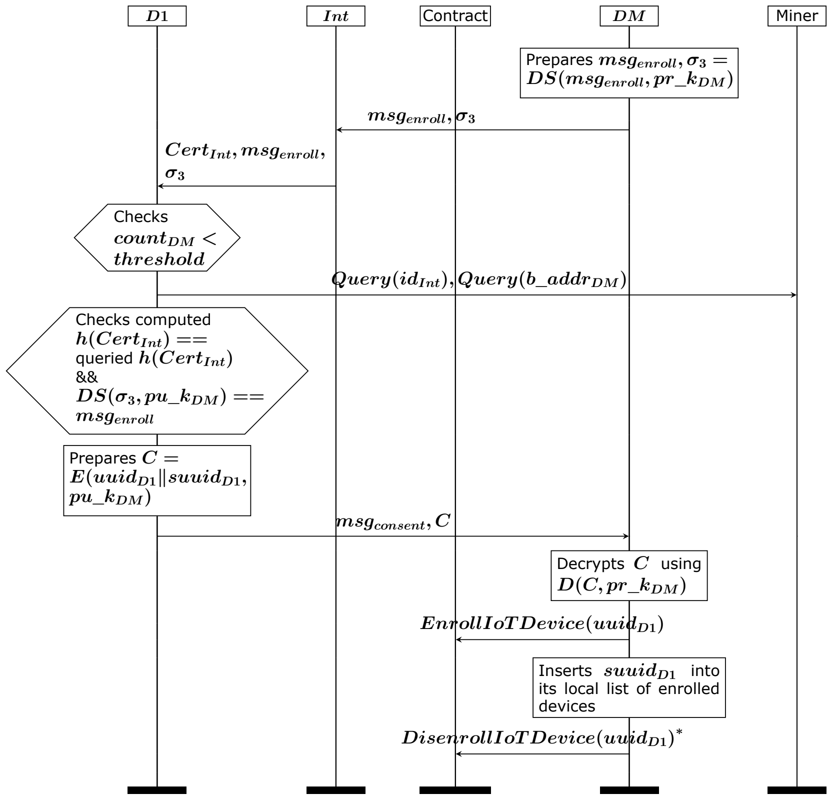

Fig. 4: Enrollment of IoT device.

$D1$. This $'exp'$ usually is shorter than the $'exp'$ for Int since the scope of device certificate is less than that of interface certificate. $D1$ generates cryptographically strong pseudo random number $'rnd'$ using $CSPRNG()$ function. It computes registration id $'rid'$ by performing bitwise XOR of $suuid_{D1}$ with $'rnd'$. Subsequently, $D1$ prepares authenticate device message msg_{auth} and the respective signature $\sigma_4 = DS(msg_{auth}, pr\_k_{D1})$. Besides, $D1$ prepares self-signed X.509 certificate $Cert_{D1} = (rid \| pu\_k_{D1} \| exp \| \sigma_4)$ and encrypts it as in $C = E(Cert_{D1} \| rnd, pu\_k_{DM})$. $D1$ sends msg_{auth} and C to DM through Int.

Step 2. DM decrypts C using pr_k_{DM}, obtains $Cert_{D1}$ and then pu_k_{D1}. Next, DM computes $suuid_{D1}$ by performing bitwise XOR of $'rid'$ with $'rnd'$. DM checks if $D1$ is enrolled using $(suuid_{D1})$. Only if this verification succeeds, DM validates the signature of $D1$ by checking $DS(\sigma_4, pu\_k_{D1}) == msg_{auth}$. It also checks the integrity of $'exp'$. After successful validation, DM computes $h(Cert_{D1})$ and registers the certificate for $D1$ by invoking $RegisterCertificate(rid, h(Cert_{D1}))$ contract function. This function does not allow further registrations on the same $'rid'$. $D1$ is required to register certificate again, once $'exp'$ of its certificate is reached.

Step 3 (Optional). If DM finds $D1$ turning malicious at any time before the expiration time $'exp'$, $D1$'s certificate is revoked by invoking $RevokeCertificate(rid)$ contract function. Following this, DM takes all the necessary steps for disenrollment and cascading disenrollment (disenrollment by the other DMs) of $D1$. The above steps are outlined in Fig. 5.

E. Stage 5: Authentication

Suppose $D1$ wants to access a particular resource of $D2$. This is permitted only after successful mutual authentication between $D1$ and $D2$. The essential steps in mutual authentication are detailed below:

Step 1. $D1$ sends $Cert_{D1}$ to $D2$. Following this, $D2$ queries $h(Cert_{D1})$ from the miner by calling $Query(rid)$ method. $D2$ validates the certificate presented by $D1$ by checking if computed $h(Cert_{D1}) ==$ queried $h(Cert_{D1})$. If the verification succeeds, $D2$ stores $'rid', h(Cert_{D1}), 'exp'$ into the list of trusted devices. This list is maintained and managed locally by $D2$. $D1$ follows the aforementioned procedure to store $'rid', h(Cert_{D2}), 'exp'$ for $D2$ into its list of trusted devices. This step (Step 1.) is performed once per certificate registration for a device.

Step 2. $D1$ sends $Cert_{D1}, T, \sigma 5$ to $D2$. $D2$ verifies if $DS(\sigma 5, pu\_k_{D1}) == T$ and $|T - T^*| \leq \Delta T$ where T^* is the reception time and ΔT is the maximum transmission delay. If so, it extracts $'rid'$ from $Cert_{D1}$ and checks if this is present in the list of trusted devices. If present, $D2$ checks if computed $h(Cert_{D1}) ==$ stored $h(Cert_{D1})$ and $'exp'$ is greater than the current time. If the condition evaluates to true, $D1$ is authenticated to $D2$. Subsequently, $D2$ dispatches $Cert_{D2}, T, \sigma 6$ to $D1$. $D1$ checks if $DS(\sigma 6, pu\_k_{D2}) == T$ and $|T - T^*| \leq \Delta T$. If yes, $D1$ acquires $'rid'$ from $Cert_{D2}$, then verifies if this is existent in the list. If yes, $D1$ verifies if computed $h(Cert_{D2}) ==$ stored $h(Cert_{D2})$ and the current

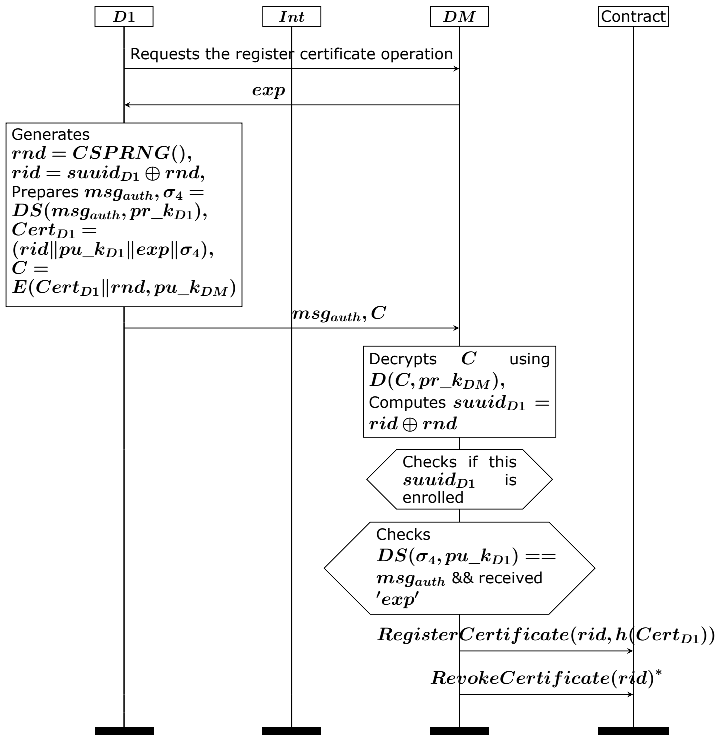

Fig. 5: Registration of device certificate.

time does not exceed $'exp'$. If the condition is tested true, $D2$ is authenticated to $D1$. $D1$ and $D2$ notify DM of this mutual authentication by sending the desired messages to DM through Int. Thus, $D1$ and $D2$ are mutually authenticated to one another. The steps in mutual authentication are represented in Fig. 6.

F. Stage 6: Access control

The steps in the access control stage are explained below:

Step 1. Suppose $D1(uuid_{D1})$ requests $D2$ access over resource $'r'$. Resources are identified by their names. $D2$ decides the context-awareness parameter $'ctxt'$ and access rights AR. Since the resource $'r'$ is most likely a file, $AR \in \{null, read, write, \{read, write\}\}$. $'ctxt'$ can be time or location. $D2$ encrypts these parameters as in $C = E(ctxt \| AR, pu\_k_{DM})$. $D2$ dispatches $C, uuid_{D1}, r$ to DM.

Step 2. DM decrypts C using pr_k_{DM}. DM decides $'exp'$ & $'rnd'$ for the capability token to be generated. Then, it computes capability for $D1$ viz., $Cap_{D1} = h(uuid_{D1}, r, ctxt, AR, exp, rnd)$. DM initiates a blockchain transaction to trigger $AddCapability(cid, uuid_{D1}, Cap_{D1} \| exp)$ contract function to add this capability in the blockchain. Also, DM sends cid, Cap_{D1} to $D1$ so that $D1$ can present the capability token to $D2$ when needed.

Step 3. When $D1$ presents cid, Cap_{D1} to $D2$, $D2$ queries the miner for $D1$'s capability by calling $Query(cid)$ method. It then verifies if presented Cap_{D1} == queried Cap_{D1}. It also checks if queried $'exp'$ is greater than the current time. If this condition is tested true, the capability token presented is valid. Subsequently, $D1$ presents the requested access a_R to $D2$. $D2$ maintains and manages $'ctxt'$ and AR for each capability token locally. $D2$ validates the current context and checks if $a_R \in AR$. If true, the requested access is granted.

Step 4 (Optional). If DM encounters an IoT device say $D1$ turning malicious in the access control stage at any time before $'exp'$, it revokes the capability token of $D1$ by invoking $RevokeCapability(cid)$ contract function. Following this, $D1$ is expelled from the current session which would require $D1$ to go through the authentication process again. These steps are illustrated in Fig. 7.

Why would the industries adopt the proposed scheme?

In recent years, nearly, all industry sectors use IoT devices in different applications. As the applications expand, the usage increases and therefore, the IoT devices would rapidly grow in number. Besides, security in these IoT applications is of great concern to the industries. Therefore, the industries are desirous of a robust access control scheme that scales well with the rapidly increasing number of IoT devices. The industries would adopt the proposed scheme since it fetches the following advantages to access control in IoT.

- Fine-grained: In the system, the access token is generated for a particular IoT device and resource. The token has context-awareness and access rights fields. The context-awareness field has information such as location or time of the day. The device holding the token has to present it at the time of access verification. The access to a specific resource is determined based on the access rights and context. Thus, access control in the proposed scheme is fine-grained.
- Scalability: In the system, any resource-constrained blockchain node can become a device manager. A device

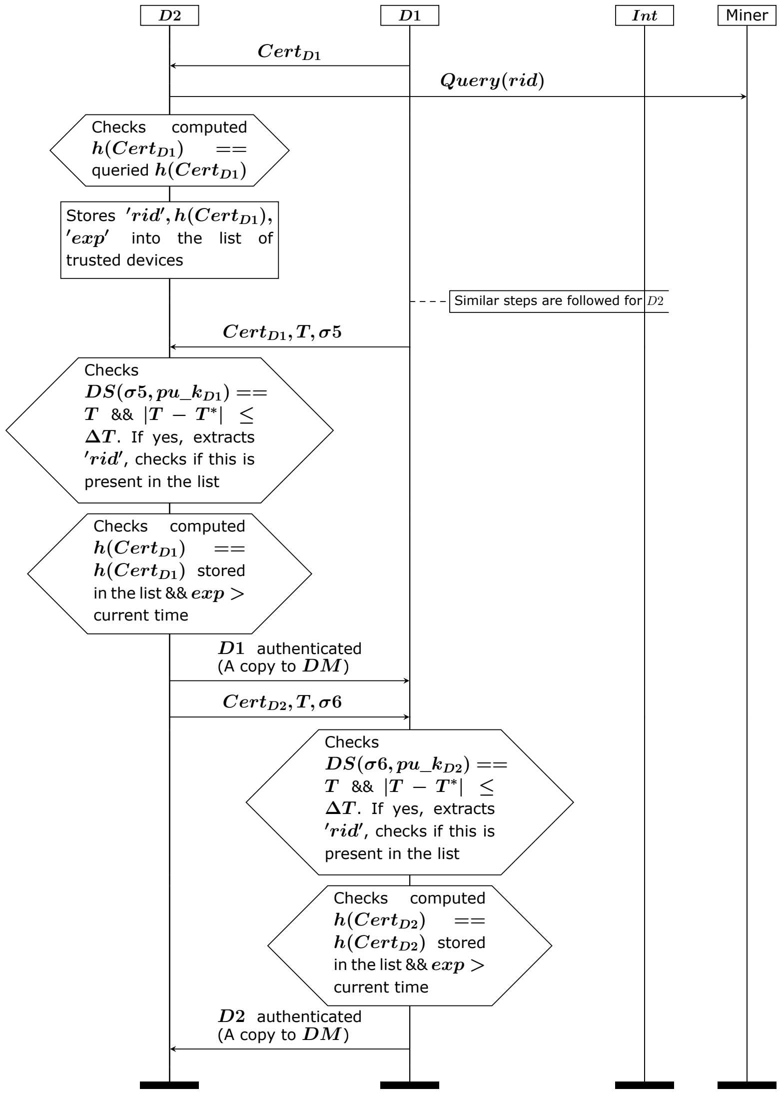

Fig. 6: Authentication.

manager controls one or more IoT devices. The manager initiates only limited number of blockchain transactions and performs only limited & lightweight offchain cryptographic operations as a part of the access control process for an IoT device. There are many such managers in the system. Therefore, the scheme scales well with the increasing number of IoT devices.

- Usability: Although blockchain technology is central to the proposed scheme, the technology is not integrated into the resource-constrained IoT devices. The IoT devices neither store blockchain information nor validate transactions. This makes the scheme usable in many IoT scenarios.
- Interoperability: The interfaces translate CoAP messages from IoT devices into blockchain intelligible JSON-RPC messages and vice versa. The essential communications between the IoT devices and blockchain network are established by the interfaces. This makes the IoT devices and blockchain network interoperable.
- Security: The IoT devices verify the authenticity of the interfaces before any communication is made with the blockchain network. Also, the IoT devices are authenticated and access rights are verified before access to a particular resource is granted. Moreover, data tampering is prevented in the root of trust with the use of blockchain. Blockchain has inherent tamper-proof property. Thus, security is ensured in all the places in the proposed scheme.

V. Security Analysis

In this section, the proposed scheme is carefully examined for its resilience to conventional security attacks listed in the classical threat models [12, 13] and additional standard attacks in IoT environment. The security analysis by reasoning is presented below.

Proposition 1. *Proposed scheme is resistant to repudiation attack.*

Proof. $D1$ or $D2$ could not claim to have not performed an action because DM keeps track of their actions. For instance,

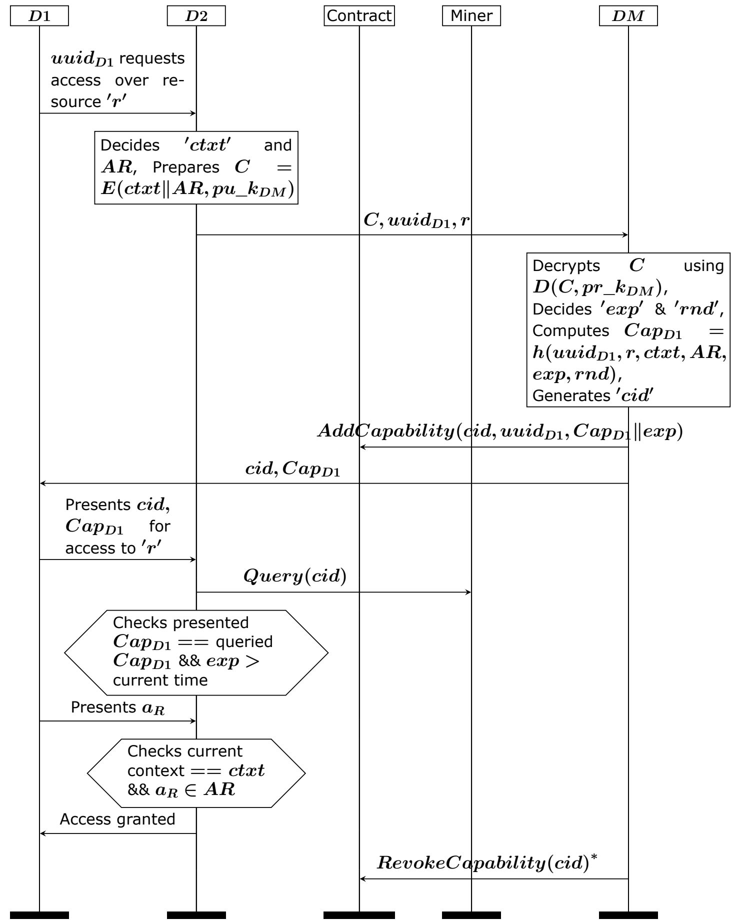

Fig. 7: Access control.

DM enrolls a device only after getting its consent. DM could not claim to have not done an action since many of its actions would initiate blockchain transactions to invoke contract functions which are recorded in the blockchain. An Int could not succeed in carrying out a repudiation attack since its essential actions are followed up by DM. For example, DM registers the certificate for an Int only after receiving the request from it. □

Proposition 2. *Proposed scheme protects off-chain cryptographic parameters from traceability attack.*

Proof. An adversary could not trace an Int since $id_{Int}, Cert_{Int} = \{id_{Int} \| pu\_k_{Int} \| exp \| \sigma 2\}$ would be different for different certificate registrations. $D1$ could not be traced because $uuid_{D1}$ and $suuid_{D1}$ are different for different enrollments. Universally unique identifiers have the properties of being random and collision-resistant. Besides, $rid = suuid_{D1} \oplus rnd, Cert_{D1} = \{rid \| pu\_k_{D1} \| exp \| \sigma 4\}$ would be different for different certificate registrations. An adversary could not succeed in tracing $D2$ for the similar reason. □

Proposition 3. *Proposed scheme is resistant to private key compromise attack.*

Proof. An attacker could not compromise pr_k_{DM} because DM keeps it secret. Besides, DM generates a new key-pair $(pr\_k'_{DM}, pu\_k'_{DM})$ at regular intervals. It updates $pu\_k'_{DM}$ in the blockchain by invoking the contract function $UpdatePubKey(pu\_k'_{DM})$. The private key of Int could not be compromised since Int generates a new key-pair $(pr\_k'_{Int}, pu\_k'_{Int})$ periodically. The resultant certificate $Cert'_{Int}$ is updated in the blockchain by DM through the invocation of contract function $UpdateCertificate(id, h(Cert'_{Int}))$. Similarly, the private keys of the IoT devices could not be compromised because fresh key-pairs are generated from time to time. □

Proposition 4. *Proposed scheme is resilient to information disclosure attack.*

Proof. No information could be revealed from $h(Cert_{Int}), h(Cert_{D1})$, and $h(Cert_{D2})$ stored in the blockchain since it is infeasible to find a collision for a cryptographic hash in polynomial time. $D1$ could not disclose any access control information viz., $'ctxt', AR$, and $'exp'$ from Cap_{D1} received from DM during capability-based access control as Cap_{D1} uses cryptographic hash function.

An Int could not divulge 1) any information about $D1$, $D2$ from $h(Cert_{D1})$ and $h(Cert_{D2})$ queried from the miner during certificate-based authentication because they are cryptographic hashes, and 2) any access control information from Cap_{D1} obtained from DM since Cap_{D1} uses cryptographic hash function. □

Proposition 5. *Proposed scheme is secure against impersonation attack.*

Proof. $D1$ or $D2$ could not be spoofed at the time of enrollment because the identities $uuid_{D1}, uuid_{D2}$ are sent to DM in encrypted form. In the same manner, $D1$ or $D2$ could not be impersonated during certificate registration since the registration identities $'rid'$s are dispatched to DM in the form of ciphertext. $D1$ or $D2$ could not be imitated during certificate-based authentication by capturing the certificates $Cert_{D1}, Cert_{D2}$ and resending them (replaying). This is because the certificates are sent along with the timestamps and digitally signed timestamps $\sigma 5, \sigma 6$ that ensure the timestamps are not manipulated during transit. These timestamps prevent replay attacks, thereby, intercepts impersonation attacks.

An Int could not be impersonated during certificate registration since id_{Int} is sent to DM in encrypted form. A DM would not be willing to impersonate another DM, since it would have to endure the transaction costs incurred on the blockchain transactions initiated in favour of the other DM. □

Proposition 6. *Proposed scheme prevents DoS attack.*

Proof. $D1$, $D2$ and Int could not become successful in conducting single identity DoS attacks using their corresponding identities $uuid_{D1}$, $uuid_{D2}$ and id_{Int}. The scheme prevents such attacks. On the other hand, a DM would not be willing to conduct DoS attacks as it would need to endure the transaction costs incurred on the initiated blockchain transactions. □

Proposition 7. *Proposed scheme intercepts collusion attack.*

Proof. The DMs of an IoT device say, $D1$ could not succeed in conducting a collusion attack on $D1$ because $D1$ verifies enrollment under every DM. Moreover, $D1$ poses an upper bound on the number of DMs with which it could enroll. $D1$ maintains a local $count_{DM}$ variable to verify if the upper bound is reached. □

VI. Result Analysis

In the proposed scheme, SCDN and DM bear the costs for the blockchain transactions initiated for contract deployment and contract operations respectively. As a result, initially, the transaction costs for contract deployment and different contract operations in the proposed scheme are studied in this section. For this purpose, a Proof-of-Concept (PoC) implementation is developed in Ethereum blockchain platform. The PoC is deployed and tested in various Ethereum public testnets before deployment in the Ethereum Mainnet [17]. First, it is tested in the Remix testnet. Thereafter, the PoC is tested in the recent Goerli [18] and Sepolia [19] testnets. The PoC is then deployed in the mainnet to obtain real transaction costs for contract deployment and operations in the scheme.

The proposed scheme is anticipated to scale reasonably well. Therefore, secondly, the scalability of the proposed scheme in different scenarios is evaluated. These different scenarios require the use of different type and number of cryptographic primitives. The cryptographic complexities of these primitives are studied and used in scalability evaluation. Lastly, the storage, computational, and communication overhead of the proposed scheme are studied.

A. Proof-of-concept implementation

The purpose of this PoC implementation is to evaluate the transaction costs for the contract deployment and contract operations in the proposed scheme. The single smart contract in the proposed scheme is implemented as two contracts in PoC. The motive for two smart contracts is that the transaction cost for the deployment of a single smart contract containing all the required functions for access control exceeded the default gas limit. When the transaction cost for a contract deployment exceeds the default gas limit, it implies it is heavy for the blockchain network. This has necessitated the break down of a single contract into two smart contracts in PoC. The first contract (Contract1) contains the definitions of functions needed for handling the device managers, IoT devices and access tokens. Whereas, the second contract (Contract2) holds the definitions of functions to manage the certificates required for verifying the legitimacy of IoT devices/interfaces. Thus, Contract2 forms the basis for blockchain-based PKI in the proposed scheme.

Algorithm 1: Enroll as device manager
Contract operation –"Op1"

```
function EnrollAsDeviceManager()
if deploy_node = msg.sender then
    'The smart contract deploy node cannot enroll as device manager'
    return 0;
end
if devicemanager[msg.sender].isdevicemanager = true then
    'The caller is already a device manager'
    return 0;
end
devicemanager[msg.sender].isdevicemanager ← true;
/* Let b_addr be the blockchain address returned for
   the enrolled device manager */
return b_addr;
```

Algorithm 2: Enroll IoT device
Contract operation – "Op2"

```
function EnrollIoTDevice(uuid)
if devicemanager[msg.sender].isdevicemanager ≠ true then
    'The caller has to be a device manager'
    return 0;
end
/* The above steps verify if the caller is a device
   manager */
for i = 1 to device[uuid].devicemanagers.length do
    if device[uuid].devicemanagers[i] = msg.sender then
        flag ← true;
        break;
    end
end
if flag = true then
    'The device is already enrolled under this device manager'
    return 0;
end
if device[uuid].isdevice ≠ true then
    device[uuid].isdevice ← true;
end
device[uuid].devicemanagers.push(msg.sender);
devicemanager_devices[msg.sender].list_devices.push(uuid);
return 1;
```

The data structures used in Contract1 and Contract2 are depicted in Fig. 8. Mapping data structures are hash tables that

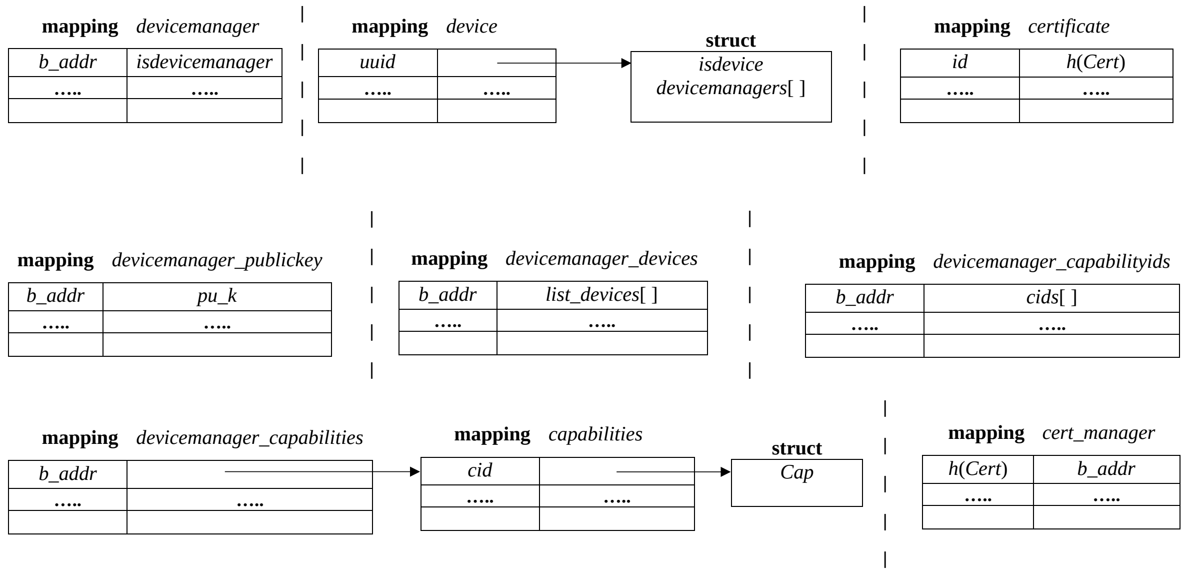

Fig. 8: Data structures used in the smart contracts.

store information in the form of key-value pairs. Algorithms 1–9 are realization of the contract functions in the proposed scheme. At PoC level, the algorithms 1–6 represent Contract1 functions. Whereas, algorithms 7–9 represent Contract2 functions. These algorithms demonstrate that the contract functions carry out necessary security validations to avoid security loopholes at the system level. For instance, almost every algorithm performs the security validation "if the caller is a device manager" to ensure that no component in the architecture (Fig. 1) other than the device manager is allowed to invoke any of the smart contract functions.

Algorithm 3: Add capability
Contract operation – "Op3"

```
function AddCapability(cid, uuid, Cap)
/* Verifies if the caller is a device manager */
for i = 1 to device[uuid].devicemanagers.length do
    if device[uuid].devicemanagers[i] = msg.sender then
        flag1 ← true;
        break;
    end
end
/* Let #uuid be the id of the other device */
/* The above steps are repeated for #uuid. The flag variable is
   flag2 */
if flag1 or flag2 ≠ true then
    'The caller is not a manager of both the devices'
    return 0;
end
devicemanager_capabilities[msg.sender].capabilities[cid].Cap ← Cap;
return 1;
```

As the first step of testing the PoC, the contracts are deployed in Remix testnet under JavaScript VM (London) environment. The transaction costs incurred in blockchain transactions for contract deployment and different contract operations are indicated in Fig. 9.

It can be observed that the costs (in gas) incurred in deploying the contracts are within the default gas limit of Remix viz., 3000000 gas. The cost for deploying Contract1 is significantly higher than that for Contract2, since Contract1 has more number of functions than Contract2. Had there been a single contract managing all the operations, the cost of deployment would have definitely surpassed the default gas

Algorithm 4: Disenroll device manager
Contract operation – "Op4"

```
function DisenrollDeviceManager()
/* Verifies if the caller is a device manager */
for i = 1 to devicemanager_devices[msg.sender].list_devices.length do
    temp ← devicemanager_devices[msg.sender].list_devices[i];
    if device[temp].devicemanagers.length = 1 then
        flag ← true;
        break;
    end
end
if flag = true then
    'The device manager cannot disenroll as it is the only manager for a device'
    return 0;
end
for i = 1 to devicemanager_devices[msg.sender].list_devices.length do
    temp ← devicemanager_devices[msg.sender].list_devices[i];
    for j = 1 to device[temp].devicemanagers.length do
        if device[temp].devicemanagers[j] = msg.sender then
            delete device[temp].devicemanagers[j];
            break;
        end
    end
end
/* Invalidate all the capabilities added by the device manager */
for i = 1 to devicemanager_capabilityids[msg.sender].cids.length do
    RevokeCapability(devicemanager_capabilityids[msg.sender].cids[i]);
end
delete devicemanager_devices[msg.sender].list_devices;
/* Make the device manager a mere blockchain node */
devicemanager[msg.sender].isdevicemanager ← false;
return 1;
```

Algorithm 5: Disenroll IoT device
Contract operation – "Op5"

```
function DisenrollIoTDevice(uuid)
/* Verifies if the caller is a device manager */
for i = 1 to devicemanager_devices[msg.sender].list_devices.length do
    if devicemanager_devices[msg.sender].list_devices[i] = uuid then
        delete devicemanger_devices[msg.sender].list_devices[i];
        break;
    end
end
for i = 1 to device[uuid].devicemanagers.length do
    if device[uuid].devicemanagers[i] = msg.sender then
        delete device[uuid].devicemanagers[i];
        break;
    end
end
if device[uuid].devicemanagers.length = 0 then
    device[uuid].isdevice ← false;
end
return 1;
```

limit of Remix. This justifies the need of having two contracts in the PoC implementation.

Algorithm 6: Revoke capability
Contract operation – "Op6"

```
function RevokeCapability(cid)
/* Verifies if the caller is a device manager */
for i = 1 to devicemanager_capabilityids[msg.sender].cids.length do
    if devicemanager_capabilityids[msg.sender].cids[i] = cid then
        flag ← true;
        break;
    end
end
if flag ≠ true then
    'The caller cannot revoke the capability since the caller has not worked on it'
    return 0;
end
delete devicemanager_capabilities[msg.sender].capabilities[cid];
return 1;
```

The contract functions presented in Algorithms 1–9 represent the different contract operations. It can be seen from Fig. 9b that the transaction costs incurred in all the operations are well within the default gas limit of Remix. The costs are bearable by DM. The maximum cost is incurred in "Enroll IoT device" operation whose corresponding function $EnrollIoTDevice(uuid)$ is presented in Algorithm 2.

Algorithm 7: Register certificate
Contract operation – "Op7"

```
function RegisterCertificate(id, h(Cert))
/* Verifies if the caller is a device manager */
if certificate[id] ≠ NULL then
    'A certificate for the interface is already registered'
    return 0;
end
certificate[id] ← h(Cert);
cert_manager[h(Cert)] ← msg.sender;
/* Similar steps are followed to register certificate
   for a device with registration id 'rid' */
return 1;
```

Algorithm 8: Update certificate
Contract operation – "Op8"

```
function UpdateCertificate(id, h(Cert'))
/* Verifies if the caller is a device manager */
if certificate[id] = NULL then
    'A certificate is not yet registered for the interface'
    return 0;
end
if cert_manager[certificate[id]] ≠ msg.sender then
    'The certificate can't be updated by this manager'
    return 0;
end
certificate[id] ← h(Cert');
cert_manager[h(Cert')] ← msg.sender;
return 1;
```

It is worth noticing the costs incurred in the revocation operations namely "Revoke capability" and "Revoke certificate" whose functions are presented in Algorithms 6 and 9. It can be ascertained from Fig. 9b that the cost for "Revoke capability" operation is less than that of all other operations of Contract1. The cost for "Revoke certificate" operation is less than 30000 gas which is considered trivial. Had the costs for these operations been high, it would have lead to some serious security threats. For instance, high costs incurred in "Revoke capability" operation would allow a malicious IoT device gain

Algorithm 9: Revoke certificate
Contract operation – "Op9"

```
function RevokeCertificate(id)
/* Verifies if the caller is a device manager */
if cert_manager[certificate[id]] ≠ msg.sender then
    'The certificate can't be revoked as the device manager has not worked
      on it'
    return 0;
end
cert_manager[certificate[id]] ← 0;
delete certificate[id];
return 1;
```

unauthorized access. While, that in "Revoke certificate" would leave a malicious Int or IoT device as a legitimate one. Thus, the proposed scheme ensures no security loopholes arising from the revocation operations.

As the main testing phase, we deployed the smart contracts in the possibly-stable Goerli and Sepolia testnets. The ETH to work on these testnets are obtained from the testnet faucets [20, 21]. The transaction costs (ETH) for different contract operations in these testnets are shown in Fig. 9c, 9d. The costs are significantly low in Sepolia testnet. This is due to the low and constant base fee of 0.000000007 GWEI in Sepolia testnet compared to the variable and comparatively high base fee in Goerli testnet. The priority fee in both the testnets was 2.5 GWEI.

After rigorous testing and "The Merge" upgrade [22], we deployed PoC in Ethereum Mainnet to obtain the real and latest figures for the transaction costs for contract deployment and different contract operations in the scheme. We conducted the evaluation on October 23, 2022. Fig. 10 presents the transaction costs in ETH and USD. The priority fee in the mainnet during observation was 2.5 GWEI. The ETH price in the mainnet was $1313.19. The highest cost ($63.14) is recorded for Contract1 deployment. Though the cost is high, the contract is deployed only once. On the other hand, the costs for the contract operations are less (sometimes significantly less) than $3.00 which are fairly low transaction costs.

Table III provides a summary of transaction costs of different contract operations in Ethereum Mainnet, Goerli and Sepolia testnets, and Remix testnet. The total transaction cost of contract operations in Ethereum Mainnet is 0.057952 ETH ($76.05). Majority of the cost $67.5 ($63.14 + $4.36) is incurred in one-time contract (Contract1 and Contract2) deployment operations. Hence, the contract operations in the proposed scheme incur only reasonable transaction costs in Ethereum Mainnet. The total transaction cost in Sepolia testnet is significantly less compared to the total cost in Goerli testnet due to significantly low effective gas price (low base fee) in Sepolia testnet. The total gas needed for the contract operations in the scheme is 3862217.

As a part of cost evaluation, we conducted a study on the frequency of use of smart contract functions in a typical use. According to a survey by Deloitte [23], the average number of IoT devices per US household in this year is 22. Another study by Statista [24] claims that the average number of connected devices per household in Australia in 2021 was 21. These statistics are used to define the frequency of use of contract

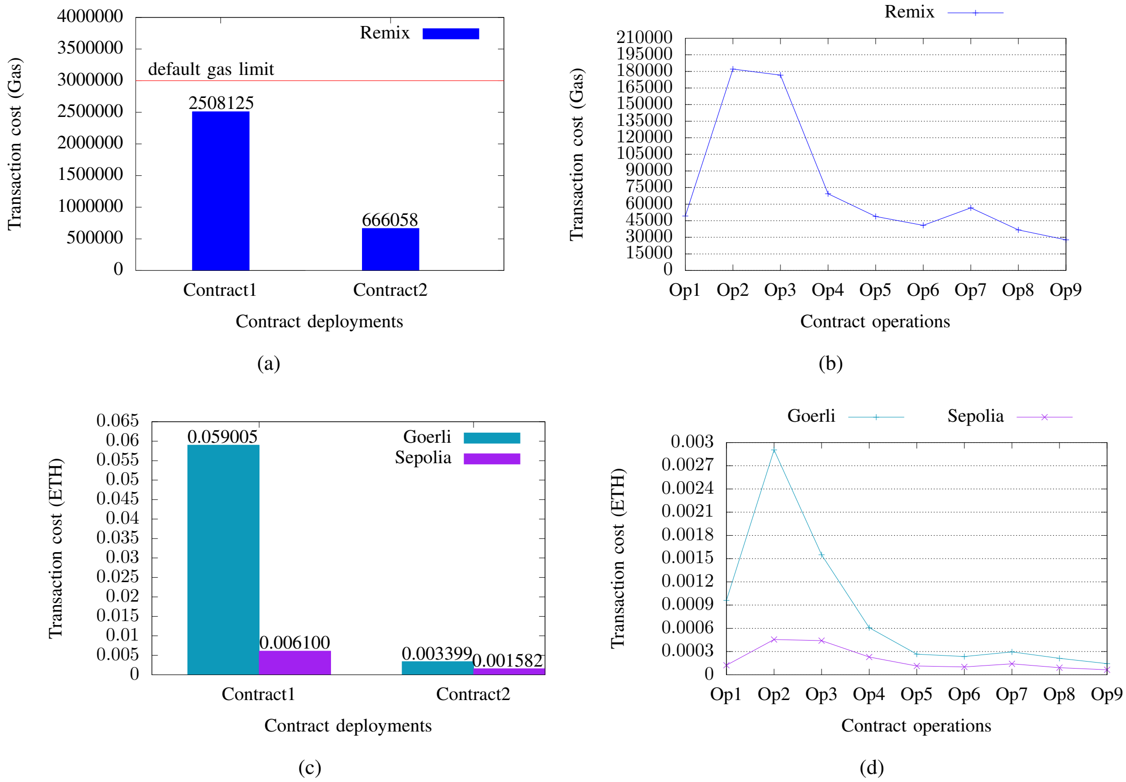

Fig. 9: (a) and (b): Transaction costs (gas) of contract deployments and operations in Remix testnet; (c) and (d): Transaction costs (ETH) of contract deployments and operations in the recent Goerli and Sepolia testnets. The priority fee in these testnets was 2.5 GWEI.

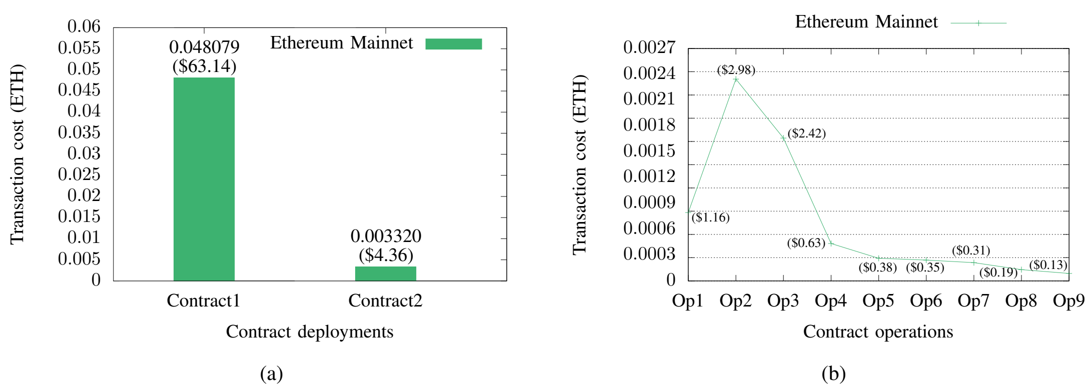

Fig. 10: (a) and (b): Transaction costs (ETH) of contract deployments and operations in Ethereum Mainnet after "The Merge". The priority fee in the mainnet was 2.5 GWEI. The ETH price in the mainnet on October 23, 2022 was $1313.19

TABLE III: Summary of transaction costs of different contract operations in Ethereum Mainnet, Goerli and Sepolia testnets, and Remix testnet

Contract operation	**Ethereum Mainnet**		**Ethereum Public Testnets**		**Remix Testnet**
			Goerli	**Sepolia**	
	(ETH)	**(USD)**	**(ETH)**	**(ETH)**	**(Gas)**
Contract1 deployment	0.048079	63.14	0.059005	0.006100	2508125
Contract2 deployment	0.003320	4.36	0.003399	0.001582	666058
Enroll as device manager (Op1)	0.000882	1.16	0.000962	0.000123	49290
Enroll IoT device (Op2)	0.002305	2.98	0.002904	0.000455	182091
Add capability (Op3)	0.001844	2.42	0.001551	0.000442	176652
Disenroll device manager (Op4)	0.000483	0.63	0.000609	0.000230	69364
Disenroll IoT device (Op5)	0.000292	0.38	0.000266	0.000114	48811
Revoke capability (Op6)	0.000269	0.35	0.000235	0.000102	40797
Register certificate (Op7)	0.000236	0.31	0.000296	0.000142	56651
Update certificate (Op8)	0.000145	0.19	0.000212	0.000092	36698
Revoke certificate (Op9)	0.000097	0.13	0.000145	0.000064	27680
Summation	0.057952	76.05	0.069584	0.009446	3862217

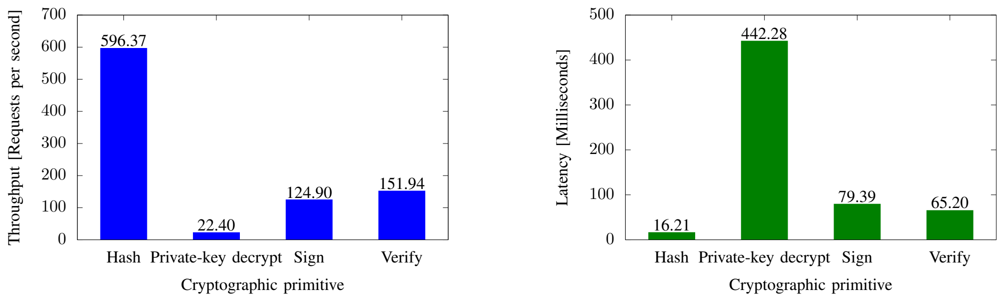

Fig. 11: Throughput and latency of different cryptographic operations in Raspberry Pi 3.

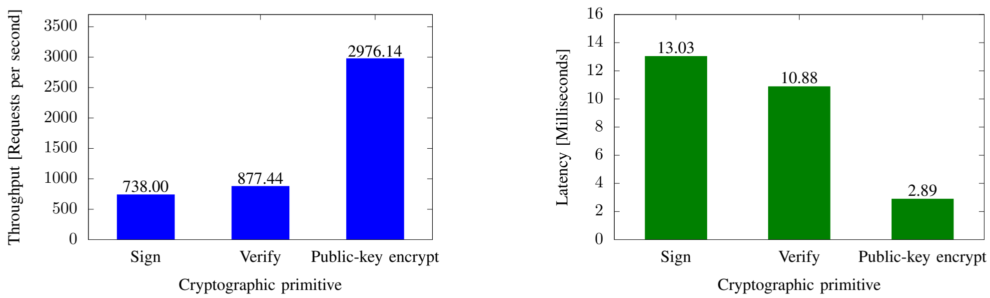

Fig. 12: Throughput and latency of different cryptographic operations in a Workstation with 2 CPU cores.

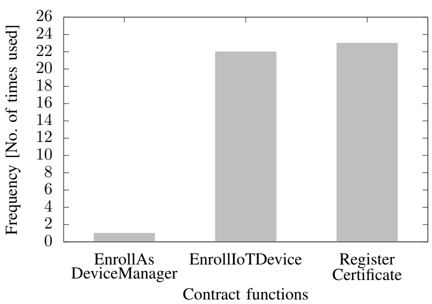

Fig. 13: Frequency of use of smart contract functions in typical use.

functions in typical use. Based on the statistics, we take 22 IoT devices, 2 device managers (so that just in case if one goes offline, the other manager would be available to manage the IoT devices), and 1 interface for typical use. The frequency of use of "$EnrollAsDeviceManager$", "$EnrollIoTDevice$", and "$RegisterCertificate$" would be 1 per device manager, 22 per device manager, and 23 (1 for the interface and 22 for the devices) respectively as shown in Fig. 13. The frequency of use of other contract functions would vary depending on the requirements.

B. Study of scalability of the proposed scheme

In the proposed scheme, DM performs different cryptographic operations at different levels. On the other hand, Int performs different cryptographic operations during registration of certificate for Int. It is essential to study the cryptographic complexities of various cryptographic operations in the respective hardware. As the first step of this study, the cryptographic operations performed by DM and Int and the necessary hardwares are identified. The different cryptographic operations performed by DM are hash, private-key decrypt, sign and verify. Whereas, the different cryptographic operations performed by Int include public-key encrypt, sign and verify. Since DM is a lightweight device, the cryptographic complexities of the corresponding operations are studied in Raspberry Pi 3. The hardware requirement for Int is more than desktop computers because Int has to serve more number of IoT devices than DM. Hence, the cryptographic complexities of the respective operations are studied in a workstation with 2 CPU cores.

To implement the cryptographic operations, Node.js library "Crypto" is used. Crypto library [25] provides support for most of the cryptographic operations and standards. However, it does not support ECC-based encryption and decryption. Therefore, the cryptographic standards used for hash, public-key encrypt, private-key decrypt, sign and verify are "SHA256", "RSA 3072-bit encryption", "RSA 3072-bit decryption", "ECDSA 256-bit sign" and "ECDSA 256-bit verify". "RSA 3072-bit encryption" is used to achieve the security level as that of "ECC 256-bit encryption". However, using RSA standard of this key size degrades the performance of the scheme. Alternatively, the performance of the scheme can be improved by choosing RSA standard of less key size. Express module [26] is used to host a server at localhost in Raspberry Pi 3 and workstation which expose the necessary endpoints for the cryptographic operations. Autocannon module [27] is used to compute the cryptographic complexities in Raspberry Pi 3 and workstation in terms of throughput and latency.

The throughput and latency parameters of different cryptographic operations in Raspberry Pi 3 and workstation are presented in Fig. 11 and 12. For computation of the above parameters using autocannon, the number of client connections is set to 10 and duration is set to 30 seconds. It is evident from Fig. 11 that, the throughput value is at its highest for hash operation and is at its lowest for private-key decrypt operation. Consequently, the latency values are the lowest and highest for hash and private-key decrypt operations respectively. Sign and verify operations have reasonable throughput and

TABLE IV: The number and type of cryptographic operations performed by the loaded entities in different scenarios.

Scenario	Loaded entity	Loading entity	Process	Cryptographic primitive				
				Hash	Public-key encrypt	Private-key decrypt	Sign	Verify
1	DM	Int	Register certificate	1	-	1	1	1
2	DM	IoT device	Register certificate	1	-	1	-	1
3	DM	IoT device	Add capability	1	-	1	-	-
4	IoT device	IoT device	Authentication & access control	1	1	-	-	1

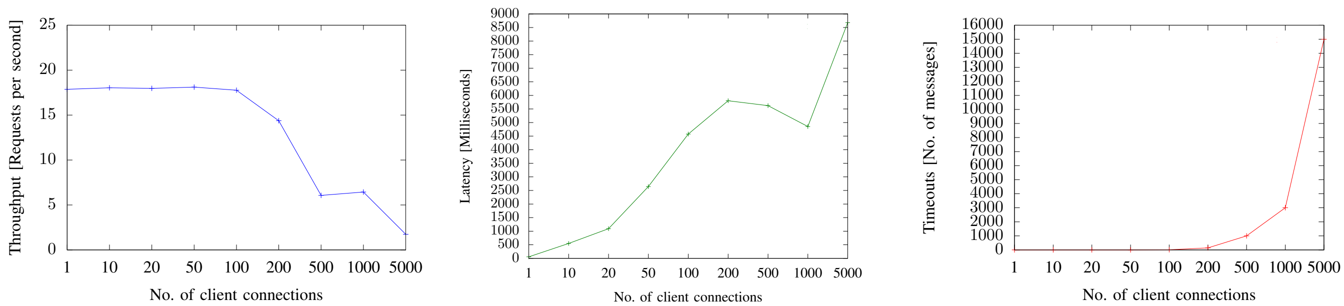

Fig. 14: Throughput, latency, and timeouts in a DM for varying number of Int or IoT devices requesting "Register certificate" operation.

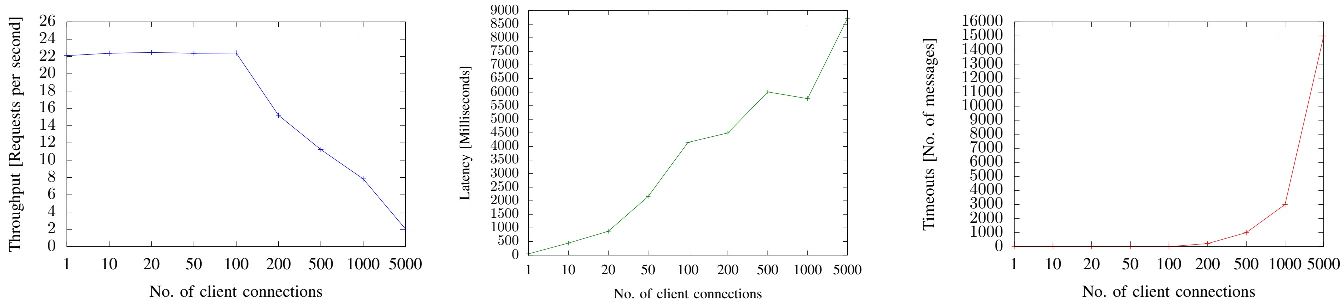

Fig. 15: Throughput, latency, and timeouts in a DM for varying number of IoT devices requesting "Add capability" operation.

latency values. It can be observed from Fig. 12 that, public-key encrypt operation has the highest throughput and lowest latency. As is usual, sign and verify operations have acceptable throughput and latency values. The scalability feature of the proposed scheme in different scenarios comprising different cryptographic operations at these cryptographic complexities is benchmarked.

DM and IoT device are loaded with cryptographic operations due to requests from Int and IoT devices in different scenarios of the proposed scheme. Table IV presents these details along with the type and number of operations performed by the loaded entities. As seen from the table, there are 4 different scenarios: 1) Load on a DM due to an Int requesting "Register certificate" operation, 2) Load on a DM due to an IoT device requesting "Register certificate" operation, 3) Load on a DM due to an IoT device requesting "Add capability" operation and 4) Load on an IoT device due to another IoT device participating in authentication and access control. For each scenario, the number of loading entities is gradually increased and the scalability of the loaded entity is investigated in terms of throughput, latency and timeouts. These simulations are carried out in Raspberry Pi 3 hardware since the loaded entity is either DM or IoT device. In these simulations, "Crypto" library, Express and Autocannon modules are used.

Fig. 14 presents the simulation results for "Scenarios 1 and 2". Throughput appears to be 19 reqs/s up to 100 client connections after which it drops. Latency appears to be continuously rising as the number of client connections increases. Timeouts appear to have started when throughput has begun to fall. Timeouts is as high as 15k when the number of client connections is 5000. From these results, it is clear that a DM can decently tolerate upto 50 Int or IoT device connections in view of these scenarios.

Fig. 15 depicts the simulation results for "Scenario 3". Throughput stays constantly at 22 reqs/s up till 100 client connections. Thereafter it descends. The trend of latency and timeouts in this scenario is the same as the trend in Scenarios 1 and 2. A DM can graciously tolerate upto 50 IoT device connections in this scenario. The simulation results for "Scenario 4" are illustrated in Fig. 16. Throughput is 300 reqs/s or above even up to 5000 client connections. This is because the scenario has only public-key encryption whose cryptographic complexity is less than that of sign, verify and private-key decrypt operations. As is usual, latency increases with increase in the number of client connections. The maximum timeouts in this scenario is 9k which is comparatively less than the maximum timeouts in other scenarios. In this scenario, an IoT device can considerately support upto 500 IoT device connections. The scalability of the proposed scheme can be greatly improved by employing ECC 256-bit encryption using Curve25519 instead of RSA 3072-bit encryption. At present, libraries do not support ECC-based encryption.

As a part of the scalability study, the proposed scheme's

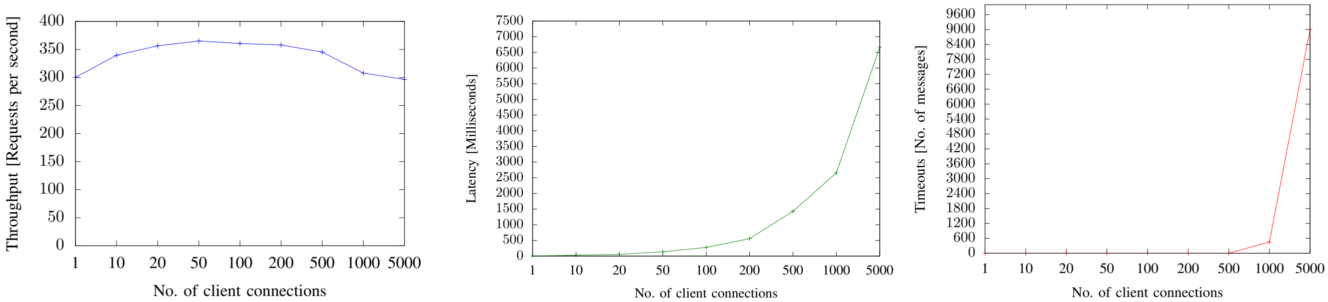

Fig. 16: Throughput, latency, and timeouts in an IoT device for varying number of IoT devices participating in authentication and access control processes.

scalability for a typical use in Ethereum Mainnet is examined. According to Etherscan, the number of blockchain transactions processed per second in Ethereum Mainnet on an average is 12 [17]. The frequency of use of the smart control functions in typical use is obtained from Fig. 13. The number of blockchain transactions in a typical use would be 69 (1×2 + 22×2 + (1 + 22)). It takes 6 seconds to process these transactions in the Mainnet. The proposed scheme would scale reasonably well for substantial number of IoT devices since an average of 1050K blockchain transactions are processed per day in the Mainnet. It is less likely that delays would be introduced in the Mainnet in typical use. However, if delays are introduced, the proposed scheme prioritizes revocation operations (if any) over the other contract operations to prevent possible malicious activities in the scheme.

C. Study of different overhead in the proposed scheme

1) Storage overhead: All the blockchain transactions initiated in the proposed scheme, after successful validation by miners, are stored in the miners. Hence, it is essential to study the storage overhead in miners. The storage overhead stage-wise is presented in Fig. 17. The total storage overhead in miners due to offline stages is 5. Whereas, the aggregate overhead in miners owing to online stages is 1. The required blockchain data is queried from the miner wherever possible using $Query()$ method instead of initiating blockchain transactions. This ensures that the number of blockchain transactions in the scheme is kept to a minimum. Consequently, the storage overhead in the miners is less.

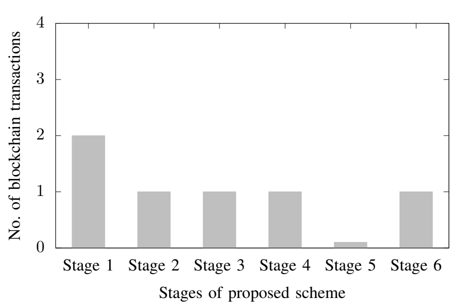

Fig. 17: Storage overhead in miner.

2) Computational and communication overhead: The computational and communication overhead of the proposed scheme are studied in this section. Only the messages exchanged in the online stages in the proposed scheme are considered for the overhead study. Let $T_{SIGN/VER}$, $T_{ENC/DEC}$, and T_H denote the overhead to execute ECC-based digital signing and verification, public-key encryption and decryption using ECC, and hashing respectively. The time complexities of ECC-based digital signing and verification (using Curve25519), public-key encryption & decryption using ECC (Curve25519), and hashing (SHA256) for $n - bits$ are deemed $\mathcal{O}(n)$. Hence, the time complexity of the proposed scheme is $\mathcal{O}(n)$. The number of messages exchanged in the proposed scheme is 5 ($\{Cert_{D1}, T, \sigma_5\}, \{Cert_{D2}, T, \sigma_6\}, C, Cap_{D1}, Cap_{D1}$) which is quite acceptable. The size of the longest message exchanged $\{Cert_{D1}, T, \sigma_5\}$ or $\{Cert_{D2}, T, \sigma_6\}$ is 1616 (1000+104+512) bits. The requirement for a slightly higher network bandwidth for the longest message is complemented by the better security features of the proposed scheme.

D. Comparison of features

Table V presents the comparison of security features of different schemes. It is evident from the table that the proposed scheme has better security features compared to [4, 5] [9–11]. More specifically, it is secure against repudiation, traceability, private-key compromise, information disclosure, impersonation, DoS and collusion attacks. Thus, the proposed scheme offers the necessary level of security. The schemes [6–8] do not have most of the security features considered for comparison.

TABLE V: Comparison of security features of different schemes.

Security feature (Resistance to)	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	Proposed
Repudiation attack	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes
Traceability	No	No	Yes	Yes	No	Yes	No	No	Yes
Private-key compromise attack	Yes	NA	NA	NA	NA	No	No	No	Yes
Information disclosure attack	No	Yes	No	No	No	No	Yes	No	Yes
Impersonation attack	Yes	Yes	No	No	No	Yes	Yes	Yes	Yes
DoS attack	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes
Collusion attack	No	Yes	No	No	No	NA	Yes	NA	Yes

* NA : Not Applicable.

Table VI shows the scalability comparison of the proposed scheme with [6, 7]. It can be observed that the proposed scheme's throughput in the scenario "Access a resource of an IoT device" is 300 requests/sec for up to 5000 client connections in the simulation hardware raspberry pi 3. This is quite comparable with the throughput of [6, 7] where the experiments are conducted in a desktop computer with Intel Core i7-950@3.07 GHz processor and 16GB of RAM.

TABLE VI: Comparison of scalability of the proposed scheme with Novo's [6, 7] work.

Scheme	Simulation hardware	Throughput (requests/sec)
[6, 7]	Desktop computer with Intel Core i7 processor and 16GB of RAM	400
Proposed	Raspberry Pi 3	300

* Scenario : "Access a resource of an IoT device". Client connections: 5000.

The comparison of the features namely scalability, usability, and interoperability of various schemes are provided in Table VII. It can be seen that the proposed scheme is scalable, usable, and interoperable while maintaining the necessary level of security.

TABLE VII: Comparison of the features of various schemes.

Feature	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	Proposed
Scalability	Yes	Yes	Yes	Yes	No	ND	No	Yes	Yes
Usability	Yes	No	Yes	Yes	Yes	Yes	Yes	No	Yes
Interoperability	No	No	Yes	Yes	No	No	Yes	Yes	Yes

* ND : Not Done.

Thus, the proposed scheme is better than the closely related existing schemes.

VII. Conclusion

In this paper, a new blockchain-enabled authentication and access control scheme for IoT is presented. The scheme uses blockchain as the root-of-trust. The scheme stores the necessary information for authentication and access control in the blockchain. The scheme exhibits resilience to different attack vectors in IoT namely repudiation, traceability, private-key compromise, information disclosure, impersonation, DoS, and collusion attacks. The experimental results indicate that the transaction costs for all the blockchain transactions are well within the recommended gas limit of 3000000 gas. Moreover, the transaction costs of contract functions in Ethereum Mainnet are fairly less than $3. Scalability study demonstrates that the scheme is appreciably scalable in all the different scenarios considered. The storage overhead statistics indicate that the number of blockchain transactions in the scheme is kept to a minimum. The scheme's computational and communication overhead are fairly acceptable. Most of all, the scheme has better features compared to the recent and closely related existing schemes. One of the future enhancements would be to monitor the device managers for any misbehaviour since they possess many sensitive information in the scheme. The smart contract deploy node, having the highest privilege and trust, may monitor the device managers for any misbehaviour and take necessary actions.

References

[1] S. R. Department, *Internet of Things - number of connected devices worldwide 2015-2025*, Accessed: Jan. 2022. [Online], https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/.

[2] V. Mohindru and A. Garg, "Security attacks in Internet of Things: A review," in *Recent Innovations in Computing. ICRIC 2020. Lecture Notes in Electrical Engineering*, vol. 701, 2020, pp. 679—-693.

[3] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, 2008.

[4] C. Lin, D. He, X. Huang, K. K. R. Choo, and A. V. Vasilakos, "BSeIn: A blockchain-based secure mutual authentication with fine-grained access control system for Industry 4.0," *Journal of Network and Computer Applications*, vol. 116, pp. 42–52, 2018.

[5] R. Xu, Y. Chen, E. Blasch, and G. Chen, "BlendCAC: A smart contract enabled decentralized capability-based access control mechanism for the IoT," *Computers*, vol. 7, no. 3, pp. 1–27, 2018.

[6] O. Novo, "Blockchain meets IoT: An architecture for scalable access management in IoT," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1184–1195, 2018.

[7] O. Novo., "Scalable access management in IoT using blockchain: A performance evaluation," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4694–4701, 2019.

[8] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, "Smart contract-based access control for the Internet of Things," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1594–1605, 2019.

[9] B. Bera, S. Saha, A. K. Das, and A. V. Vasilakos, "Designing blockchain-based access control protocol in IoT-enabled smart-grid system," *IEEE Internet of Things Journal*, vol. 8, no. 7, pp. 5744–5761, 2020.

[10] S. N, V. B. K, and M. Rajarajan, "Blockchain-based scheme for authentication and capability-based access control in IoT environment," in *2020 11th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, 2020, pp. 0323–0330.

[11] Y. E. Oktian and S. Lee, "BorderChain: Blockchain-based access control framework for the Internet of Things endpoint," *IEEE Access*, vol. 9, pp. 3592–3615, 2021.

[12] S. Hernan, S. Lambert, T. Ostwald, and A. Shostack, "Uncover security design flaws using the STRIDE approach," *MSDN Magazine*, 2006.

[13] D. Dolev and A. Yao, "On the security of public-key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198—-208, 1983.

[14] L. Dykcik, L. Chuat, P. Szalachowski, and A. Perrig, "BlockPKI: An automated, resilient, and transparent public-key infrastructure," in *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, 2018, pp. 105–114.

[15] M. Y. Kubilay, M. S. Kiraz, and H. A. Mantar, "CertLedger: A new PKI model with certificate transparency based on blockchain," *Computers & Security*, vol. 85, pp. 333–352, 2019.

[16] J. Shi, X. Zeng, and R. Han, "A blockchain-based decentralized public-key infrastructure for information-centric networks," *Information*, vol. 13, no. 5, pp. 1–15, 2022.

[17] *Ethereum mainnet*, Accessed: Oct. 2022. [Online], https://etherscan.io/.

[18] *Goerli testnet*, Accessed: Sep. 2022. [Online], https://goerli.etherscan.io/.

[19] *Sepolia testnet*, Accessed: Sep. 2022. [Online], https://sepolia.etherscan.io/.

[20] *Goerli faucet*, Accessed: Sep. 2022. [Online], https://goerlifaucet.com/.

[21] *Sepolia faucETH*, Accessed: Sep. 2022. [Online], https://faucet.sepolia.dev/.

[22] *The Merge*, Accessed: Sep. 2022. [Online], https://ethereum.org/en/upgrades/merge/.

[23] Deloitte, *2022 Connectivity and Mobile Trends Survey (third edition)*, Accessed: Oct. 2022. [Online], https://www2.deloitte.com/us/en/pages/about-deloitte/articles/press-releases/connectivity-and-mobile-trends.html.

[24] S. R. Department., *Average number of internet-connected devices in households in Australia 2020-2025*, Accessed: Oct. 2022. [Online], https://www.statista.com/statistics/1202887/australia-average-number-of-internet-connected-devices-per-household/.

[25] *Crypto*, Accessed: Feb. 2022. [Online], https://nodejs.org/api/crypto.html.

[26] T. Holowaychuk, *Express: Fast, unopinionated, minimalist web framework for Node*, Accessed: Feb. 2022. [Online], https://www.npmjs.com/package/express.

[27] M. Collina, *Autocannon: A HTTP/1.1 benchmarking tool written in Node, greatly inspired by Wrk and Wrk2, with support for HTTP pipelining and HTTPS*, Accessed: Feb. 2022. [Online], https://www.npmjs.com/package/autocannon.

Sivaselvan N received his M.E degree from Anna University Tiruchirapalli. He is currently pursuing the Ph.D degree in Information Engineering with the Department of Electrical and Electronic Engineering, City, University of London, UK. He is working as an Assistant Professor with the Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal. His research interests include Blockchain and Internet-of-Things security. He has published papers in reputed International conferences and journals.

Vivekananda Bhat K (SM'19) received the Ph.D. degree in computer science and engineering from IIT Kharagpur, India, and the M.Tech. degree in systems analysis and computer applications from the National Institute of Technology Karnataka, Surathkal, India. He is currently an Associate Professor with the Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, India. He has published several articles in reputed Web of Science indexed international journals and conferences. His research interests include cryptology and cyber security.

Muttukrishnan Rajarajan is currently a Professor of security engineering with the City, University of London, U.K., where he currently leads the Information Security Group. He is the Director of Institute for Cyber Security at City. He is a Visiting Researcher with the British Telecommunication's Security Research and Innovation Laboratory. His research interests include privacy-preserving data analytics, cloud computing, the Internet of Things security, and wireless networks. He has published well over 300 articles and continues to be involved in the editorial boards and technical programme committees of several international security and privacy conferences and journals. He is an Advisory Board Member of the Institute of Information Security Professionals, U.K., and acts as an advisor to the U.K. Government's Identity Assurance Programme.

Ashok Kumar Das (Senior Member, IEEE) received a Ph.D. degree in computer science and engineering, an M.Tech. degree in computer science and data processing, and an M.Sc. degree in mathematics from IIT Kharagpur, India. He is currently an Associate Professor with the Center for Security, Theory and Algorithmic Research, IIIT, Hyderabad, India, and also a visiting faculty with the Virginia Modeling, Analysis and Simulation Center, Old Dominion University, Suffolk, VA 23435, USA. His current research interests include cryptography, system and network security including security in smart grid, Internet of Things (IoT), Internet of Drones (IoD), Internet of Vehicles (IoV), Cyber-Physical Systems (CPS) and cloud computing, intrusion detection, blockchain and AI/ML security. He has authored over 335 papers in international journals and conferences in the above areas, including over 290 reputed journal papers. He was a recipient of the Institute Silver Medal from IIT Kharagpur. He has been listed in the Web of Science (Clarivate™) Highly Cited Researcher 2022 in recognition of his exceptional research performance. He was/is on the editorial board of IEEE Systems Journal, Journal of Network and Computer Applications (Elsevier), Computer Communications (Elsevier), Journal of Cloud Computing (Springer), Cyber Security and Applications (Elsevier), IET Communications, KSII Transactions on Internet and Information Systems, and International Journal of Internet Technology and Secured Transactions (Inderscience). He also served as one of the Technical Program Committee Chairs of the first International Congress on Blockchain and Applications (BLOCKCHAIN'19), Avila, Spain, June 2019, International Conference on Applied Soft Computing and Communication Networks (ACN'20), October 2020, Chennai, India, and second International Congress on Blockchain and Applications (BLOCKCHAIN'20), L'Aquila, Italy, October 2020. His Google Scholar h-index is 74 and i10-index is 213 with over 15,400 citations.