COMPUTERS & SECURITY 86 (2019) 318–334

Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Blockchain based permission delegation and access control in Internet of Things (BACI)

Gauhar Ali[a], *Naveed Ahmad*[a], *Yue Cao*[b,1,*], *Muhammad Asif*[c], *Haitham Cruickshank*[d,1], *Qazi Ejaz Ali*[a]

[a] *Department of Computer Science, University of Peshawar, Pakistan*
[b] *School of Transportation Science and Engineering, Beihang University, China*
[c] *Department of Electronic, University of Peshawar, Pakistan*
[d] *Institute of Communication Systems, University of Surrey, GU2 7XH, UK*

ARTICLE INFO

Article history:
Received 4 January 2019
Revised 28 May 2019
Accepted 18 June 2019
Available online 19 June 2019

Keywords:
Blockchain
Internet of Things
Access control
Permission delegation
Platform verification

ABSTRACT

Access control with permission delegation mechanism allows fine granular access to secure resources. In the literature, existing architectures for permission delegation and access control are either event-based or query-based. These previous works assume a single trusted delegation service, which however is likely biased or fails to service. Also, they fail to allow users to verify delegation service operations, as such cannot be directly applied to IoT (Internet of Things) due to low power, low-bandwidth, ad-hoc and decentralized nature. This paper proposes a novel decentralized architecture for permission delegation and access control for IoT application, with demands on event and query base permission delegation. We further apply Blockchain (BC) technology to make delegation services secure, trusted, verifiable and decentralized. We investigate our proposed approach in Simple PROMELA INterpreter (SPIN) model checker using PROMELA (Process Meta Language). The "Platform Verification", "Delegation", "Mutual Exclusion" properties written in Linear Temporal Logic (LTL) are also verified against the PROMELA model.

1. Introduction

Internet of Things (IoT) (Al-Fuqaha et al., 2015) facilitates a massive number of sensors embedded devices, with the ability to collect and transfer data to each other without human interaction. The Gartner (Rivera and van der Meulen, 2014) predicts that, around 25 billion devices will be connected in IoT at the end of the year 2020. Since anything is accessible from anywhere at any time in IoT, such nature raises new security challenges (Granjal et al., 2015) as a lot of attacks (Abomhara et al., 2015) may happen on the IoT devices and the data in transit. In IoT, an intruder can reach critical online data through a refrigerator. However, traditional access control mechanisms cannot be directly applied to IoT, due to low power, low-bandwidth, ad-hoc and decentralized nature of IoT network. Therefore, IoT needs a comprehensive, secure permission delegation and access control mechanisms.

Access control is a mechanism that limits the operations or activities of a legitimate user (Sandhu and Samarati, 1994). The access control functionalities increase with incorporation of delegation module, where the delegation is a process for assigning temporary permissions to a user (Wang et al., 2008). For example, a delegator can behave as a person who

* Corresponding author.
E-mail addresses: gauharstd@uop.edu.pk (G. Ali), n.ahmad@uop.edu.pk (N. Ahmad), yue.cao@lancaster.ac.uk (Y. Cao), m.asif@uop.edu.pk (M. Asif), h.cruickshank@surrey.ac.uk (H. Cruickshank), qaziejazali@uop.edu.pk (Q.E. Ali).
[1] Member of IEEE.
https://doi.org/10.1016/j.cose.2019.06.010
0167-4048/

transfers his permissions to another person. The person, who receives these permissions, is called the delegatee. A user delegates permissions in the response of either a query or an event. In event-based permission delegation, permissions are delegated to a specific user in the response of an event[2]. In query-based permission delegation, the user requests for permission on a resource from the owner.

In IoT, devices like sensors, cameras are used to detect events, e.g., fire, car accident and traffic jam. In the response of an event, relevant permissions must be delegated to concerns personally. For example, in response to a fire event in a house, the system/administrator delegates permissions on smart door locks and smart electric equipments to rescue people. Similarly, IoT devices generate a query for accessing a protected resource. In response to a query, the system/administrator delegates permissions on the resource to the requester. For example, a physician from another hospital requests for accessing a patient record. The internal physician delegates permission on patient record to the external physician. Therefore, IoT demands both event and query base permission delegation.

In IoT, the access control with delegation service has many challenges:

- A huge number of interconnected devices.
- The lack of trust between IoT devices.
- Usually delegation operations are performed in distributed fashion because each user has a certain control on his own permission delegation.

In the existing literature, a single trusted delegation service for enforcing delegation authorization rules is applied. Here, if the delegation service fails, then no permission delegation will occur. The centralized delegation service may be biased, while permitting illegal delegations and denying legal delegations. Therefore, a solution is required to develop trust amongst partners, meanwhile without a trusted third party.

The Blockchain (BC) is an incorruptible digital ledger of economic transactions, that can be programmed to record not just financial transactions, but also virtually everything of value (Tapscott and Tapscott, 2016). By applying BC, above challenges could be solved:

- BC demolishes the role of the trusted third party, since it provides unbiased delegation service. Fundamentally, BC stores a copy of every transaction executed in a system, and this copy is available to all nodes linked with BC.
- BC provides uninterrupted delegation service, because of distributed and timely replicated transactions ledger. For example, even if a number of BC nodes lost their copy of transaction detail due to attack against BC, this situation will not interrupt the delegation service. This is because that each node in BC can still synchronize by replication from other nodes still keeping the copy.
- In the absence of a controlling entity, the consensus mechanism of BC helps to develop trust among nodes. As each block is only valid with Proof-of-Work (PoW). To validates a block, a node sends its own block, of which the PoW will be validated by other miners receiving the block.

Our proposed novel architecture, BACI is compatible to both event and query base permission delegations and access control. It initially, assigns a node to a minimum permission group, and if necessary additional permissions are delegated to a node in the response to a query or event. For example, if a traffic jam event occurs, the permission of traffic signal light and lane control are delegated to nearby traffic wards. BACI provides decentralized BC based delegation and platform behavior verification service, and alleviates the case if some of nodes malfunction in the chain. In our design, any user can verify any others' delegated resources while still preserving user privacy. Our contributions are as follows:

- We analyze permission delegation at access control/authorization level in IoT, and propose BACI architecture. The novelty comes from event as well as query-based permission delegations.
- We integrate BC into BACI architecture, this alleviates the need for centralized trusted delegation service. In the absence of a controlling entity, our architecture uses BC consensus mechanism to develop trust among IoT devices.
- To prevent a legitimate users from misuse of delegated permission, the BC manager of in BACI architecture performs platform verification at the time of permission activation. BC manager has access to the database holding binary hash values of the IoT devices and users.
- We develop a model of the proposed approach using PROMELA. The SPIN model checker (Holzmann, 1997) is used to analyze our PROMELA model. SPIN validates the "Platform Verification", "Delegation", "Mutual Exclusion" LTL (Gerth et al., 1995) properties against the PROMELA model.

The rest of the paper is organized as follows. In Section II, we introduce related works. In Section III, we discuss use cases. In Section IV, we present BACI architecture and its core components. In Section V, we present formal modeling of our proposed architecture. Section VI presents verification results, followed by conclusion in Section VII.

1.1. *Blockchain (BC)*

The BC concept was first introduce by Nakamoto (2008), where Bitcoin is the first implementation. The BC is based on private key cryptography, peer-to-peer network and BC protocols, with the following features:

1.1.1. *Decentralized computing*

There is no central controlling authority, where a consensus protocol is used to validate a transaction.

1.1.2. *Shared and distributed ledger*

All transactions are stored in a shared ledger. Each node maintains a copy of its own ledger, while copies of other peers are synchronized by replication.

[2] For example, if a car accident event is reported in Intelligent Transportation System (ITS), then the permission of traffic signal light and lane control are delegated to nearby traffic warden.

1.1.3. *Transparency*
All transactions are stored, and available to all peers in the BC.

1.1.4. *Fault tolerant peer-to-peer network*
All BC miners process transactions in parallel. If some of BC miners fail to work, the network is still able to process transactions.

1.1.5. *Security*
Blocks are included in the BC after validation. The block cannot be copied, deleted or updated, if it has been cryptographically sealed. To hack a block, an attacker would have to need 51% of network computing power.

1.2. *How BC works*

BC consists of a number of blocks. The first block in BC is called genesis block. Every block contains a hash of the previous block, whereas the genesis block does not contain hash of any block. Each block contains PoW, previous block hash, block header and a set of transactions. Suppose someone requests a transaction, the transaction is broadcasted to all miners. Each miner tries to validate the transaction by solving a complex mathematical puzzle. After verification, the transaction becomes a part of the block, then the block is added into BC.

Here, miners are resource-intensive BC nodes. Miners validate a transaction and add it to the block. On average, in every 10 minutes, a new block is added to the BC. Miners compete to find a PoW by solving a complex mathematical problem. More computing power, increases the chance of winning the competition. Other miners verify the PoW when they received the block. Miners spend their resource to solve a mathematical problem, so they receive a reward in the form of transaction fee.

BC uses a consensus algorithm to allow BC nodes to agree on a single value. The miners use consensus algorithm to ensure trust in trustless BC network. Some of the consensus algorithm are PoW, Proof-of-Stack (PoS). PoW is a requirement to define a mathematical problem which is difficult to solve but easy to validate. Miners need more computational power to solve the mathematical problem. The first miner who solve the problem gets the reward. PoS uses an election process to choose one node to validate the next block. To become a validator, a node have to deposit certain amount of money called stack. A validator is chosen base on its stack size. In PoS miners are called validators. The validator does not mine a block instead it mint a block.

1.3. *Smart contract*

The smart contracts are computer programs that aid in the transfer of money or anything having value. These programs run automatically, when a specific policy is met. Each smart contract consists of a contract address, private storage, and predefined functions. Ethereum (Buterin et al., 2014), the second generation of BC, is an decentralized and open-source platform that executes smart contracts (Clack et al., 2016a). Ethereum platform uses solidity programming language (web, 2014) to build smart contract. Here, solidity is a Turing complete language used to develop decentralized application. Ethereum uses a cryptocurrency called ether. Ethereum platform allows to create two types of accounts, i.e., externally owned account and contract account. Externally owned account sends transactions whereas contract account executes contract code with a transaction is received. Contract account is used to deploy smart contract. Ethereum uses a revised version of original PoW consensus algorithm. Some of the other platforms the run smart contracts are Rootstock[3], Hyperledger[4] and cardano[5].

1.4. *Blockchain challenges*

BC is a peer-to-peer network. The following are some of the possible attacks on BC peer-to-peer network.

1.4.1. *Denial of Service (DoS) attacks*
In DoS attack, the attacker malfunctions a node by flooding a huge traffic. It prevents a legitimate users from accessing the resource or service. Similarly, another type of attack called Distributed Denial of Service (DDOS) attack overwhelms a node with malicious requests. Unlike DOS, in DDOS several attackers target a single node.

1.4.2. *Sybil attacks*
In sybil attack, a node uses multiple identities to compromise a major portion of the network. Moreover, the attacker lunches 51% attack, if it gains more than 50% computing power of the whole BC network.

1.4.3. *Eclipse attacks*
In eclipse attack (Heilman et al., 2015), the attacker isolates specific node from the peer-to-peer network. Like sybil attack, it does not attack the entire network. Once the target node is isolated, the attacker controls all outgoing connections of the node. Thereafter, the attacker can exploit the target node and launch different types of attacks on BC mining power and consensus mechanism. These attacks include double spending, engineering block races, selfish mining and splitting mining power.

1.4.4. *Routing attacks*
In routing attack, the attacker intercepts a message in the BC network. The attack tampers the message and forwards it to their neighbors. Furthermore, the routing attack is divided into partitioning attack and delay attack. In partitioning attack, the entire BC network is divided into two or more portions. Whereas, in delay attack, the attacker captures the message and tamper with it. Then, it redirects the tamper message to another BC network portion.

BC uses consensus mechanism to develop trust among BC nodes. The following are some of the possible attacks on BC consensus mechanism.

[3] [Online]. Available: http://www.rsk.co.
[4] [Online]. Available: https://www.hyperledger.org.
[5] [Online]. Available: https://www.cardano.org/en/home/.

1.4.5. *51% attacks*

A miner may lunch 51% attack, if it gains more than 50% hashing power of the whole BC network. After successfully lunching of 51% attack, the attacker can block new transactions confirmations. Moreover, he can reverse already confirmed transactions.

1.4.6. *Double spending*

In double spending, a user performs multiple transactions with the same cryptocurrency. A user transaction is broadcasted to all the nodes in the network. These nodes need to receive and confirm the transaction, which takes time. The attacker can exploit the intermediate time between two transactions initiation and confirmation to quickly launch an attack.

1.4.7. *Alternative history attacks*

In alternative history attack, the attacker sends a transaction to the merchant. Meanwhile, the attacker includes the double spending transaction in an alternative BC fork. The merchant sends the product after n blocks confirmation. So, the attacker tries to find more than n blocks. If the attacker succeed, he gains his coins by releasing the fork.

1.4.8. *Race attacks*

In the race attack, the attacker creates two transactions. The attacker sends the first transaction to the merchant. The merchant sends the product without transaction confirmation. Meanwhile, the attacker broadcasts the second transaction to invalidate the first transaction.

1.4.9. *Finney attack*

In the finney attack, the attacker uses two identical transactions i.e., one transaction crediting the target and the other crediting the attacker himself. The attack mined a block which include the first transaction and delay its publishing. Meanwhile, the attacker starts mining the second transaction. when the attacker succeed, he purchases goods with the first transaction. Then, he releases the pre-mined block which include the first transaction. The attacker receives both goods and coins whereas merchant finds their transaction invalid.

2. Related work

In Tapas et al. (2018), the authors have proposed a BC based authorization and delegation framework for IoT-Cloud. It allows all users to audit both access control decisions and authorization operations. BC is used to develop trust between the nodes in the peer-to-peer network. However, the authors assumed all users devices, i.e., not a member of the BC peer-to-peer network, trusted. Even legitimate user can misuse delegated permission. Therefore, BACI stores platform hashes of IoT and user devices on BC. BACI architecture performs platform verification at the time of permission activation. BC manager has access to the database holding binary hash values.

The authors in Novo (2018), have proposed a BC based architecture for access management in IoT. The architecture consists of wireless sensor networks, managers, agent node, smart contract, BC and management hub. The IoT devices query information through "management hub" from smart contract stored on BC. The smart contract after validation of certain policy provides the requested information. The authors in Maesa et al. (2017), have proposed a BC based access control. In this approach, the owner of a resource defines access policy and publishes it on the BC. A user can access the resource if being allowed through policy. The user can further transfer the rights to another user through a transaction in BC. Any user can verify who has the right to a particular resource.

In Gattolin et al. (2018), the authors have proposed an architecture called BlAsT. It has integrated BC to gain transparency in key management. The identity provider binds user ID with the public key during registration. Then, the identity provider writes Signed Tree Root (STR) in BC. The STR consists of security policy, current and previous epoch, current and previous Merkle Root and the ECDSA signature. The STR transactions are chained together to form the STR-chain. BlAsT allows the clients to validate their data stored at identity provider, by retrieving STR from the BC. In Novo (2018), Maesa et al. (2017) and Gattolin et al. (2018), the authors did not discuss permission delegation which to provide additional functionalities for access control in IoT. Moreover, in Gattolin et al. (2018), additional computations are required while validating the data in read operation. Therefore, the proposed solution is not suitable for IoT devices.

In Gusmeroli et al. (2012), the authors have proposed a capability-based access control for IoT. Initially, the subject is assigned with the least amount of permission. Additional permissions are delegated to the subject on a particular object, using the concept of capability. Authors have defined capability as an object with a set of permissions. In Ouaddah et al. (2017), authors have proposed a FairAccess framework based on BC. It is a decentralized and privacy preserving access control framework for IoT. The owner of a protected resource defines an access policy for the requester. The owner uses "GrantAccess" transaction to store the policy at the BC. The requester uses "GetAccess" transaction to access the resource. The requester can further delegates access rights using "DelegateAccess" transaction. In Xu et al. (2018), the authors have proposed a decentralized BC based access control for IoT, called BlendCAC. A capability token is used for the registration, delegation and revocation of access rights. The capability token is managed through smart contract. In Gusmeroli et al. (2012), Ouaddah et al. (2017) and Xu et al. (2018), right access/delegation occurs on user query/request. There is no event-based right delegation. IoT needs both query as well as event-based architecture for access control and permission delegation. For example, if an accident event occurs then lane control and traffic signal light rights are delegated to traffic warden and rescue.

In Alam et al. (2011), the authors have proposed xDAuth framework for cross-domain access and delegation control. A trusted third-party called Delegation Service (DS) makes an authentication and authorization. The DS redirects the user for authentication to his own domain. The DS makes authorization decision based on service provider access policies after successful authentication. Here, if the DS fails, then no permission delegation will occur. The DS may be biased, while permitting illegal requests and denying legal requests. Therefore, a solution is required to develop trust amongst partners and alleviate the need for a trusted third-party.

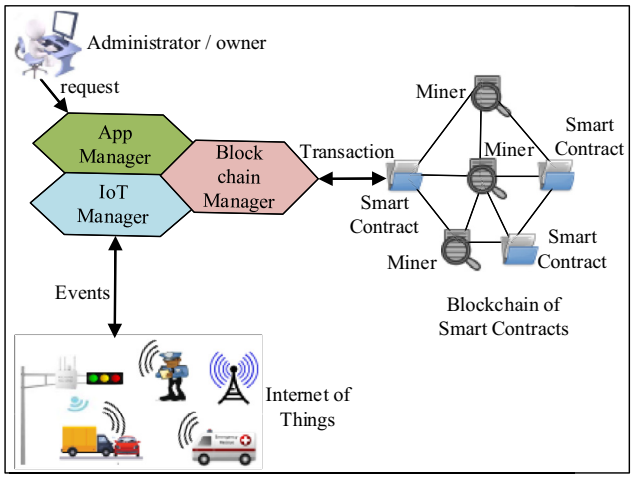

Fig. 1 – High-level BACI architecture.

The authors in Christidis and Devetsikiotis (2016), have discussed whether BC and smart contracts fit for IoT. The maintenance cost of existing centralized model is very high. Also, there is a lack of trust between devices. Therefore, IoT needs a secure, trustless and scalable model. The BC is a decentralized, trustless and secure peer to peer network. The authors have concluded that BC provides an efficient solution for IoT. The integration of BC and IoT will bring significant changes in many industries.

In Dorri et al. (2016), the authors have proposed a lightweight BC based framework for IoT. Due to resource-constrained nature of IoT devices, the authors have eliminated the concept of coin and proof of work in his approach. While the authors believe that still most of its security and privacy features are maintained. Their BC-based smart home framework consists of smart home, cloud storage and overlay components. In Dorri et al. (2017), authors have elaborated the above three core components of his BC-based smart home framework. Each smart home is managed by a local BC. These smart homes are connected to each other. Every device has a shared key for access data on the BC. Authors have assumed that all devices in a smart home are trusted. It is a very strong assumption and only sharing key authentication is not sufficient to secure a system. Our novel framework performs platform verification before permission activation. It could be possible that a genuine user after shared key authentication installs a malicious application which misuses the delegated permission.

3. Proposed architecture

Our novel proposed architecture given in Fig. 1, provides a framework for access control with permission delegation in IoT. It performs both query and event base permission delegation. BC is used to perform permission delegation and access control. The main building blocks of our proposed BACI architecture are IoT device, User device, Application manager, IoT manager, BC manager, Smart contract and BC network.

3.1. *IoT device*

IoT devices have limited processing power and memory. These IoT devices are part of a wireless sensor network. Each IoT device has a unique identity. The identity is used to uniquely identify a device in the BC network. Each device has a public and private key pair for encryption. In IoT applications, each node may hold some resources, e.g., information, services. The owner/administrator provides these resources when needed by the other nodes. An IoT device owner/administrator is a party who has the ultimate control over the IoT device. The owner creates smart contract for his resource on the BC.

3.2. *User device*

User device has higher processing power and more memory than IoT device, e.g., smart phones, laptop, PCs. A user device is used by the owner to register his IoT devices on BC. The delegatee uses “User device” to activate a resource.

IoT devices and user devices authentication is not the focus of this research work. However, in the literature Ouaddah et al. (2016), Wu et al. (2018) and Puthal et al. (2019), different authentication mechanisms have been proposed. We assume bubbles-of-trust (Hammi et al., 2018) as our proposed architecture authentication mechanism, because it has lower time and power requirements and resiliency toward different attacks.

3.3. *Application manager*

Application manager provides a user interface and registration in BACI architecture. It is consists of a query handler, event handler, alerts/warnings generator and delegation handler modules. The details tasks of application manager is shown in Fig. 2. The application manager performs the following functions.

(1) The query handler receives subscribe, unsubscribe requests from the user for a particular event. It subscribes the user for the event and stores the subscribed user information in the database. The query handler receives delegation request and forwards to delegation handler. Similarly, the query handler receives activation and revocation requests and forwards to BC manager.
(2) The delegation handler receives event and query requests from event handler and query handler, respectively. Then, the delegation handler retrieves subscribe user data from database and sends to BC manager.
(3) The event handler receives composite events after filtration and co-relation from IoT manager. Then, it sends the event to the alerts/warnings generator and delegation handler.
(4) The alerts/warnings generator receives event notification from the event handler and generates alerts/warnings to the subscribed users at the application layer.

3.4. *IoT manager*

The IoT manager consists of data filter & event co-relator and IoT interface modules. The details tasks of IoT manager is

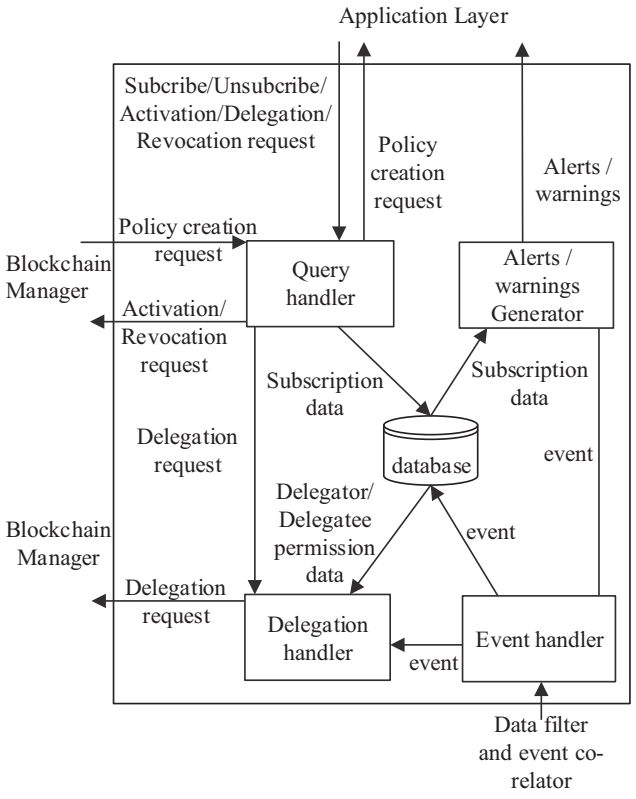

Fig. 2 – Details tasks of application manager.

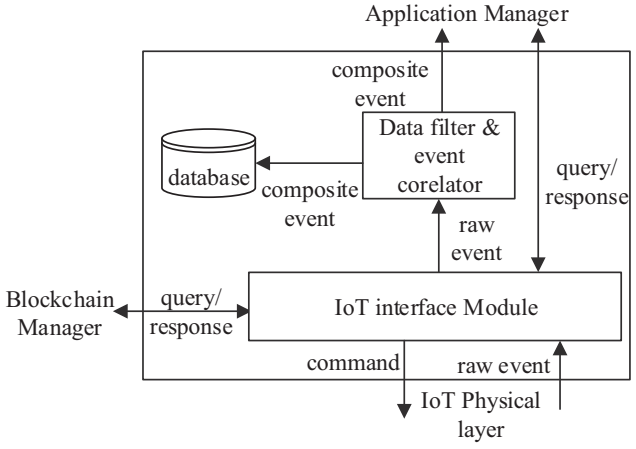

Fig. 3 – Details tasks of IoT manager.

shown in Fig. 3. The IoT manager performs the following functions.

(1) The IoT interface module sends a command to and receives data from IoT devices. It allows the subscribed user to query IoT devices and gets real-time data.
(2) The data filter & event co-relator module performs filtration on raw data generated by IoT devices. Similarly, the event co-relator produces a composite event, e.g., smoke and heat events aggregate to form a fire event. Then, it stores the events in the database and sends a notification to the application manager for alerts/warnings generation.

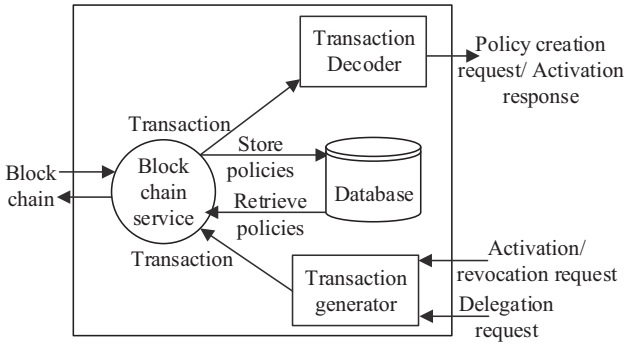

Fig. 4 – Details tasks of BC manager.

Table 1 – Permission delegations stored at smart contract.

S.No	Delegator ID	Delegatee ID	Permission
1	67a45d234f	98a45c8744	34e674f896, access
2	3a145d294c	68d45c8c24	24e674c836, read
3	57a45d2f44	13c45c8f74	39f6c4a296, read/write

3.5. *BC manager*

The IoT devices are registered by BC manager with BC. BC produces smart contract for each IoT device. User can access information stored on the BC through transactions. The BC manager accepts request from user and generates BC transaction. The details tasks of BC manager is shown in Fig. 4. The BC manager performs the following functions.

(1) On receiving request, BC manager generates "T.register" transaction to create a smart contract at BC.
(2) Allows delegatee to activate or deactivate the delegated permission using "T.activate" or "T.deactivate" transactions.
(3) Generates "T.delegate" or "T.revoke" transactions, on delegator request, to delegate or remove a permission delegation stored at smart contract.

3.6. *Smart contract*

Smart contracts are agreements or set of rules between parties stores on the BC (Clack et al., 2016b). In our architecture, the IoT devices have smart contract. User sends "T.register" transaction to create smart contract in the BC network. The smart contract stores IoT devices platform hashes and delegation policies in two different data structures. The data structure that stores delegation consist of delegator ID, delegatee ID and permission fields. It is shown in Table 1. The data structure that stores platform hashes consist of IoT device ID and hash value fields. The smart contract functions are used to access both the data structures. These functions are defined by "Permission Activation Algorithm", "Permission Deactivation Algorithm", "Event-based Permission Delegation Algorithm", "Query-based Permission Delegation Algorithm", "Policy

Table 2 – BC transactions.

Name	Transaction format	Description
T.register	$s_i, \hbar_c$	To create a smart contract at the BC.
T.publish	$s_i, s_j, p_k, \mathcal{C}$	To publish an access policy at the BC.
T.delegate$_S$	$s_i, s_j, p_k, \hbar_c$	To delegate a permission to a single user through the BC.
T.delegate$_G$	$s_i, g_j, p_k, \hbar_c$	To delegate a permission to a group of the users through the BC.
T.revoke	$s_i, s_j, p_k, pol_{si,sj}$	To revoke a publish policy from the BC.
T.activate	$s_j, p_k, \hbar_c$	To activate a permission on a smart contract store at the BC.
T.deactivate	$s_j \ggg p_k$	To deactivate a permission on a smart contract store at the BC.

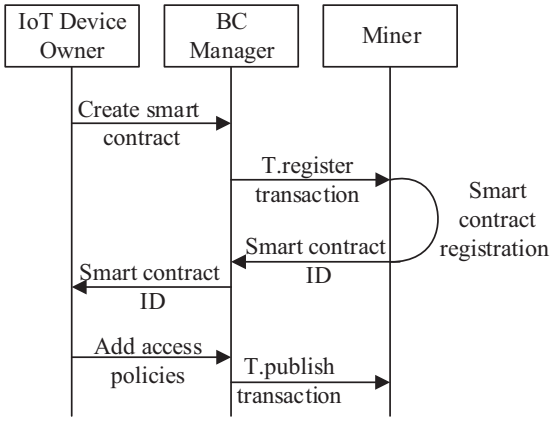

Fig. 5 – IoT device registration.

Creation Algorithm", "Permission Revocation Algorithm". These algorithms are described in section V.

IoT devices call these functions using BC transactions. The BC transactions are shown in Table 2. The transactions are recorded on a public ledger. These transactions are combined by miner into block. After validation, the block is appended to the BC.

3.7. *Smart contract registration*

At the beginning, the owner creates a smart contract for his IoT device.

(1) In step 1, the owner "s_i" sends a "create smart contract" request to BC manager as show in Fig. 5.
(2) In step 2, the BC manager generates a "T.register" transaction and forwards it to the BC network.
(3) In step 3, the BC miner, after platform verification of the IoT device, creates smart contract with a unique identifier. The BC miner returns the unique identifier to the BC manager.
(4) In step 4, the BC manager returns the unique identifier to the owner of IoT device

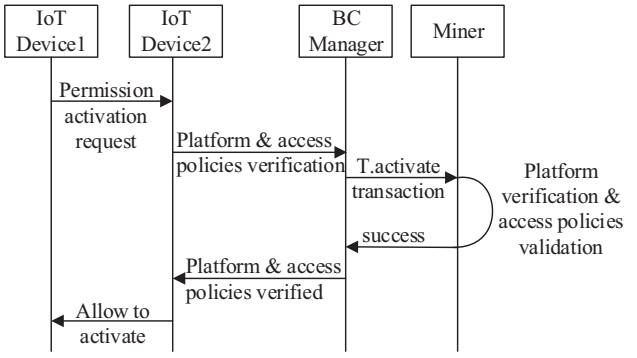

Fig. 6 – Permission activation.

(5) In step 5, the owner sends "Add access policy" request to BC manager.
(6) In step 6, the BC manager uses "T.publish" transaction to store the access policies at the smart contract as shown in Algorithm.5 of Section IV.

3.8. *BC network*

Our proposed architecture consists of a private BC. It allows each node to read but dedicated nodes to write in the distributed ledge. In ITS, Road Side Units (RSUs) can form a BC. In our architecture, IoT devices have smart contracts on BC. Due to low power and storage constraints, IoT devices like in-vehicle sensors or road-side sensors for safety and traffic management are not a part of the BC. These IoT devices communicate with BC through BC manager. The access control and delegation authorization rules are stored at smart contract. The miners in the BC network approve every transaction and save copies of the BC. Therefore, data stored at BC is decentralized and tamper-resistant. The transactions are given in Table 2 with description and transaction format.

3.9. *Delegated permission activation*

Suppose "IoTDevice1" wants to access services or data of "IoTDevice2". Fig. 6 shows how "IoTDevice1" interact with "BC Manager" and "Miner" to access "IoTDevice2".

(1) In step 1, the "IoTDevice1" sends a "permission activation" request to "IoTDevice2".
(2) In step 2, the "IoTDevice2" forwards it to "BC Manager" for platform and access policies verification.
(3) In step 3, the "BC Manger" generates a "T.activate" transaction and forwards it to the BC network.
(4) In step 4, the "Miner" invokes "Permission Activation Algorithm", as discussed in section IV, Algorithm.1, coded in smart contract of "IoTDevice2". The algorithm verifies platform hash value and access policies stored at the smart contract of "IoTDevice2".
(5) In step 5, after successful verification, the "Miner" sends a success message to "BC Manager".
(6) In step 6, "BC Manager" sends "platform and access policies verified" message to "IoTDevice2".

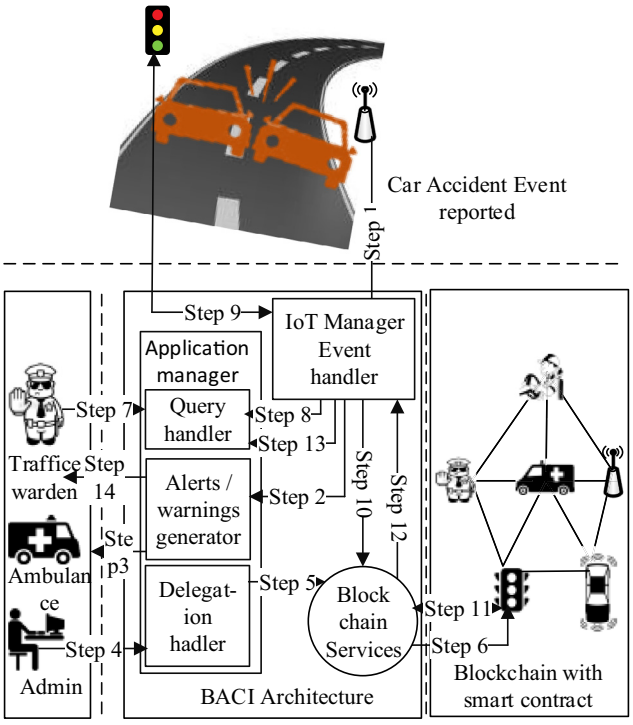

Fig. 7 – Event-based permission delegation.

(7) In step 7, the "IoTDevice2" allows the "IoTDevice1" to activate the delegated permission.

4. Use cases

The permission delegation occurs in response to either query or an event. Therefore, we have divided permissions delegation into two categories, which are event and query base permission delegation. Both are elaborated with the help of the following use cases.

4.1. *Event-based permission delegation*

This use case is aimed for handling car accident on the highway due to overspeeding as shown in Fig. 7. The highway is equipped with ITS sub-systems, e.g., roadside ITS sub-system, vehicle ITS sub-system, personal ITS subsystem and central ITS sub-system. Each sub-system contains ITS station ETSI (2010). The following steps describe permission delegation procedure in the response to a car accident event.

(1) In step 1, roadside ITS station detects the accident and informs IoT manager about the event.
(2) In step 2, IoT manager performs data filtering and event correlation on the data received from roadside ITS station. It notifies application manager about the event.
(3) In step 3, application manager generates alerts/warnings about the accident event to all the subscribed ITS stations like a police vehicle, rescue and fire brigade vehicles.
(4) In step 4, administrator delegates the required permission to the subscribed ITS station for car accident event.

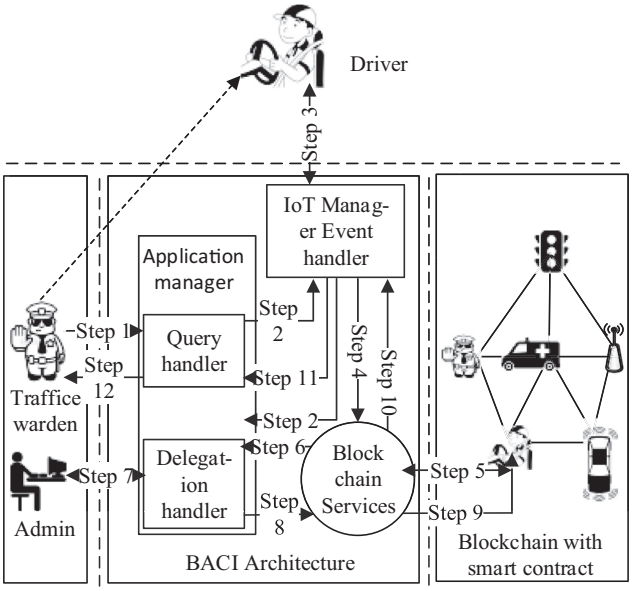

Fig. 8 – Query-based permission delegation.

(5) In step 5, application manager forward the delegated permission with delegator and delegatee information to BC manager.
(6) In step 6, BC manager submits delegation request to BC for validation using "T.delegate$_g$" transaction. BC validates the transaction and appends it to the end of a block.
The following steps describe permission activation procedure. The traffic warden follows this procedure to resolve car accident event.
(7) In steps 7 and 8, the police warden sends an activation query through application manager to IoT manager for a traffic signal light control.
(8) In step 9, the IoT manager unicasts the activation query to the requested traffic signal light ITS station.
(9) In step 10, through IoT manager, the traffic signal light ITS station redirects the activation query to delegation service.
(10) In step 11, the delegation service generation "T.activate" transaction and submit it to the traffic signal light smart contract on the BC. The BC miners verify traffic warden device platform hash value. Then BC checks its database. There must be an entry in the database of traffic signal light smart contract because the administrator has delegated the permission.
(11) In step 12, the delegation service informs the traffic signal light ITS station about the permission delegation.
(12) In steps 13 and 14, the traffic signal light ITS station displays a signal light control panel on the traffic warden smartphone through IoT manager and application manager.

4.2. *Query-based permission delegation*

(1) This case supposes a traffic warden who wants to put a penalty on a driver due to some traffic offenses as shown in Fig. 8. The traffic warden wants to update driver profile by adding this offense.

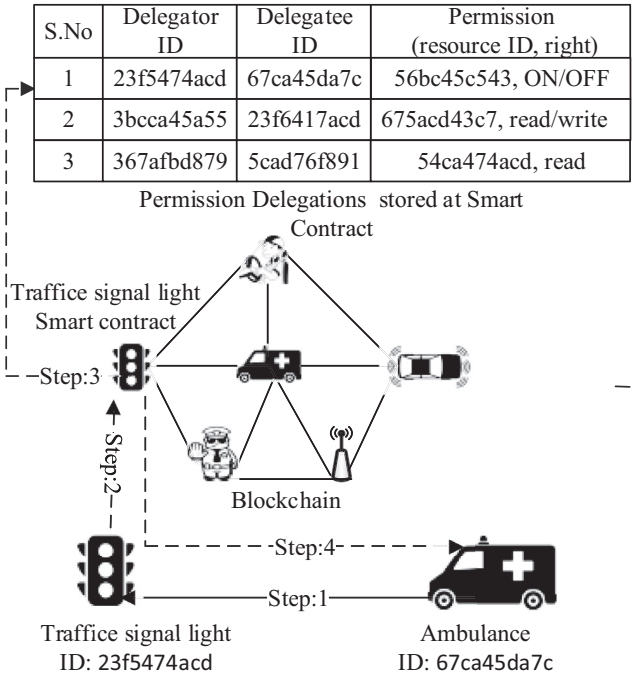

Fig. 9 – Query-based permission delegation.

(a) In steps 1 and 2, traffic warden ITS station unicasts a message to the driver vehicle ITS station through application manager and IoT manager for read and write permission on driver profile.
(b) In steps 3 and 4, the vehicle ITS station redirects the query to the delegation service for verification and validation.
(c) In step 5, the delegation service generates and submits the "T.activate" transaction to the driver smart contract on the BC. The smart contract represents driver profile. The BC miners verify traffic warden vehicle ITS subsystem or personal ITS subsystem. After successful verification, BC checks permission delegation for traffic warden on driver profile.
If permission is delegated before then skip steps 6–9 otherwise proceed step 6.
(d) In steps 6 and 7, the delegation service requests the administrator to delegate permission for the traffic warden.
(e) In steps 8 and 9, administrator delegates read and write permission on driver profile to traffic warden through delegation service.
(f) In step 10, delegation service informs the requesting vehicle ITS station that permission delegation has been made.
(g) In steps 11 and 12, the vehicle ITS station allows traffic warden ITS station to access driver profile through IoT manager and application manager. The traffic warden vehicle or personal ITS subsystem displays driver profile with read and write permission.

(2) This case supposes an ambulance that wants to pass crossroads as shown in Fig. 9. A traffic signal light is installed at the intersection of the crossroads. The traffic signal light status is red at the moment and ON/OFF permissions are delegated to the ambulance ITS station. The following steps describe the permission activation process.
(a) In step 1, ambulance ITS station unicasts a message to the signal light ITS station to change its status to green.
(b) In step 2, the traffic signal light ITS station redirects the query to the BC through delegation service for verification.
(c) In step 3, the BC checks the database of the signal light smart contract. If permission is delegated then there must be an entry for the ambulance in the database.
(d) In step 4, BC provides the control panel of traffic signal light to the ambulance.
(e) If two ambulances have made a request for signal light control ITS station at the same time. Then miners will perform a calculation based on attributes like vehicle type, vehicle priority, direction, speed, distance from the signal light for both the ambulances. After calculation, BC will allow a single user to control traffic signal light.

Table 3 – Notations.

Notation	Description
s_i	ith subject.
o_j	jth object.
p_k	kth permission.
g_i	ith group of subjects.
$\mathcal{C}$	Set of all constraints, e.g., Static Mutually Exclusive Permission (SMEP).
$\cup_{i=0}^{\infty} s_i$	All the subjects in the system.
$\cup_{j=0}^{\infty} o_j$	All the objects in the system.
$\cup_{k=0}^{\infty} p_k$	All the permissions in the system.
$\cup_{i=0}^{\infty} s_{i,att_j}$	All the subjects having a common attribute.
$\cup_{i=0}^{\infty} s_i$	All group in the systems.
$\cup_{i,j=0}^{\infty} \{pol_{si,sj}\}$	All access policies of s_i for s_j.
$\cup_{i,j=0}^{\infty} \{s_i \times p_j\}$	All users and their activated permissions.
att_{s_i}	All attributes of a subject s_i
$\{s_i \rightsquigarrow^{p_k} s_j\}$	s_i delegated a permission p_k to s_j.
$\{s_i \nrightsquigarrow^{p_k} s_j\}$	s_i revoked a permission p_k from s_j.
$\{s_i \ggg p_k\}$	s_i activates a permission p_k.
$\{s_i \lll p_k\}$	s_i deactivates a permission p_k.
$\hbar_c$	recently calculated hash value of a user platform.
$\hbar_s$	a hash value of a user platform, stored at BC.

5. Formal modeling of our proposed architecture

5.1. Notations

5.1.1. Subject set

$\mathcal{S} \supseteq \cup_{i=0}^{\infty} s_i$, $\mathcal{S}$ is a super set of all the things that make a request for a resource in IoT. A subject acts both as delegator and delegatee. A user set is a subset of "Subject set" which refers to all human operators (Table 3).

5.1.2. Object Set

$\mathcal{O} \supseteq \cup_{j=0}^{\infty} o_j$, $\mathcal{O}$ is a super set of all the things that are requested by requester in IoT, e.g., information, services.

5.1.3. *Permission set*

$\mathcal{P} \supseteq \cup_{j=0}^{\infty} p_k$, $\mathcal{P}$ is a super set of all the permissions in IoT. Permission p_k is the kth pair of (o,a), where o is object and a is an action. Permission defines actions on an object.

5.1.4. *Group*

$\mathcal{G} = \cup_{i=0}^{\infty} s_{i,att_j}$, a group $\mathcal{G}$ is a set of all the subjects having specific attributes. For example traffic warden group, rescue group etc. att_j is jth attribute of subject s_i.

5.1.5. *Delegation set ($\mathcal{DS}$)*

$\mathcal{DS} \supseteq \cup_{i,j,k=0}^{\infty}\{s_i \rightsquigarrow^{p_k} s_j\}$. A super set of all the delegations of permission p_i made by delegator s_i to delegatee s_j.

5.1.6. *Attribute set*

$\mathcal{ATT} \supseteq \cup_{i=0}^{\infty} att_i$. Attribute set contains all the attributes of all the subjects in the system. The att_i is the ith attribute in the attribute set $\mathcal{ATT}$. A characteristic of a subject is called attribute. For example, if speed is the characteristic of a vehicle v_i then we write attribute as (v_i.speed).

5.1.7. *Permission activation aet ($\mathcal{PAS}$)*

$\mathcal{PAS} \supseteq \cup_{i,j=0}^{\infty}\{s_i \times p_j\}$, $\mathcal{PAS}$ is a super set contains all the subjects and their activated permissions.

5.1.8. *Policy set (*Pol)

$Pol \supseteq \cup_{i,j=0}^{\infty}\{pol_{si,sj}\}$, Pol is a super set of all the policy in a system.

5.2. *Definitions*

5.2.1. *Delegation*

We define delegation as a triple (s_i, s_j, p_k), where s_i is a delegator, s_j is a delegatee, and p_k is a permission or set of delegated permission. $(s_i, s_j, p_k) \in s_i \rightsquigarrow^{p_k} s_j$

5.2.2. *Delegation_Function*

$(s_i, s_j, p_k, pol_i) \rightarrow (s_i \rightsquigarrow^{p_k} s_j)$. Delegation function maps delegator s_i and permission p_i to a set of delegatees s_j based on access policy pol_i. If a permission is successfully delegated to the delegatee then

$\{s_i \rightsquigarrow^{p_k} s_j\} = \{s_i \rightsquigarrow^{p_k} s_j\} + (s_i, s_j, p_k)$.

5.2.3. *Revocation_Function*

$(s_i, s_j, p_k, pol_i) \rightarrow (s_i \nrightsquigarrow^{p_k} s_j)$. Delegator uses revocation function to revoke permissions from delegatee. $\{s_i \rightsquigarrow^{p_k} s_j\} = \{s_i \rightsquigarrow^{p_k} s_j\}$ - (s_i, s_j, p_k).

5.2.4. *Activation_Function*

$(s_i, p_k) \rightarrow \{s_i \ggg p_k\}$. If a permission is successfully activated by a user, then $\overline{\mathcal{PAS}} = \mathcal{PAS} \cup \{s_i \ggg p_k\}$.

5.2.5. *Deactivation_Function*

$(s_i, p_k) \rightarrow \{s_i \lll p_k\}$. If a permission is successfully deactivated by a user, then $\overline{\mathcal{PAS}} = \mathcal{PAS}$ - $\{s_i \ggg p_k\}$.

5.2.6. *Policy (Pol)*

Policy is a set of permissions that a delegator grants to delegatee on an object. For example, a doctor wants to read a patient record then the policy will be $pol_{admin,doctor} = \{\text{read}\}$.

5.2.7. *Create_Policy_Function*

$(s_i, s_j, p_k) \rightarrow pol_{si,sj} = \{p_k \subseteq P\}$. Delegator uses create policy function to define access rules for delegatee to access the delegated object. $\overline{Pol} = Pol \cup pol_{si,sj}$.

5.2.8. *T.delegate$_{\mathcal{S}}$ transaction*

It is a cartesian product of delegator, delegatee and permission. Formally, $\{\mathcal{S} \times \mathcal{S} \times \mathcal{P}\}$.

5.2.9. *T.delegate$_{\mathcal{G}}$ transaction*

It is a cartesian product of delegator, group of delegatee and permission. Formally, $\{\mathcal{S} \times \mathcal{G} \times \mathcal{P}\}$.

5.2.10. *T.activate transaction*

It is a cartesian product of delegatee, permission and delegatee platform hash value. Formally, $\{\mathcal{S} \times \mathcal{P} \times \hbar_c\}$.

5.2.11. *T.deactivate transaction*

It is a cartesian product of delegatee, permission. Formally, $\{\mathcal{S} \times \mathcal{P}\}$.

5.2.12. *T.revoke transaction*

It is a cartesian product of delegator, delegatee, permission and delegation policy. Formally, $\{\mathcal{S} \times \mathcal{S} \times \mathcal{P} \times pol_{si,sj}\}$.

5.2.13. *T.publish transaction*

It is a cartesian product of delegator, delegatee, permission and a set of constrains. Formally, $\{\mathcal{S} \times \mathcal{S} \times \mathcal{P} \times \mathcal{C}\}$.

5.3. *Permission Activation Algorithm (PAA)*

The "Algorithm 1" takes user s_i, permission p_k and a hash value of the user platform $\hbar_c$ as inputs. It either allows a user to activate a requested permission or return an error. In lines 3–5, the platform hash value received in a request is matched against a previously store hash value at the BC. If hash values do not match, return an error. User s_i cannot active the permission, because delegatee platform is not trustworthy. If hash values match, then in lines 7–8, "Algorithm 1" checks for permission delegation. If permission is not delegated then it calls "*Delegation_Function*". The "*Delegation_Function*" allows the delegator to delegate the requested permission. If permission

Algorithm 1 Permission Activation Algorithm.

1: Input: $(s_j, p_k, \hbar_c)$
2: Output: $\overline{\mathcal{PAS}} = \mathcal{PAS} \cup \{(s_j \ggg p_k)\}$ or error
3: **if** $(\hbar_c \neq \hbar_s)$ **then**
4: return error
5: **end if**
6: **if** $(\hbar_c \equiv \hbar_s)$ **then**
7: **if** $\{(s_i, s_j, p_k) \notin (s_i \rightsquigarrow^{p_k} s_j)\}$ **then**
8: call *Delegation_Function* $(s_i, s_j, p_k, pol_{si,sj}) \rightarrow (s_i \rightsquigarrow^{p_k} s_j)$
9: **else if** $(s_k \lll p_k)$ **then**
10: $\overline{\mathcal{PAS}} = \mathcal{PAS} \cup \{(s_j \ggg p_k)\}$
11: **else**
12: return error
13: **end if**
14: **end if**

is delegated, then in lines 9–12, "Algorithm 1" checks for permission p_k activation by other users. The user s_i is allowed to activate permission p_k, if it is not activated by another user s_k. If a user s_k has activated the permission, then "Algorithm 1" returns an error.

5.4. *Permission Deactivation Algorithm (PDA)*

The "Algorithm 2" takes user activated permission pair ($s_j \ggg p_k$) as inputs. It either allows a user to deactivate the activated permission or returns an error if the permission is not activated. In lines 3–4, if the permission is activated by the user then it calls "*Deactivate_Function*". In lines 5–6, if the permission is not activated by the user then it returns an error.

Algorithm 2 Permission Deactivation Algorithm.

1: Input: $(s_j \ggg p_k)$
2: Output: $\overline{\mathcal{PAS}} = \mathcal{PAS}$ - $\{(s_j \ggg p_k)\}$ or error
3: **if** $\{(s_j \ggg p_k) \in \mathcal{PAS}\}$ **then**
4: call *Deactivation_Function* $(s_j, p_k) \rightarrow (s_j \lll p_k)$
5: **else**
6: error
7: **end if**

5.5. *Query-base Permission Delegation Algorithm (QBPDA)*

The "Algorithm 3" takes delegator s_i, delegatee s_j and permission p_k as input and as a result delegates the required permission. It checks for the delegation policy in the Policy set *Pol*. If the policy is created by the delegator, then in line 4, it calls "*Delegation_Function*". If the policy is not created by the delegator, then in line 6, it calls *Create_Policy_Function*. After policy creation, it calls "*Delegation_Function*" for permission delegation.

Algorithm 3 Query-based Permission Delegation Algorithm.

1: Input: (s_i, s_j, p_k)
2: Output: $(s_i \rightsquigarrow^{p_k} s_j)$
3: **if** $(pol_{si,sj}) \in$ Pol **then**
4: call *Delegation_Function* $(s_i, s_j, p_k, pol_{si,sj}) \rightarrow (s_i \rightsquigarrow^{p_k} s_j)$
5: **else if** $(pol_{si,sj}) \notin$ Pol **then**
6: call *Create_Policy_Function* $(s_i, s_j, p_k) \rightarrow pol_{si,sj} = \{p_k \subseteq \mathcal{P}\}$
7: goto line 3.
8: **end if**

5.6. *Event-base Permission Delegation Algorithm (EBPDA)*

The "Algorithm 4" takes delegator s_i, delegatee group g_j and permission p_k as inputs and as a result delegates the required permission to the group or a single user in the group. It checks for the delegation policy in the Policy set *Pol*. If the policy is created by the delegator, then in line 4, it compares attributes of all the users in the group g_i. It selects one or more vehicles based on their attributes. In line 5, it calls "*Delegation_Function*". If the policy is not created by the delegator, then in line 6, it call *Create_Policy_Function*. After successful policy creation, it performs attributes comparison. It selects one or more vehicles based on their attributes and then call "*Delegation_Function*" for permission delegation.

Algorithm 4 Event-base Permission Delegation Algorithm.

1: Input: (s_i, g_j, p_k)
2: Output: $(s_i \rightsquigarrow^{p_k} s_j)$
3: **if** $(pol_{si,sj}) \in$ Pol **then**
4: **if** {compare(att_{s_i}, att_{s_k}) **then**
5: call *Delegation_Function* $(s_i, s_j, p_k, pol_{si,sj}) \rightarrow (s_i \rightsquigarrow^{p_k} s_j)$
6: **else if** $(pol_{si,sj}) \notin$ Pol **then**
7: call *Create_Policy_Function* $(s_i, s_j, p_k) \rightarrow pol_{si,sj} = \{p_k \subseteq \mathcal{P}\}$
8: goto line 3.
9: **end if**
10: **end if**

5.7. *Policy Creation Algorithm (PCA)*

The "Algorithm 5" takes user s_i, user s_j, permission p_k and constraints as input. It updates the policy set (*Pol*) with a new policy or returns an error. It evaluates SMEP constraints before new policy creation. If SMEP does not exist then it calls "*Create_Policy_Function*". It adds the newly created policy to the Policy set *Pol*. If SMEP exist then it returns an error.

Algorithm 5 Policy Creation Algorithm.

1: Input: $(s_i, s_j, P_k, \mathcal{C})$
2: Output: (pol_i) or error
3: **for** $(p_j$ in P_j of $s_j)$ **do**
4: **for** $(p_k$ in P_k of $s_i)$ **do**
5: **if** $(p_j, p_k) \notin$ SMEP **then**
6: call *Create_Policy_Function* $(s_i, s_j, p_k) \rightarrow pol_{si,sj} = \{p_k \subseteq \mathcal{P}\}$
7: **else**
8: return error
9: **end if**
10: **end for**
11: **end for**

5.8. *Permission Revocation Algorithm (PRA)*

The "Algorithm 6" takes delegator s_i, delegatee s_j, permission p_k and $pol_{si,sj}$ as inputs and as a result revokes the delegated permission. It checks for the delegation policy in the Policy set *Pol*. If the policy is revoked by the delegator, then in lines 4–5, it deactivates the permission. It removes the deactivated permission from the delegation set. If the policy is not revoked then it returns an error.

5.9. *Security and performance analysis*

We analyze the security of our architecture using Confidentiality, Integrity, and Availability (CIA) model (Komninos et al., 2014). The evaluation of security parameters are given in

Algorithm 6 Permission Revocation Algorithm.

1: Input: $(s_i, s_j, p_k, pol_{si,sj})$
2: Output: $(s_i \rightsquigarrow^{p_k} s_j)$ or error
3: **if** $(pol_{si,sj}) \notin$ Pol **then**
4: $\overline{\mathcal{PAS}} = \mathcal{PAS}$ - $\{s_i \ggg p_k\}$
5: call *Revocation_Function* : $(s_i, s_j, p_k, pol_i) \rightarrow (s_i \rightsquigarrow^{p_k} s_j)$
6: **else**
7: error
8: **end if**

Table 4 – Security parameters evaluation.

Parameters	Description
Confidentiality	DTLS is used to secure communication betw-een IoT device and IoT manager. Public key encryption is used to secure communication between IoT manager and BC.
Availability	Attained by BC distributed ledger replication and by using platform verification before per-mission activation and delegation.
Integrity	Both platform and data in transit integrity att-ained by using hashing.
Access control	Attained by using access policy stored at sma-rt contract.
Platform verification	Attained by using platform hashes stored at smart contract.

Table 4. Confidentiality means that information is not available to the unauthorized user. Integrity means that information is not modified in an unauthorized manner. Availability makes sure that the service or information is available when it is needed. We have achieved the above three security requirements by integrating BC technology into our architecture. In BC, after validation, each transaction is included in a block with other transactions. Then, the block is added to the BC. Each block contains the hash value of the previous block. The hash value in the current block will change if a transaction in the previous block is tampered. So, it is very difficult to tamper a previous transaction because it required re-validation of all subsequent blocks.

The BC distributed ledger ensures that every BC node has a copy of the ledger. A BC node can lost his ledger due to some attacks. It can restore the accurate copy from other BC nodes through replication. We assume Constrained Application Protocol (CoAP) based communicate IoT devices and IoT manager. The CoAP uses Datagram Transport Layer Security (DTLS) for the integrity and confidentiality of the communication. The communication between the BC manager and BC miner occurs through BC transactions. To ensure confidentiality, each transaction is encrypted with the key using public key encryption. To provide data integrity, each transaction carries a hash value of the data.

The Distributed Denial of Service (DDOS) attack damages the availability of a target node. The attacker, in DDOS attack, overwhelms a target node by using multiple compromised IoT devices. Suppose a scenario in which an attacker manages to compromise an IoT device by installing a malware. Now the attacker seeks to lunch an attack on other IoT devices. It exploits the delegation rule made for the compromised IoT device. To access other IoT devices, the compromised IoT device/attacker sends an activation transaction to the BC. In our architecture, the BC miners perform platform verification before every permission activation. The compromised IoT device platform current hash will not match with his original hash value stored at BC. Thus, the miner will not allow the compromised IoT device to activate a permission.

There is a trade-off between BC performance and security. Public BC like bitcoin uses PoW as a consensus mechanism. Bitcoin takes 10 minutes to validate 1 block due to difficulty level defined by PoW. The difficulty level in PoW enhances BC security. The proposed architecture consists of a private BC network. Unlike public BC, private BC achieves the consensus through selective endorsement. Private BC does not require PoW. Therefore, it requires less time and computational power to confirm a transaction. Hence, the proposed architecture has high throughput. Furthermore, we have defined seven different types of transactions. These transactions are given in Table 2. The transactions "T.activate" and "T.deactivate" use most frequently. These transactions read access policies stored in BC. Therefore, it does not execute the BC consensus mechanism. Hence, it enhances the proposed architecture throughput. The "T.register" transaction is used only once by IoT device at the time of registration within the BC network. We have defined groups of subscribed users like traffic wardens group or rescue personnel group. Each group is assigned to a particular set of permissions. The T.delegate$_{\mathcal{S}}$ and T.delegate$_{\mathcal{G}}$ Transactions execute BC consensus mechanism. However, they are used only when a user needs certain permissions assigned to a particular group i.e., a group of all signal lights on a particular road or group of drivers profile.

5.10. *Comparison with related works*

The Table 5 shows a comparison among existing related architectures and proposed architecture.

6. Verification using SPIN model checker

6.1. *Implementation details*

According to Holzmann (2003), there are three building blocks in a PROMELA implementation, i.e., Process, Data object and message channels. The following subsections describe these components with respect to our proposed model.

6.1.1. *Blockchain process*

In our model, there is one "Blockchain" process. The "Blockchain" process consists of "idle", "delegation", "activation" and "on" states. The buchi automaton Büchi (1960) of "Blockchain" process is shown in Fig. 10(a). At "idle" state, the "Blockchain" process receives permission activation request and change the state to "delegation". The "Blockchain" process can access the data objects that contain delegation rules. The "Blockchain" process moves into "activation" state after successful validation of delegation rules. If there is no delegation, then "Blockchain" process moves into "idle" state. At "activation state", the IoT device platform current hash is compared with his original hash value store at BC.

Table 5 – Comparison with related works.

Mechanism /Property	Tapas et al. (2018), Novo (2018) based on Ethereum BC	Ouaddah et al. (2017) based on bitcoin BC	Alam et al. (2011) based on Single Trusted Entity	BACI based on Hyperledger Fabric
Decentra-lized	Yes	Yes	No	Yes
Fault tolerance	Higher	Higher	Lower	Higher
Permission Delegation	Tapas et al. (2018) query based permission delegation Novo (2018) No delegation	No delegation	Query based permission delegation	Both query and event based permission delegation
Platform Verification	No	No	No	Yes
Immutabi-lity	impossible to tamper	Impossible to tamper	Could be tampered	Impossible to tamper
Consensus mechanism	PoW	Proof of Concept	No consensus	Practical Byzantine fault tolerance (PBFT)
Efficiency	performance closer to main ethereum network	Performance depended on bitcoin BC	Depended on central entity	Higher than both ethereum and bitcoin BC

If hashes match, then "Blockchain" process moves into "on" state and IoT device process is allowed to access the resource. If hashes do not match, then "Blockchain" process move into "idle" state. At "on" state, when the delegated permission is deactivated, then "Blockchain" process moves into "idle" state.

6.1.2. *IoT device process*

In our model, there are "IoTDevice1" and "IoTDevice2" processes. Both processes have identical state transition diagrams. Each process consist of "requesting", "waiting", "accessing", "done" and "denial" states. The buchi automaton of an IoT device process is shown in Fig. 10(b). In "requesting" state, IoT device process generates a permission activation request and moves into "waiting" state. At "waiting" state, the IoT device process waits for "Blockchain" process to validate delegation rules and platform hashes. On successful validation of both delegation rules and platform hashes, IoT device process moves into "accessing" state. The IoT device process moves into "denial" state if delegation rules do not validate or delegation rules are validated but platform hashes do not match. On permission deactivation, IoT device process moves into "done" state. Then the IoT device moves into "requesting" state to generate another request.

6.1.3. *Adversary process*

In our model, there is one "Adversary" process. The adversary is a compromised IoT device. So the "Adversary", "IoTDevice1" and "IoTDevice2" processes have identical states. Therefore, buchi automaton of "Adversary" process is similar to "IoTDevice1" and "IoTDevice2" processes. It is shown in Fig. 10(b).

6.1.4. *Message channels*

We have defined five channel variables. These channel variables are used to exchange messages between processes. We have defined eleven types of messages using mtype declaration.

6.1.5. *Data objects*

In our model, we have defined two data structures. One data structure, a two-dimensional array, stores delegation rules. The first row contains delegators whereas the first column contains delegatees as shown in Table 6. The value "1" at the intersection represents that the delegator has delegated their permissions. The value "0" at the intersection represents that the delegator has not delegated their permissions. The second data structure is a linear array which stores platform hashes of IoT devices. Both the data structures are accessible to "blockchain" process only.

Table 6 – Delegation rules.

Delegator/delegator	IoTDevice1	IoTDevice2	IoTDevice3
IoTDevice1	0	0	1
IoTDevice2	0	0	1
IoTDevice3	0	0	0

6.2. *Simulation results*

The SPIN model checker generated simulation result of our model is shown in Fig. 11. We have run "Adversary", "IoTDevice1", "IoTDevice2" and "Blockchain" processes concurrently. In the result, the vertical lines represent the execution path of the four processes. The boxes on the vertical line show the execution steps of the processes. The cross arrows between boxes show message exchange between processes.

6.3. *Never claims / LTL formulas verification*

The "never claim" is used to describe system behavior that should never occur. Therefore, "never claim" negates positive properties before they are interpreted into a claim. It consists of prepositions or boolean expressions. The SPIN generates "never claim" automatically from LTL formula. The LTL formula is used to specify the property that must be proved by the model. We have verified the following LTL formulas against our model.

6.3.1. *Platform verification property (P1)*

$\{[](!p) \rightarrow [](!q)\}$

```
#define p (r == hash[1])
#define q (Adversary@accessing_state)
```

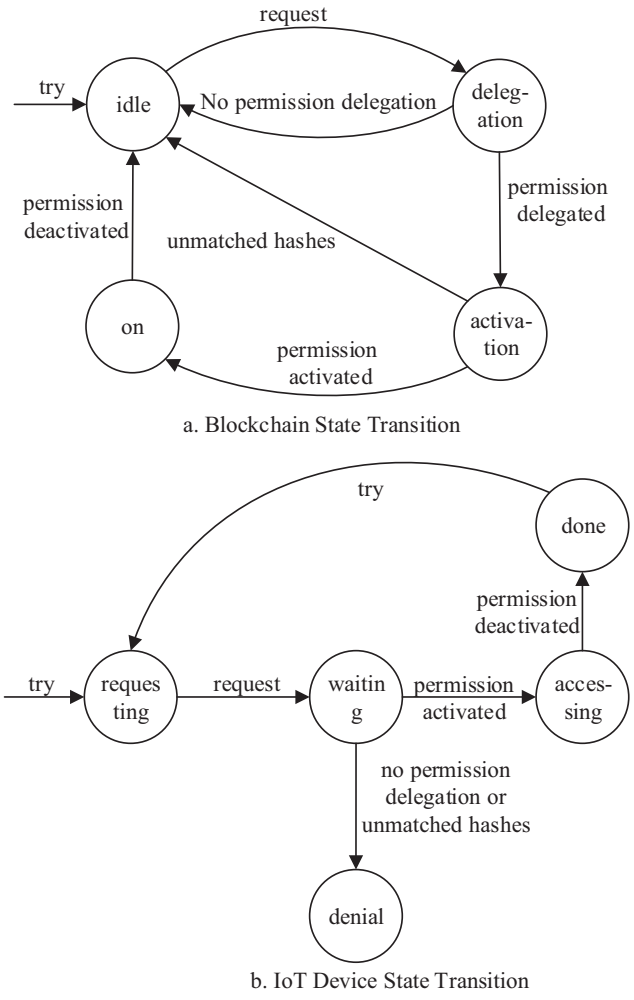

Fig. 10 – IoT device and Blockchain state transition diagrams.

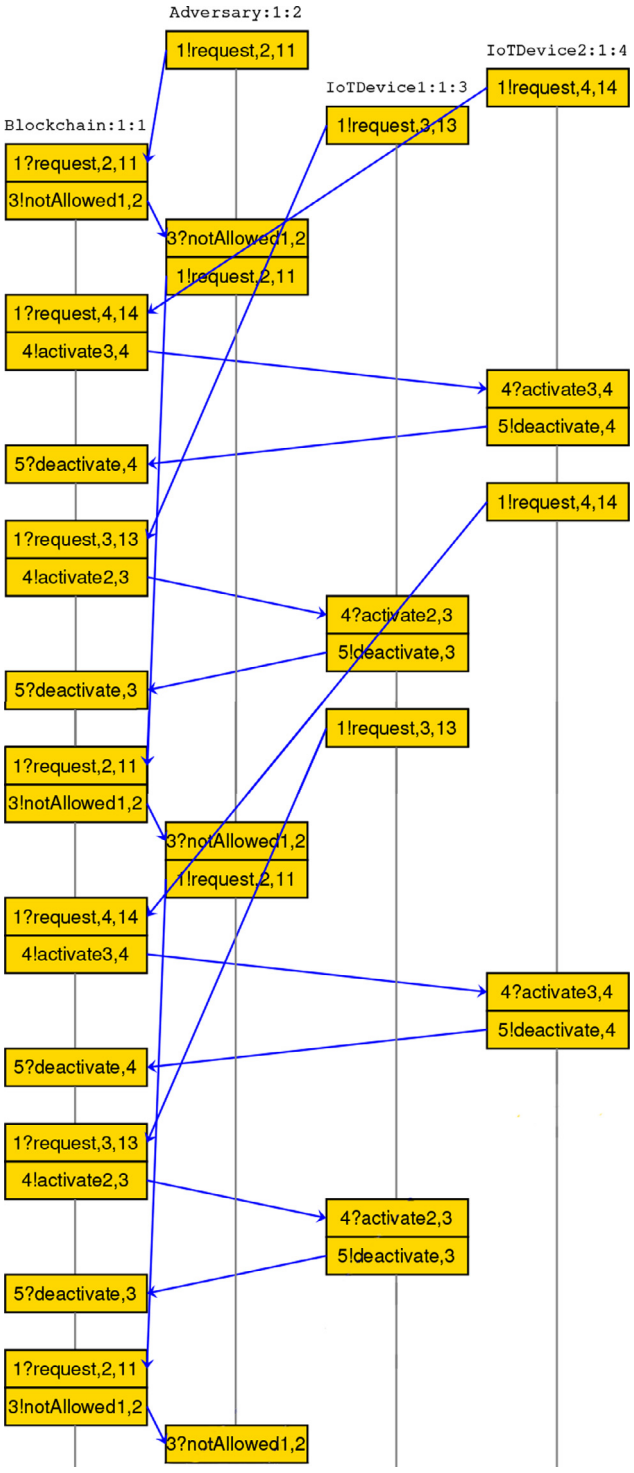

Fig. 11 – Simulation result.

Suppose adversary has installed malware on an IoT device. The adversary wants to access other devices by exploiting the compromised IoT device delegation rules. The adversary sends a permission activation request to BC with his platform hash value. The BC compares received hash with stored hash value. The hashes will not match, therefore, the adversary is not allowed to access the resource. The above property states that when hashes do not match then "adversary" process is not allowed to move into "access_state". The SPIN model checker generated "never claim" automata is shown in Fig. 12. The SPIN model checker verification result of the above property is shown in Table 7.

6.3.2. *Delegation property (P2)*

{ [](!p) → [](!q) }

#define p (a[5].aa[3] = = 1)

#define q (IoTDevice1@activation_State)

P defines the delegation rule stored at BC data structure whereas q defines activation_State of "IoTDevice1". The above property states that a delegatee cannot activate a permission if the the permission is not delegated by the delegator. The SPIN model checker verification result and "never claim" of the above property is shown in Table 8 and Fig. 13, respectively.

6.3.3. *Mutual Exclusion property (P3)*

[]!(p && q)

#define p (IoTDevice1status = = 1)

#define q (IoTDevice2status = = 1)

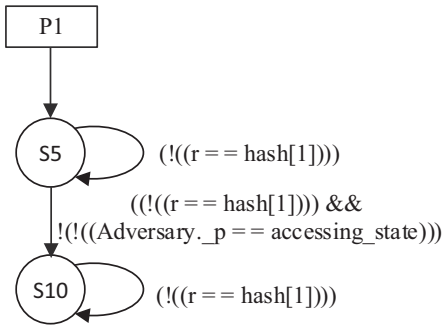

Fig. 12 – Automata view of "Platform Verification" property.

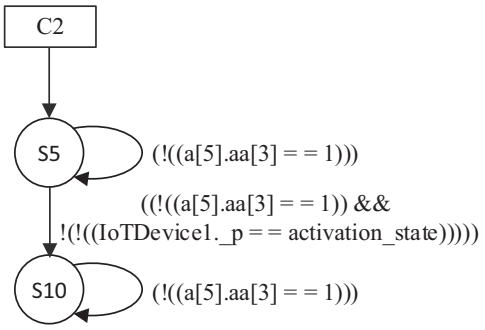

Fig. 13 – Automata view of "Delegation" property.

Table 7 – SPIN output: verification of "Platform Verification" property.

```
pan: ltl formula P1
(Spin Version 6.4.6 – 2 December 2016)
+ Partial Order Reduction
Full statespace search for:
        never claim             + (P1)
        assertion violations    + (if within scope of claim)
        acceptance cycles       + (fairness disabled)
        invalid end states      – (disabled by never claim)
State-vector 156 byte, depth reached 2863, errors: 0
24108 states, stored
36835 states, matched
60943 transitions (= stored+matched)
0 atomic steps
hash conflicts: 12 (resolved)
```

Table 8 – SPIN output: verification of "Delegation" property.

```
pan: ltl formula P2
(Spin Version 6.4.6 – 2 December 2016)
+ Partial Order Reduction
Full statespace search for:
        never claim             + (P2)
        assertion violations    + (if within scope of claim)
        acceptance cycles       + (fairness disabled)
        invalid end states      – (disabled by never claim)
      State-vector 156 byte, depth reached 82, errors: 0
564 states, stored
495 states, matched
1059 transitions (= stored+matched)
0 atomic steps
hash conflicts: 0 (resolved)
```

In access control, mutual exclusion means that a permission p cannot be assigned to users u1 and u2 at the same time, i.e., {(u1, u2), p)}. The above property states that both "IoTDevice1" and "IoTDevice2" cannot activate a permission at the same time, e.g., two ambulances cannot control a traffic signal light at the same. The SPIN model checker verification result

Table 9 – SPIN Output: Verification of "Mutual Exclusion" Property

```
pan: ltl formula P3
(Spin Version 6.4.6 – 2 December 2016)
+ Partial Order Reduction
Full statespace search for:
        never claim             + (P3)
        assertion violations    + (if within scope of claim)
        acceptance cycles       + (fairness disabled)
        invalid end states      - (disabled by never claim)
State-vector 156 byte, depth reached 2863, errors: 0
24108 states, stored
36835 states, matched
60943 transitions (= stored+matched)
0 atomic steps
hash conflicts: 14 (resolved)
```

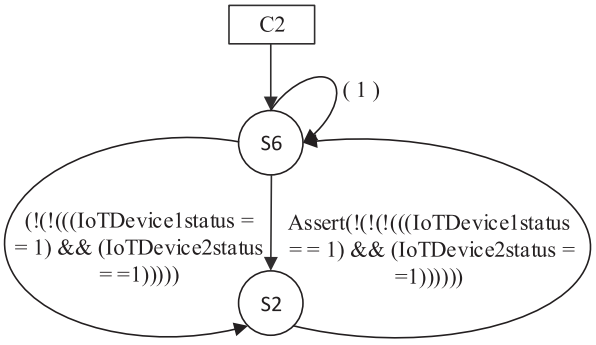

Fig. 14 – Automata view of "Mutual Exclusion" property.

and "never claim" of the above property is shown in Table 9 and Fig. 14, respectively.

7. Conclusion

In this paper, we have presented an architecture for permission delegation and access control in IoT. Initially, an IoT device or user is assigned to a limited permission group, additional permissions are granted through permission

delegation. BC is used to performed permission delegation service. The IoT device owner creates a smart contract on the BC for his resource. To access a resource, the IoT device or user sends his request to the resource smart contract on the BC. Before permission activation, BC verifies delegatee platform trustworthiness and delegation rules. Our architecture is a hybrid architecture that performs both event-based as well as query-based permission delegation. The security of our architecture is analyzed using the Confidentiality, Integrity and Availability model. We have achieved the above three security requirements by integrating BC technology into our architecture. BC technology is proven as secure, verifiable, trusted and tamper resistant mechanism for decentralized systems.

Furthermore, we have modeled our approach in PROMELA. The "Platform Verification", "Delegation", "Mutual Exclusion" properties are validated against the PROMELA model using SPIN model checker. The results show that our BACI architecture is safe and secure. Currently, we are working on permission delegation and access control in cross-domain. In the future, we planned to work on formal modeling and formal verification of BC.

Declarations of interest

None.

REFERENCES

Abomhara M, et al. Cyber security and the internet of things: vulnerabilities, threats, intruders and attacks. J Cyber Secur Mobil 2015;4(1):65–88.

Al-Fuqaha A, Guizani M, Mohammadi M, Aledhari M, Ayyash M. Internet of things: a survey on enabling technologies, protocols, and applications. IEEE Commun Surv Tutor 2015;17(4):2347–76.

Alam M, Zhang X, Khan K, Ali G. xdauth: a scalable and lightweight framework for cross domain access control and delegation. In: Proceedings of the 16th ACM symposium on access control models and technologies. ACM; 2011. p. 31–40.

Büchi JR. Weak second-order arithmetic and finite automata. Math Logic Q 1960;6(1–6):66–92.

Buterin V, et al., 2014. A next-generation smart contract and decentralized application platform. White Paper.

Christidis K, Devetsikiotis M. Blockchains and smart contracts for the internet of things. IEEE Access 2016;4:2292–303.

Clack CD, Bakshi VA, Braine L, 2016a. Smart contract templates: essential requirements and design options, arXiv:1612.04496.

Clack CD, Bakshi VA, Braine L, 2016b. Smart contract templates: foundations, design landscape and research directions, arXiv:1608.00771.

Dorri A, Kanhere SS, Jurdak R, Gauravaram P. Blockchain for IoT security and privacy: The case study of a smart home. In: Proceedings of the 2017 IEEE international conference on pervasive computing and communications workshops (PerCom workshops). IEEE; 2017. p. 618–23.

Dorri A, Kanhere SS, Jurdak R., 2016. Blockchain in internet of things: challenges and solutions, arXiv:1608.05187.

Ethereum foundation 2014. the solidity contract-oriented language. https://github.com/ethereum/solidity.

ETSI EE. 302 665 v1. 1.1: Intelligent transport systems (its), communications architecture. European Standard (Telecommunications Series)(September 2010) 2010.

Gattolin A, Rottondi C, Verticalc G. Blast: Blockchain-assisted key transparency for device authentication. In: Proceedings of the 2018 IEEE 4th international forum on research and technology for society and industry (RTSI). IEEE; 2018. p. 1–6.

Gerth R, Peled D, Vardi MY, Wolper P. Simple on-the-fly automatic verification of linear temporal logic. In: Protocol specification, testing and verification XV. Springer; 1995. p. 3–18.

Granjal J, Monteiro E, Silva JS. Security for the internet of things: a survey of existing protocols and open research issues. IEEE Commun Surv Tutor 2015;17(3):1294–312.

Gusmeroli S, Piccione S, Rotondi D. Iot access control issues: a capability based approach. In: Proceedings of the 2012 sixth international conference on innovative mobile and internet services in ubiquitous computing (IMIS). IEEE; 2012. p. 787–92.

Hammi MT, Hammi B, Bellot P, Serhrouchni A. Bubbles of trust: a decentralized blockchain-based authentication system for IoT. Comput Secur 2018;78:126–42.

Heilman E, Kendler A, Zohar A, Goldberg S. Eclipse attacks on bitcoin's peer-to-peer network. In: USENIX Security; 2015. p. 129–44.

Holzmann GJ. The model checker spin. IEEE Trans Softw Eng 1997;23(5):279–95.

Holzmann G. Spin model checker, the: primer and reference manual. Addison-Wesley Professional; 2003.

Komninos N, Philippou E, Pitsillides A. Survey in smart grid and smart home security: Issues, challenges and countermeasures. IEEE Commun Surv Tutor 2014;16(4):1933–54.

Maesa DDF, Mori P, Ricci L. Blockchain based access control. In: Proceedings of the IFIP international conference on distributed applications and interoperable systems. Springer; 2017. p. 206–20.

Nakamoto S, 2008. Bitcoin: a peer-to-peer electronic cash system, Available: http://bitcoin.org/bitcoin.pdf.

Novo O. Blockchain meets IoT: An architecture for scalable access management in IoT. IEEE Int Things J 2018;5:1184–95.

Ouaddah A, Abou Elkalam A, Ait Ouahman A. Fairaccess: a new blockchain-based access control framework for the internet of things. Secur Commun Netw 2016;9(18):5943–64.

Ouaddah A, Elkalam AA, Ouahman AA. Towards a novel privacy-preserving access control model based on blockchain technology in IoT. In: Europe and MENA cooperation advances in information and communication technologies. Springer; 2017. p. 523–33.

Puthal D, Mohanty SP, Nanda P, Kougianos E, Das G. Proof-of-authentication for scalable blockchain in resource-constrained distributed systems. In: Proceedings of the 2019 IEEE international conference on consumer electronics (ICCE). IEEE; 2019. p. 1–5.

Rivera J, van der Meulen R. Gartner says 4.9 billion connected things will be in use in 2015. Gartner; 2014.

Sandhu RS, Samarati P. Access control: principle and practice. IEEE Commun Mag 1994;32(9):40–8.

Tapas N, Merlino G, Longo F. Blockchain-based iot-cloud authorization and delegation. In: Proceedings of the 2018 IEEE international conference on smart computing (SMARTCOMP). IEEE; 2018. p. 411–16.

Tapscott D, Tapscott A. Blockchain revolution: how the technology behind Bitcoin is changing money, business, and the world. Penguin; 2016.

Wang Q, Li N, Chen H. On the security of delegation in access control systems. In: Proceedings of the European symposium on research in computer security. Springer; 2008. p. 317–32.

Wu L, Du X, Wang W, Lin B. An out-of-band authentication scheme for internet of things using blockchain technology. In: Proceedings of the 2018 international conference on

computing, networking and communications (ICNC). IEEE; 2018. p. 769–73.
Xu R, Chen Y, Blasch E, Chen G. Blendcac: a smart contract enabled decentralized capability-based access control mechanism for the IoT. Computers 2018;7(3):39.

Gauhar Ali received his MS (Computer Science) degree from Institute of Management Sciences, Peshawar, in 2012. He is a PhD scholar at University of Peshawar. His research interests include Internet of Things, Access Control, Blockchain, Intelligent Transport System, Formal Verification, and Model Checking.

Naveed Ahmad received his BS (Computer Science) degree from University of Peshawar, Pakistan in 2007 and PhD in Computer Science from University of Surrey, UK in 2013. He is currently working as an Assistant Professor in Department of Computer Science, University of Peshawar, Pakistan. His research interests include security and privacy in emerging networks such as VANETs, DTN, and Internet of Things (IoT).

Yue Cao received the PhD degree from the Institute for Communication Systems (ICS), 5G Innovation Centre (5GIC), at University of Surrey, Guildford, UK in 2013. He was a Research Fellow at the ICS until September 2016, and Lecturer in Department of Computer and Information Sciences, at Northumbria University, Newcastle upon Tyne, UK until July 2017, and currently the Senior Lecturer since August 2017. His research interests focus on Intelligent Mobility. He is the Associate Editor of IEEE Access, KSII Transactions on Internet and Information Systems.

Muhammad Asif is working as an assistant professor in Department of Electronics, University of Peshawar. He did his PhD in Electronics Engineering from University of Surrey, UK. His research interests include MANETs, sensor networks, signal processing, and renewable energy.

Haitham Cruickshank worked in ICS (formerly CCSR) since January 1996 on several European research projects in the ACTS, ESPRIT, Ten-Telecom and IST programmes. His main research interests are network security, satellite network architectures, VoIP and IP conferencing over satellites. He has worked in several FP6 projects such as SATLIFE, EuroNGI, and SATNEX, also taught in the Data and Internet Networking and satellite communication courses at University of Surrey.

He is a chartered engineer and corporate member of the IEEE in UK, also a member of the Satellite and Space Communications Committee of the IEEE ComSoc, and is active in the ETSI BSM (Broadband Satellite Multimedia) and the IETF MSEC groups.

Qazi Ejaz Ali did his MS (Computer Science) degree in 2008 from IBMS, Agricultural University Peshawar, Pakistan. He is working towards his Ph.D. Degree in Computer Science from Department of Computer Science, University of Peshawar and in addition, he is working as an Assistant Professor in Department of Computer Science, University of Peshawar, Pakistan. His research interests are network security, intelligent transport system security and privacy, and its efficiency.