BCON: Blockchain Based Access CONtrol across Multiple Conflict of Interest Domains

Gauhar Ali, Naveed Ahmad, Yue Cao, *Member, IEEE*, Qazi Ejaz Ali, Fazal Azim, and Haitham Cruickshank, *Member, IEEE*,

***Abstract*—In today's on-demand computing and virtual coalition environment, cross-domain services are acquired and provided. These business domains may belong to either the same or different conflict of interest system. "Transitive access" can cause leakage of information between competitors through some other conflict of interest system's member. Therefore, a secure access control mechanism is required to detect and deny "transitive access" efficiently with minimal trust in externalist. Existing access control mechanisms focused on either single or multiple conflict of interest domains but with no "transitive access". In addition, these existing mechanisms are centralized with inherited unfair access control and are a single point of failure. Blockchain (BC) is a shared digital ledger encompassing a list of connected blocks stored on a decentralized distributed network that is secured through cryptography. We propose a BC based access control for conflict of interest domains. We have integrated a BC in our architecture to make access control fair, verifiable and decentralized. Users access histories and "transitive accesses" are stored on BC ledger. We propose a novel mechanism called "Transitive Access Checking and Enforcement (TACE)" i.e., "Algorithm.1". It makes an authorization decision based on BC endorsement that "transitive access" will not occur. "Algorithm.2" verifies and updates users access histories stored at BC before each request approval. Similarly, "Algorithm.3" detects possible future "transitive accesses" and updates Transitive Access Set (TAS) stored at BC after each request approval. The Simple Promela Interpreter (SPIN) model checker is used to verify the proposed mechanisms for "transitive access" detection and prevention. We have identified four conflicting sequences of execution that can cause "transitive access". Results show that the proposed mechanism is safe against "transitive access" by checking all the four possible conflicting sequences of execution.**

***Index Terms*—Access Control, Formal Verification, Blockchain, Resource Management and Allocation, Model Checking.**

I. Introduction

In todays ubiquitous and dynamic environment, most of the devices have the ability to share information and resources. These devices can be connected in a variety of ways such as wireless sensor network, cloud computing [1], Internet of Things (IoT) [2] to enable ubiquitous and on-demand access to servers, storage, computer networks, computer applications and services. These dynamic coalitions enable the organizations to open their data for cross-domain access. For example, commercial organizations take consultancy services such as audit from financial institutions. Therefore, consultants from a financial institution need to access the data of the organization.

These dynamic coalitions raise new access control challenges. The authentication and access control mechanisms need to be redefined with the least trust for cross-domain users. The consultant or even financial institution can misuse the organization data. It can provide the organization data to other competitors. The situation becomes worse when the consultant has access to multiple organizations data. In such a virtual coalition, users i.e., consultants can request and access resources from different conflict of interest domains/classes. These cross-domain accesses can cause "transitive access".

The "transitive access" allows a user to get an unpermitted resource through some legitimate subject and/or object. Suppose α and β are two conflict of interest classes. Class α has two competitors, company α_1 and company α_2. Similarly, company β_1 belongs to conflict of interest class B. The consultancy firm assigns two consultants Alice and Bob to β_1 for consultancy services and both have read (r) and write (w) access to β_1 dataset. Later on, Alice and Bob can also generate a request for company α_1 and company α_2 datasets with r and w permissions respectively. If access is granted, then it is highly likely that information of company α_1 may be leaked to company α_2 through β_1. A secure mechanism is required to stop "transitive access" that can cause leakage of information between conflict of interest classes.

In [3], authors have proposed a formal framework for cross-domain access control. They have considered a use case where the trust relationship between collaborating organizations is minimal. They have proposed a trusted third party called delegation service for cross-domain access control. The trusted third party is used to protects the privacy of requesting user and privacy of the resource owner. Both the service requester and service provider are unknown to each other. In [4], authors have presented a framework xDAuth, for cross-domain access and delegation control. A trusted Delegation Service (DS) makes an authorization for cross-domain access requests. The DS redirects the user for authentication to his own domain. After successful authentication, DS makes an authorization decision based on service provider access policies. Similarly, a centralized external entity called OAuth based authorization service (OAS) is proposed in [5]. The service provider redirects all users access request to OAS for authorization.

The coalition partners in the real world outsource cross-domain authentication and authorization to trusted third party called identity provider or delegation service [4], [3] and

Gauhar Ali and Naveed Ahmad are with the Department of Computer Science, University of peshawar. Email: gauharstd; n.ahmad@uop.edu.pk.

Yue Cao is with the School of Computing and Communications, Lancaster University, UK. Email: yue.cao@lancaster.ac.uk.

Qazi Ejaz and Fazal Azim are with the Department of Computer Science, University of peshawar. Email: qaziejazali@uop.edu.pk; azimfazal@gmail.com.

Haitham Cruickshank is with the Institute of Communication Systems, University of Surrey, GU2 7XH, UK. Email: h.cruickshank@surrey.ac.uk.

[5]. These security companies authorize cross-domain's users based on some pre-defined access control policy of the organization. Moreover, the security companies store user's credentials and organization's access control policies. Furthermore, a single trusted entity controls cross-domain accesses. Here, if the single trusted entity fails, then cross-domain access will not occur. These centralized security companies may be biased while permitting illegal or transitive requests and denying legal requests. Also, the centralized entity knows the credentials of both service requester or service provider. It may disclose credential to other users. Furthermore, the exiting architectures have targeted multiple conflict of interest domains but with no "transitive access". Therefore, a mechanism is required to develop trust amongst coalition partners, meanwhile without a single trusted entity. Also, a mechanism is required to allow the clients i.e., resource owner to verify the allocation of consultants to their resources.

Our Proposed work suggests BC based access control across multiple conflict of interest domains. Our proposed TACE mechanism (**Algorithm.1**) makes an authorization decision based on BC validation. Similarly, we propose two transaction validation algorithms i.e., **Algorithm.2** and **Algorithm.3**. **Algorithm.1** executes for every request and checks TAS before the request is granted. Then, it calls **Algorithm.2** to validate the transaction and adds the new request to the BC block. Each user permitted accesses are maintained in a directed graph called Access Flow Graph (AFG) which is stored on BC in Process Access Set (PAS). New "transitive access" may occur when a user request is allowed. Therefore, **Algorithm.1** checks PAS for possible new "transitive access" after each request is granted. Then, it calls **Algorithm.3** to validate the new "transitive access" and adds the pair of transitive requests to the BC block.

Furthermore, the proposed architecture makes unbiased user authorization decision through BC consensus. Since, the miners validate user request against access control policies stored at BC. Also, the user can audit BC authorization decision. Every BC node has a copy of the distributed ledger and synchronized through replication. Therefore, it eliminates the case if some of the nodes malfunction in the chain.

Moreover, we use SPIN (Simple Promela INterpreter) [6] model checker for formal verification of our approach. The SPIN model checker uses a high-level language called Process Meta Language (PROMELA). We use PROMELA to develop a model for our proposed mechanism. The behavior of the model is described in Linear Temporal Logic (LTL) [7]. The LTL is used to describe properties that to be verified by the SPIN model checker. Our contributions are as follows:

- Our proposed TACE mechanism (**Algorithm.1**) provides decentralized BC based cross-domain access control services. It approves every request when BC guarantee that it is not in TAS.
- We have proposed two transaction validation algorithms i.e., **Algorithm.2** and **Algorithm.3**. **Algorithm.2** is used by BC to validate "T.updateP" transaction i.e., no "transitive access" will occur if the request is granted. Similarly, **Algorithm.3** is used by BC to validate "T.updateT" transaction i.e., the given pairs of requests are transitive.
- To provides unbiased and uninterrupted access control across multiple conflict of interest domain, the PAS and TAS are stored on the BC. It alleviates the case if some of the nodes malfunction in the chain. Also, any user can verify any others assigned resources while still preserving user privacy.
- We develop a promela model of our proposed mechanism by using SPIN model checker. We identify four conflicting sequences of execution in cross-domain accesses. These conflicting sequences of execution cause "transitive access". The SPIN model checker verifies our mechanism against "transitive access" across multiple conflict of interest domains.

The rest of the paper is organized as follows. In Section II, we discuss and summarize preliminaries and related works. In section III, we present our proposed architecture. In section IV, we discuss the use cases. Section V presents the verification results. In section VI, conclusions are drawn.

II. Preliminaries and Related Works

A. Blockchain (BC)

The BC was invented by Satoshi Nakamoto [8], where cryptocurrecy bitcoin is it first implementation. The cryptography, BC protocols and peer-to-peer network are the major building blocks of BC. The following are the main features of BC.

1) Decentralized and distributed ledger: Every transition is recorded on a shared ledger. Every node maintains a copy of the ledger. These distributed ledgers are synchronized by replication.

2) Transparency: Every BC block is tamper-proof and available to all the parties in the BC for audit [9].

3) consensus: Every block is validated independently via a consensus mechanism. This mechanism is called mining and works without the use of central trusted authority.

4) Security: Each block contains the hash value of previous blocks in the chain. When a block tampers, its hash will change. Therefore, it makes all following blocks invalid. The attacker must re-calculate all the following blocks hashes. Also, it must re-execute the consensus algorithm, like Proof of Work (PoW) [10] or Proof of Work (PoS) [11], to validate the blocks.

B. How BC Works

BC is a chain of blocks linked together by cryptographic hashes. Every block contains a hash of the previous block. The genesis block is the first block in the chain and does not contain a hash of any block. Every block contains PoW, a timestamp, the hash of the previous block, block header and a number of transactions.

Here, miners are BC network nodes having greater computing power. Each miner validates the newly created block and adds to the BC. In bitcoin, approximately every 10 minutes, a new block is added to the BC [12]. Miners compete by solving a complex mathematical problem called PoW. The miner with more computing power has a greater chance of winning the competition. The newly created block is broadcasted in the BC network. Every node has a copy of BC and adds the block

after PoW verification. Miners receive a transaction fee for spending their computing power to find PoW.

The consensus mechanism enables BC nodes to agree on a one only value. The PoW, PoS are some of the consensus mechanisms used by BC. PoW generates a value which is difficult to solve but easy to verify. For example, generate a hash value having n=4 leading zero. Actually, it is a mechanism to slow down the creation of new blocks. Miners use their computational power to find PoW and get the reward. In PoS, an election process determine a validator for the next block. PoS has replaced miner with a validator. To become a validator the node has to deposit a certain amount of money called stack. The stack size determines a validator for the next block. The validator mint a block while miner mine the block.

C. Public and Private BC Networks

The BC networks can be divided into three categories i.e., public BC, private BC and consortium BC [13].

1) Public BC Network: It is also called permissionless BC. It is a fully decentralized BC network. Anyone in the world can send and read from BC. Anyone can participate in the mining process. The public BC requires a huge amount of computational power to run a consensus algorithm called PoW. Some examples of public BC are bitcoin[1] and ethereum[2].

2) Private BC Network: It is also called permissioned BC. It is an invitation-only network. Every member requires permission before reading or writing. Therefore, organization data is not publicly accessible. So, it lacks the decentralized feature of blockchains. However, private BC is faster and more efficient than public BC which require more computational power to find PoW. The Hyperledger[3] is an example of private BC

3) Consortium BC Network: It is considered partially decentralized where the consensus mechanism is executed by a set of selected nodes. It is a perfect platform for organizational collaboration. Like private BC, consortium BC can provide greater efficiency and transaction privacy. The r3[4] and EWF[5] are examples of a consortium network.

D. Access Control through BC

In [14], the authors have proposed a BC based access management architecture for IoT. The core components of the proposed architecture are managers, wireless sensor networks, agent node, BC network, management hub and smart contract. The IoT devices use "management hub" to access information from a smart contract at BC. The smart contract has a set of access policies. It provides the requested information after successful validation of certain policies. In [15], the authors have proposed a BC based access control. Every owner defines access the policy for his resource and publishes on the BC. The BC allows a user to access a resource after successful verification of access policies. The proposed framework allows a user to transfer access right to another user using the Right Transfer Transaction (RTT). All the policies and the rights transfer are visible on the blockchain. Therefore, any user can verify at any time who can access a particular resource.

The authors in [16] have proposed a BC based "FairAccess" framework. The proposed access control framework is decentralized and privacy preserving. Each owner defines the access policy for his protected resource. The owner stores newly created access policy at the BC using the "GrantAccess" transaction. The requester can access the resource using the "GetAccess" transaction. It can further delegates the permission using "DelegateAccess" transaction.

A BC based lightweight framework has been proposed for IoT in [17]. The proposed framework consists of cloud storage, smart home and overlay network. IoT devices have limited computing power. Therefore, the proposed framework did not have coin and PoW. The authors believe that still the BC security and privacy features are maintained. The authors have further elaborated the core components of their previously proposed smart home framework in [18]. The authors have considered a scenario of smart homes connected to each. Each smart home has a local BC. All the IoT devices installed in a smart home are assumed trusted. These IoT devices use shared key authentication. The shared key authentication is not a strong security measurement.

In [19], the authors have proposed a BC based decentralized framework called BlendCAC for access control in IoT. A capability token is defined for cross-domain delegation and revocation of permission. Each domain owner defines a smart contract at BC to manage the capability tokens. Similarly, in [20], the authors have proposed a BC based cross-platform collaboration framework for IoT called IoT Passport. IoT Passport consists of local and global trust domains. A global trust domain contains multiple organizations. Therefore, global BC is used to share resource/data among organizations of the global trust domain. Each organization has a local trust domain which consists of IoT devices. Therefore, local BC is used to share resource/date among IoT devices in a local trust domain. Smart contracts i.e., Collaborative Rule Contracts are used to provide trust between organizations using cross-platform trust policies.

All the above discussed architectures have targeted access control in either single or multiple domains. A single domain did not have a conflict of interest problem. However, cross-domain access control can suffer from a conflict of interest problem. These conflict of interest problems produce "transitive access" which can cause leakage of information. All the proposed architectures did not consider "transitive access". However, our proposed mechanism finds a conflict of interest problems and denies "transitive access".

E. Cross-Domain Access Control

In [3], authors have proposed a formal framework for cross-domain access control. Authors have considered a scenario where collaborating organizations have less trust relationship. They proposed a trusted third party called delegation service for cross-domain access control. A single trusted delegation

[1][Online]. Available: https://bitcoin.org/
[2][Online]. Available: https://www.ethereum.org/
[3][Online]. Available: https://www.hyperledger.org
[4][Online]. Available: https://www.r3.com/
[5][Online]. Available: http://energyweb.org/

service is a single point of failure. Also, it can make bias decisions in the delegation of permission to users.

The authors in [21], have proposed a capability-based access control for IoT. Initially, the system grants a limited amount of permissions to the requester. The system delegates additional permissions to the requester when required. These permissions are delegated to the requester using capability. The capability is defined as an object with a set of permissions. Similarly, in [5] the authors have proposed an authorization framework for IoT called IoT-OAS. The authors have proposed a centralized OAuth based authorization service called OAS. OAS is an external entity that validates user requests on behalf of the service provider.

All the above discussed architectures have targeted cross-domain access control. However, they did not consider conflict of interest problem arises in cross-domain access control.

The authors in [22], have proposed a new access control model for data mining environment. It is based on the Chinese Wall Security Policy (CWSP) model [23]. It has re-defined the definition of "conflict of interest class". According to CWSP, all airline companies belong to the same conflict of interest because they are competitors. However, airline company α_1 and petroleum company β_1 are not competitors, so they belong to different conflict of interest classes. According to the author, if airline company α_1 buy shares in petroleum company β_1, then they must be grouped in the same conflict of interest class because now both have a conflicting interest. The airline company α_1 may withdraw his shares if it knows the negative growth in petroleum company β_1's cash flow. Similarly, in our proposed scenario, the conflict of interest of a consultant changes with his own and other consultants access histories. Therefore, our framework records every user access in an AFG. It allows a user request only if it does not cause leakage of information.

III. Transitive Access Checking and Enforcement

Our Proposed architecture provides BC based access control for conflict of interest domains. Our proposed architecture consists of BC manager, TACE module, smart contract and BC network. The high-level architecture is shown in **Fig.1**.

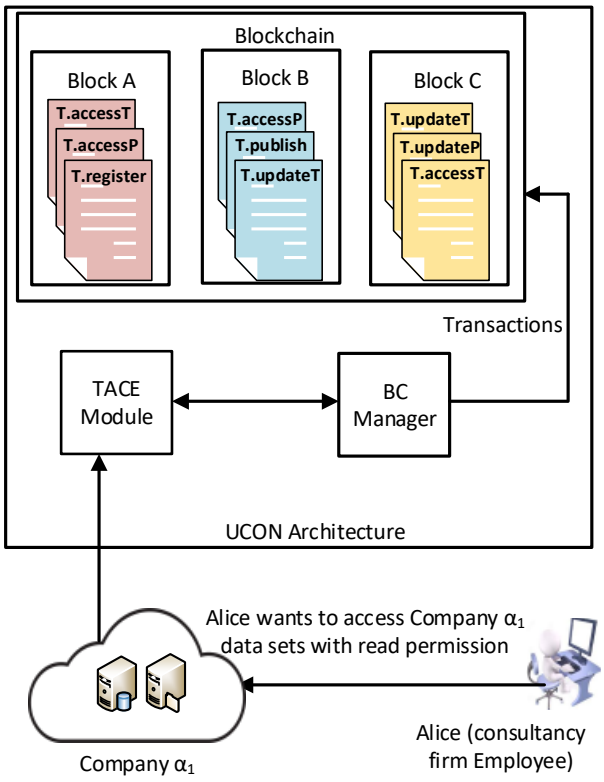


Fig. 1. High-level Architecture

A. BC Manager

BC manager registers user devices/IoT devices with BC. Users use BC transaction to access information i.e., PAS and TAS stored on the BC. The BC manager receives a request from the TACE module and initiates a BC transaction. BC has limited storage capacity. Therefore, access control policies are stored in off-chain storage while their hashes are stored on the BC. The details tasks of BC manager are shown in **Fig.2**. The BC manager performs the following functions.

1) On receiving a request, BC manager generates "T.register" transaction to register a node at BC.
2) Allow the owner to defines access policies for his resource using "T.publish" transaction.
3) Generate "T.accessP" and "T.accessT" transactions, to retrieve user access histories and "transitive accesses" stored at BC respectively.
4) Generate "T.updateP" and "T.updateT" transactions, to add a user approved request and new "transitive access" to the BC block respectively.

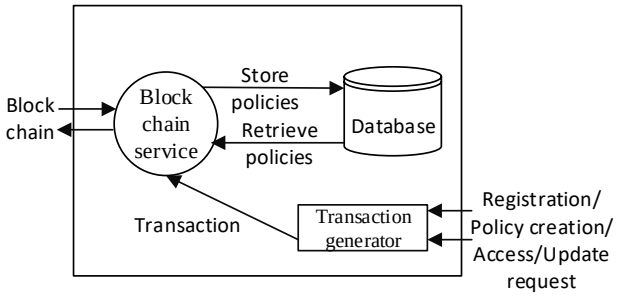


Fig. 2. BC Manager

B. BC Network

Our proposed architecture consists of a consortium BC. Therefore, only members of the coalition organization are allowed to become a member of the BC network. It allows all nodes to read but dedicated nodes to write in the distributed ledge. A pre-defined set of nodes, called validators in the BC network, approve every transaction and save copies of the BC current state. Therefore, data stored at BC is decentralized and tamper resistant. The access control policies and users access

histories are stored on the BC. The transactions are given in **Table.I** with description and transaction format.

TABLE I
BC TRANSACTIONS

Name	Transaction Format	Description
T.register	s_i, $\hbar_c$	To register a node at the BC.
T.publish	s_i, s_j, p_k, $\mathcal{C}$	To publish an access policy at the BC.
T.accessP	s_i, $\mathcal{C}$	To retrieve user access history stored at the BC.
T.accessT	s_i, α_p, p_k, $\mathcal{C}$	To retrieve "transitive accesses" stored at the BC.
T.updateP	s_i, O(β_s or α_p), p_k	To add a user approved request to the block.
T.updateT	{(s_i,O(β_s or α_p),p_k), (s_j,O(β_s or α_p),p_k)}	To add new "transitive access" to the block.

C. TACE Module

Our proposed TACE module is called with every user request. The operations of TACE module are given in "Algorithm.1". It uses PAS to detects "transitive access" and sends it to BC for validation. Each user access histories and "transitive accesses" are maintained at BC. The PAS stores users access histories whereas, TAS stores possible future "transitive accesses". All the authorization decisions are taken by the TACE module. These authorization decisions are based on "transitive access" detected by TACE module. It allows a request only when BC guarantees that "transitive access" will not occur.

D. Smart Contract

In our proposed architecture each organization in the coalition has a smart contract. The operations of the smart contract are defined in "Algorithm.2" and "Algorithm.3". Our proposed BC transaction validation algorithms, i.e., "Algorithm.2" and "Algorithm.3" validate "T.updateP" and "T.updateT" transactions. The "T.updateP" transaction is used to update users access histories Whereas, "T.updateT" transaction is used to update "transitive accesses" stored at BC. After successful validation, a block of transactions is added to the chain. Each block contains a previous block hash, group of transactions and block header.

E. Notations

Table.II contains all the notations used in the proposed research work.

F. Components of Our Model

Our model consists of the following components.

1) Subject Set: $s_i \in S$, where S is a set of all users in the system.

TABLE II
NOTATIONS

Notation	Description
s_i	ith subject.
o_j	jth object.
r_k	kth right.
$\mathcal{C}$	Set of all constraints, e.g., Static Mutually Exclusive Permission (SMEP).
α, β, γ	conflict of interest classes.
$\alpha = \bigcup_{i\in1}^{n} O(\alpha_i) \in \alpha$	all companies dataset in a conflict of interest class α.
$O(\alpha_1) = \bigcup_{i\in1}^{n} o_i(\alpha_1) \in O(\alpha_1)$	all the objects in company α_1 dataset.
$\cup_{i,j=0}^{\infty}\{pol_{si,sj}\}$	all access policies of s_i for s_j.
$\hbar_c$	recently calculated hash value of a user platform.
PAS	a super set of all approved user requests.
TAS	a super set of all "transitive accesses".

2) Object Set: $o_i \in O$, where O is a set of all objects in the system.

3) Rights Set: $r_i \in R$, where R is a set of all rights in the system.

4) Conflict of Interest Class: The conflict of interest class consists of a group of companies that has common business interest i.e., competitors. The conflict of interest classes is denoted By capital letter α, β, γ.

5) Company Dataset: The companies datasets are denoted by O(α_1), O(α_2), O(α_3) if they belong to conflict of interest class α. Similarly, companies datasets are denoted by O(β_1), O(β_2), O(β_3) if they belong to conflict of interest class β.

$$\alpha = \bigcup_{i\in1}^{n} O(\alpha_i) \in \alpha \tag{1}$$

For example
α = {O(α_1), O(α_2), O(α_3) ....}, Where

$$O(\alpha_1) = \bigcup_{i\in1}^{n} o_i(\alpha_1) \in O(\alpha_1) \tag{2}$$

6) legal Access: To get a resource by authorized user is called legal access. The set of legal access is denoted by L. $l_i \in L$, where L is a set of all legal accesses in the system.

7) Illegal Access: To get a resource by an unauthorized user is called illegal access. The set of illegal access is denoted by I.
$i_j \in I$, where I is a set of all illegal accesses in the system.

8) Transitive Access: To get an unpermitted resource through some legitimate subject and/or object is called "transitive access". "Transitive access" is discussed in detail in section IV.
T $\subseteq$ I and $t_i \in T$, where T is a set of all "transitive accesses"

in the system.
The "transitive access" is a illegal access because it can cause leakage of information when allowed.

9) Relationship among Legal, Illegal and Transitive Accesses: Transitive Access set is a subset of Illegal Access set. In other words, Illegal Access set is a superset of Transitive Access set.

$$T \subseteq I \; or \; I \supseteq T \tag{3}$$

Legal Access set and Transitive Access set are disjoint sets. Similarly, Legal Access set and Illegal Access set are disjoint sets.

$$L \cap T = \emptyset \; and \; L \cap I = \emptyset \tag{4}$$

10) Process Request (PR): Generally, PR consist of subject, object and rights i.e., (s_i, o_l, r_y), where $s_i \in S$, $o_l \in O$, $r_y \in R$. Here, o_l object may belong to any conflict of interest class. The PR with company label can be written as $(s_i, O(\alpha_1), r_y)$, where O(α_1) is a dataset of company α_1.

11) Process Access Set (PAS): It is a relation between subjects, objects and rights.

$$PAS \subseteq \{S \times O \times R\} \tag{5}$$

When a request (s_i, O(α_1), r_y) is approved, PAS is extended by (s_i, O(α_1), r_y).

$$\overline{PAS} = PAS \cup (s_i, O(\alpha_1), r_y) \tag{6}$$

where, $s_i \in S$, s_i is a requester belongs to a set of Subject *S*, O(α_1) denotes dataset of company α_1. The $r_y \in R$, r_y is a subset of rights belongs to a super set of rights *R* i.e., (read, write, execute, print etc).

12) Transitive Access Set (TAS): It is a relation between the requests in the PAS.

$$TAS \subseteq \{PAS \times PAS\} \tag{7}$$

Where PAS is a set of requests that can cause "transitive access" i.e., $\{(s_i, O(\alpha_1), r), (s_j, O(\alpha_2), r)\}$. After each request approval, TAC algorithm traverses AFG to find "transitive access". Then, TAS is extended when new transitive requests are found.

$$\overline{TAS} = TAS \cup \{(s_i, O(\alpha_1), r), (s_j, O(\alpha_2), r)\} \tag{8}$$

13) Policy Set (Pol): *Pol* contains all the policies in a system.

$$Pol \supseteq \cup_{i,j=0}^{\infty} \{pol_{si,sj}\} \tag{9}$$

14) ObjectAccesssed: It is a function which takes a subject as input and retrieves a list of objects accessed by that subject.

$$ObjectAccessed : s_i \rightarrow l_{s_i} \tag{10}$$

Where l_{s_i} is a set of object accessed by s_i and either $l_{s_i} \subseteq$ O or l_{s_i}=$\emptyset$.

15) Accessed: It is a function which takes a subject, object and permission i.e., $(s_i, O(\alpha_1), r)$ as input and generates 0, 1 or 2 values. The output "0" means subject did not access the object. The output "1" means that subject has accessed the object with permission r. The output "2" means that subject has accessed the object with permission w.

$$Accessed : (s_i, O(\alpha_1), r) \rightarrow \{0/1/2\} \tag{11}$$

16) Requesting: It is a function which takes a subject, object and permission as input and generates a request i.e., $(s_i, O(\alpha_1), r)$.

$$Requesting : \{subject, object \; and \; permission\} \rightarrow (s_i, O(\alpha_1), r) \tag{12}$$

17) Access Flow Graph: It is a directed graph, where the node represent subject or object. A directed edge between subject and object represent that a subject has accessed object with a right i.e., read, write. AFG is stored in PAS at BC. AFG is shown in **Fig.3**.

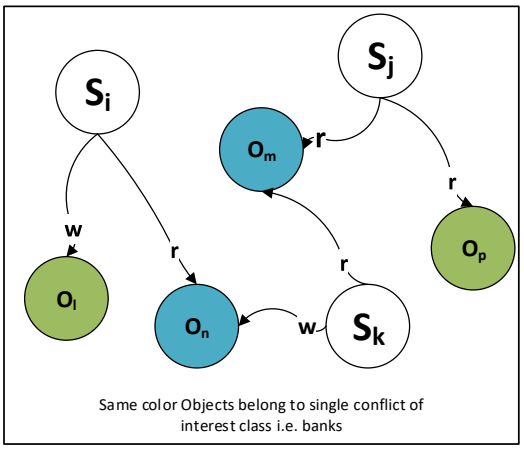


Fig. 3. Access Flow Graph

18) Axiom 1: A subject i.e., s_i can be assign to any company if it is not assigned to any other company before.

$$ObjectAccessed : s_i \rightarrow \{\} \tag{13}$$

19) Axiom 2: A subject i.e., s_i can not be assign to two companies in same conflict of interest class.

$$l_{s_i} \subseteq O(\alpha_1) \; or \; l_{s_i} \subseteq O(\alpha_2) \; and \; l_{s_i} \nsubseteq (O(\alpha_1) \cup O(\alpha_2)) \tag{14}$$

20) Axiom 3: Two subjects i.e., s_i and s_j can not be assign to the same company in conflict of interest class if they are already assigned to different companies in some other conflict

of interest class.

$$l_{s_i} \subseteq O(\alpha_1)\ and\ l_{s_j} \subseteq O(\alpha_2)\ then \\ (l_{s_i} \cap l_{s_j}) \nsubseteq O(\beta_1) \tag{15}$$

21) Axiom 4: Two subjects i.e., s_i and s_j can not be assign to two different companies in same conflict of interest class if both are already assigned to same company in some other conflict of interest class.

$$l_{s_i} \subseteq O(\beta_1)\ and\ l_{s_j} \subseteq O(\beta_1)\ then \\ (l_{s_i} \cup l_{s_j}) \nsubseteq (O(\alpha_1) \cap O(\alpha_2)) \tag{16}$$

G. Transitive Access Checking and Enforcement Algorithm

The **Algorithm.1** checks all the four possible conflicting execution sequences as discussed in section IV. A for every request (s_i,O(β_1 or α_1 or α_2),r|w). If the requests are transitive then they must be denied. All the "transitive accesses" are stored in TAS on the BC. It uses "T.accessT" transaction to access TAS. To approve a user request it sends "T.updateT" transaction to BC for recalculation and validation of "transitive access". It generates a BC transaction to extends PAS with a pair of s_i and s_j requests after a request is approved. It calls *ObjectAccessed* function to retrieve a list of objects accessed by s_i and all the other subjects in PAS. Then, it compares user requests in PAS to identify "transitive access". We have identified four conflicting sequences of executions, so four conditions are used. Then, it sends "T.updateT" transaction to BC for "transitive access" identification and updating of TAS.

H. Transaction Validation Algorithms

We have proposed two algorithms for BC transactions validation. **Algorithm.2** is used by BC to validate "T.updateP" transaction whereas, **Algorithm.3** is used by BC to validate "T.updateT" transaction. The **Algorithm.2**, takes a user request and PAS as input and verifies that no "transitive access" will occur if the request is granted. Similarly, the **Algorithm.3** verifies that the pair of requests are transitive and add it to BC block. The computation logic of BCON architecture is shown in **Fig.4**.

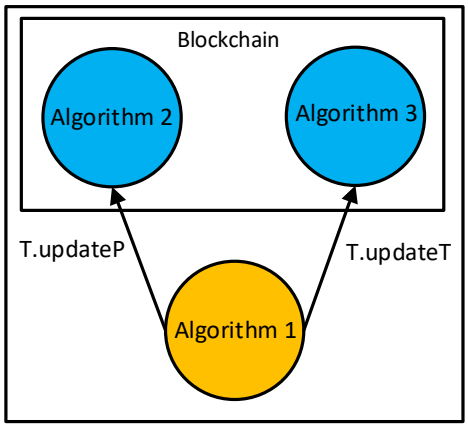


Fig. 4. Computation Logic of BCON Architecture

Algorithm 1 Transitive Access Checking and Enforcement Algorithm

Input: (s_i,O(β_1 or α_1 or α_2),r|w), **PAS**
Output: permit(s_i,O(β_1 or α_1 or α_2),r|w) or deny(s_i,O(β_1 or α_1 or α_2),r|w)
Initialise bool variable *isAdded* = ***FALSE***;
Initialise bool variable *isUpdated* = ***FALSE***;
Initialise bool variable *isNottransitive* = ***FALSE***;
Retrieve **TAS** from BC using **T.accessT** transaction
if {(s_i,O(β_1),r|w) $\wedge$ (s_j,O(β_1),r|w)} $\in$ **TAS** **then**
  deny(s_i,O(β_1),r|w)
else if {(s_i,O(α_1),r|w) $\wedge$ (s_j,O(α_2),r|w) $\in$ **TAS**} **then**
  deny(s_i,O(α_1),r|w)
else
  Send **T.updateP** transaction to BC and call **Algorithm.2** at BC to validate the transaction.
  if *(isValidated* $\equiv$ ***TRUE)*** **then**
    ObjectAccessed: $s_i \rightarrow \{l_{s_i}\}$
    call **ObjectAccessed** function for every subject in **PAS**
    ObjectAccessed: $s_j \rightarrow \{l_{s_j}\}$
    Retrieve **PAS** from BC using **T.accessP** transaction
    for {i=0 to size l_{s_i} } **do**
      for {j=0 to size l_{s_j} } **do**
        if {(s_i,O(α_1),r) $\wedge$ (s_j,O(α_2),w)} $\in$ **PAS** ***OR*** {(s_i,O(α_1),w) $\wedge$ (s_j,O(α_2),r)} $\in$ **PAS** ***OR*** {(s_i,O(β_1),r) $\wedge$ (s_j,O(β_1),w)} $\in$ **PAS** ***OR*** {(s_i,O(β_1),w) $\wedge$ (s_j,O(β_1),r)} $\in$ **PAS** ***AND*** {O(α_1) $\wedge$ O(α_2)} $\in \alpha$ **then**
          Send **T.updateT** transaction to BC. call **Algorithm.3** at BC to validate the transaction.
          if *(isAdded* $\equiv$ ***TRUE)*** OR *(isNottransitive* = ***TRUE)*** **then**
            permit(s_i,O(α_1),r|w)
          else
            deny(s_i,O(α_1),r|w)
          end if
        else
          deny(s_i,O(α_1),r|w)
        end if
      end for
    end for
  else
    deny(s_i,O(α_1),r|w)
  end if
end if

IV. Use Cases

Let us consider a consultancy firm, which provides consultancy service to different companies. Firm stores all information of the companies in a three level hierarchically filing system as shown in **Fig.5**.

- Low-level contains information about individual items and each belongs to a single corporation.
- Middle-level contains all objects that belong to the same corporation, which form the company dataset.

Algorithm 2 "T.updateP" Transaction Validation Algorithm

Input: **T.updateP**(s_i,O(β_1 or α_1 or α_2),r|w), *PAS*
Output: bool variable *isValidated* and ***PAS*** = $\overline{\textbf{\textit{PAS}}}$ $\cup$ (s_i,O(β_1 or α_1 or α_2),r|w)
Initialise bool variable *isValidated* = ***FALSE***
if $\{l_{s_i} \subseteq$ O(α_1) ***OR*** $l_{s_i} \subseteq$ O(α_2) ***AND*** $l_{s_i} \subseteq$ (O(α_1) $\cup$ O(α_2)) $\}$ **then**
    return *isValidated* = ***FALSE***
else if $\{l_{s_i} \subseteq$ O(α_1) ***AND*** $l_{s_j} \subseteq$ O(α_2) ***AND*** ($l_{s_i} \cap l_{s_j}$) $\subseteq$ O(β_1)$\}$ **then**
    return *isValidated* = ***FALSE***
else if $\{l_{s_i} \subseteq$ O(β_1) ***AND*** $l_{s_j} \subseteq$ O(β_1) ***AND*** ($l_{s_i} \cup l_{s_j}$) $\subseteq$ (O(α_1) $\cap$ O(α_2))$\}$ **then**
    return *isValidated* = ***FALSE***
else
    $\overline{\textbf{\textit{PAS}}}$ = ***PAS*** $\cup$ (s_i,O(β_1 or α_1 or α_2),r|w) and return *isValidated* = ***TRUE***
end if

Algorithm 3 "T.updateT" Transaction Validation Algorithm

Input: **T.updateT** $\{(s_i$,O(β_1 or α_1 or α_2),r|w),(s_j,O(β_1 or α_1 or α_2),r|w)$\}$, *TAS*
Output: bool variable *isAdded*
Initialise bool variable *isAdded* = ***FALSE***
if $\{(s_i$,O(α_1),r) $\wedge$ (s_j,O(α_2),w)$\}$ $\in$ **PAS** and $\{$O(α_1) $\wedge$ O(α_2)$\}$ $\in \alpha$ **then**
    $\overline{\textbf{TAS}}$ = **TAS** $\cup$ (s_i,O(β_1),w),(s_j,O(β_1),r) and return *isAdded* = ***TRUE***
else if $\{(s_i$,O(α_1),w) $\wedge$ (s_j,O(α_2),r)$\}$ $\in$ **PAS** and $\{$O(α_1) $\wedge$ O(α_2)$\}$ $\in \alpha$ **then**
    $\overline{\textbf{TAS}}$ = **TAS** $\cup$ $\{(s_i$,O(β_1),r),(s_j,O(β_1),w)$\}$ and return *isAdded* = ***TRUE***
else if $\{(s_i$,O(β_1),r) $\wedge$ (s_j,O(β_1),w)$\}$ $\in$ **PAS** and $\{$O(α_1) $\wedge$ O(α_2)$\}$ $\in \alpha$ **then**
    $\overline{\textbf{TAS}}$ = **TAS** $\cup$ $\{(s_i$,O(α_1),w),(s_j,O(α_2),r)$\}$ and return *isAdded* = ***TRUE***
else if $\{(s_i$,O(β_1),w) $\wedge$ (s_j,O(β_1),r)$\}$ $\in$ **PAS** and $\{$O(α_1) $\wedge$ O(α_2)$\}$ $\in \alpha$ **then**
    $\overline{\textbf{TAS}}$ = **TAS** $\cup$ $\{(s_i$,O(α_1),w),(s_j,O(α_2),r)$\}$ and return *isAdded* = ***TRUE***
else
    return *isAdded* = ***FALSE***
end if

- Top-level groups all companies dataset whose corporations are in competition. We can call each such group as a conflict of interest class.

Scenario I: Suppose α and β are two conflict of interest classes. Class α has two competitors, company α_1 and company α_2. Similarly, β_1 belongs to conflict of interest class β. Consultancy firm assigns two consultants Alice and Bob to β_1 with read (r) and write (w) permissions respectively. Later on, Alice and Bob may also generate a request for α_1 and α_2 datasets with r/w permission respectively as shown in **Fig.6**. If access is granted, then it is highly likely that information of α_1 may be leaked to α_2. So either the request of Alice or Bob must be denied to solve the conflict of interest problem.

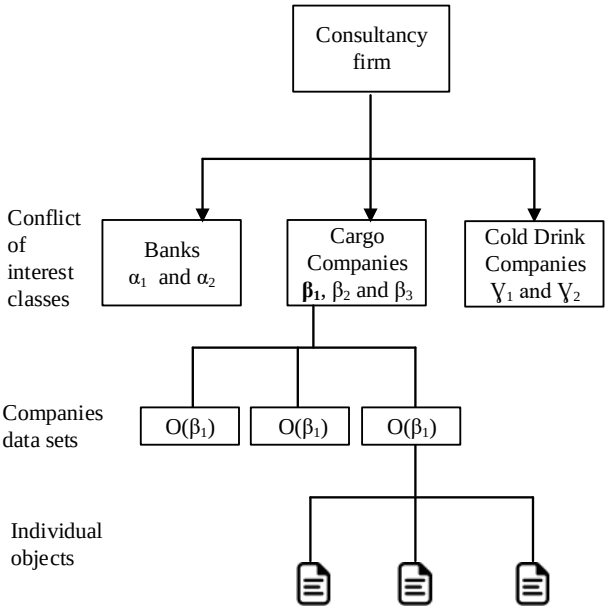


Fig. 5. Three level hierarchy of Consultancy Firm information

Scenario II: Now let discuss this problem in a different scenario such as the concurrent environment. Alice and Bob have already accessed β_1 dataset with "r" and "w" permissions respectively. Suppose Alice is requesting for α_2 dataset with "w" and at the same time Bob generates a "r" request for α_1. If such concurrent requests are not properly controlled, then both processes may enter into accessing state. Such concurrent requests can cause "transitive access".

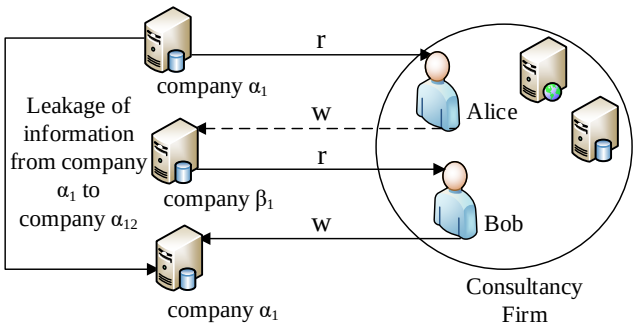


Fig. 6. Illegal Access

A. Conflicting Sequences of Execution

In the above scenarios, there are four conflicting sequences of execution. To describe these conflicting sequences of execution, we have used a two-dimensional matrix. The first row of the matrix consist of three different companies datasets and first column contains two users Bob and Alice. Two actions *Accessed()* and *Requesting()* are used in the matrix cells. Each action has a parameter either r or w. The following are the

four sequences of execution that cause the conflict of interest problem in cross-domain accesses.

1) Execution Sequence 1: Suppose Bob has accessed α_1 datasets with "r" permission and Alice has accessed α_2 datasets with "w" permission as shown in **Table.III**. Now Bob and Alice are requesting for β_1 with "w" and "r" permissions respectively. Both the requests are conflicting because information of α_2 may leak to α_1 if both the requests are allowed. For example Bob can read data from α_1 and write to β_1. Alice can read from β_1 and write to α_2. The leakage of information from α_1 to α_2 is shown in **Fig.7**.

TABLE III
EXECUTION SEQUENCE 1

S/O	company α_1	company α_2	company β_1
Bob	Accessed(r)		Requesting(w)
Alice		Accessed(w)	Requesting(r)

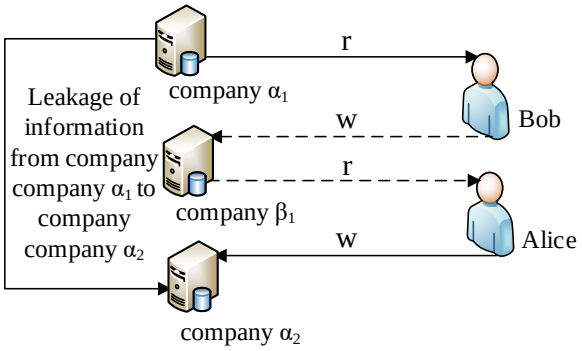


Fig. 7. Executing Sequence 1

2) Execution Sequence 2: Suppose Bob has accessed company α_1 datasets with "w" permission and Alice has accessed company α_2 datasets with "r" permission as shown in **Table.IV**. Now Bob and Alice are requesting for company β_1 with "r" and "w" permissions respectively. The information of company α_2 may leak to company α_1 if both the requests are allowed. The leakage of information from company α_2 to company α_1 is shown in **Fig.8**.

TABLE IV
EXECUTION SEQUENCE 2

S/O	company α_1	company α_2	company β_1
Bob	Accessed(w)		Requesting(r)
Alice		Accessed(r)	Requesting(w)

3) Execution Sequence 3: Suppose Bob and Alice have accessed company β_1 datasets with "w" and "r" permissions respectively as shown in **Table.V**. Now Bob is requesting for company α_2 with "r" and Alice is requesting for company α_1 with "w" permission. If both the requests are allowed then,

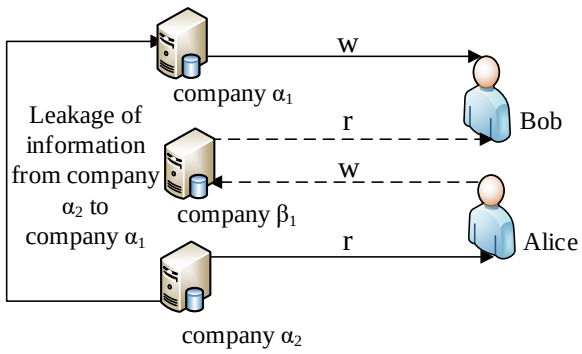


Fig. 8. Execution Sequence 2

it can cause leakage of information. For example, Bob can read data from the company α_1 and write to the company β_1. Alice can read from the company β_1 and write to the company α_2. The leakage of information from the company α_1 to the company α_2 is shown in **Fig.9**.

TABLE V
EXECUTION SEQUENCE 3

S/O	company α_1	company α_2	company β_1
Bob	Requesting(r)		Accessed(w)
Alice		Requesting(w)	Accessed(r)

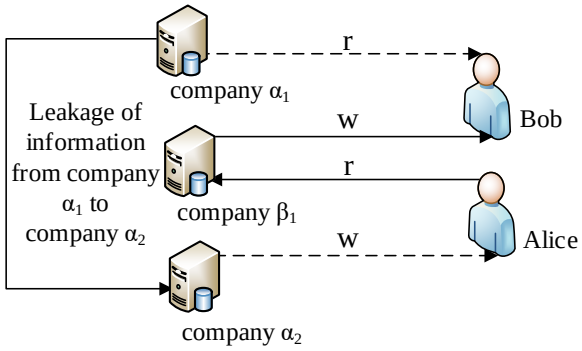


Fig. 9. Execution Sequence 3

4) Execution Sequence 4: Suppose Bob and Alice have accessed company β_1 datasets with "r" and "w" permissions respectively as shown in **Table.VI**. Now Bob is requesting for company α_1 with "w" and Alice is requesting for company α_2 with "r" permission. Both the requests are conflicting and may cause leakage of information if both the requests are allowed. The leakage of information from company α_2 to company α_1 is shown in **Fig.10**.

TABLE VI
EXECUTION SEQUENCE 4

S/O	company α_1	company α_2	company β_1
Bob	Requesting(w)		Accessed(r)
Alice		Requesting(r)	Accessed(w)

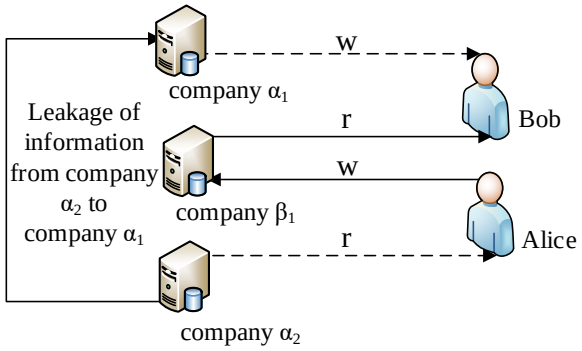


Fig. 10. Execution Sequence 4

B. Security Analysis

In this section, we discuss the security of the proposed architecture using Confidentiality, Integrity, Availability, Authorization and Non-repudiation (CIAAN) model. We have integrated BC technology in UCON architecture to achieved the CIAAN model security requirements. The analysis of the security parameters is summarized in **Table.VII**.

1) Confidentiality: It means to hide information from unauthorized user. In our proposed architecture, the communication between BC manager, TACE module and BC is secure through public key encryption.

2) Integrity: It ensures that data is not changed in an unauthorized way. In BCON, BC cryptographic hash function i.e., SHA-256 is used to provide data integrity. Each BC block contains the previous block hash value. The block hash value changes if a transaction in a previous block is rollbacked, deleted or tempered. Supposed an adversary has changed a transaction in a certain block in the chain. As a result, all subsequent blocks become invalid because they contain an old hash of the tempered block. Therefore, adversary requires huge computation power to re-execute SHA-256 for all following blocks.

3) Availability: It ensures that the service or information is always available to legitimate users. Each node in the BC network has a copy of the distributed ledger. These copies are synchronized through replication. Supposed adversary damages a copy of BC ledger at a certain node. The node can restore his ledger from their neighbor nodes using BC replication mechanism.

4) Authentication and Authorization: It ensures that a user is the individual who claims himself to be. We assume an authentication mechanism called bubbles-of-trust [24] due to low power and fast execution requirements. Similarly, authorization ensures that the user has the right to do a certain task. In proposed architecture BC performs cross-domain user authorization. The cross-domain access control has a conflict of interest problems. These conflict of interest problems produce "transitive access" which can cause leakage of information. In our proposed architecture, access control policies stored at BC are used to deny "transitive access".

5) Non-repudiation: It ensures that all users can be held responsible for their activities. Thus, later on, the user cannot be able to deny his action. BC is an immutable chain of blocks. The proposed architecture provides non-repudiation to the data stored at BC using digital signatures.

6) No Single Point Failure: In existing architectures, the central trusted entity validates all the transactions. Thus, resulting in performance bottlenecks and a single point of failure. Our proposed BCON architecture integrates BC and alleviates the need of a central trusted entity.

7) Denial-of-Service Attack (DoS) and Distributed Denial-of-Service (DDoS) Attack : In existing architectures, the central trusted entity validates all the transactions. The centralized entity resulting in performance bottlenecks. Such performance bottlenecks appear with traffic increase because the server has a finite number of the port to listen to clients request. Therefore, centralized systems are easy targets of different attacks like DoS attack and DDoS attack.

Similarly, in existing BC based access control architecture, suppose an attacker got access to a user/IoT device and installed malicious software. Now, the attacker sends an access request to the BC using the compromised user/IoT device. In our proposed architecture, the BC performs user/IoT device platform verification before authorization. Thus, both the current and stored hashes will match. Hence, the BC will deny the attacker access to the protected resource.

8) Efficiency: Public BC has problems with scalability and efficiency. Companies in a virtual coalition cope with hundred of transaction per second. Public BC take more time to process a block i.e., bitcoin takes 10 minutes [25], while Ethereum takes 14 seconds [26]. The efficiency of BC depends on consensus algorithm. BCON is built on consortium BC. Consortium BC has lower latency and higher throughput than a public BC. However, BCON major consideration is security.

C. Comparisons between Exiting and BCON Architectures

The comparisons among existing related and proposed architectures are given in **Table.VIII**. .

V. MODELING AND VERIFICATION

A. Implementation Details

The process, data object and message channels are the main building block in a PROMELA implementation [27]. The core components of our promela model are described in the following subsections.

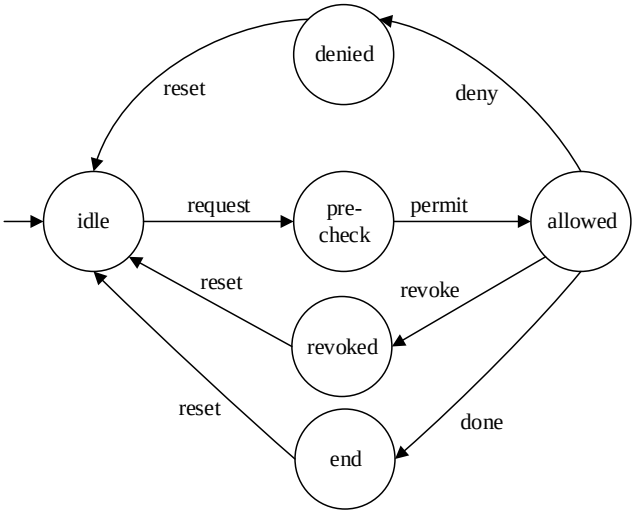
TABLE VII
EVALUATION OF SECURITY PARAMETERS

Parameters	Description
Confidentiality	The communication between BC manager, TACE module and BC is secure through Public key encryption.
Integrity	SHA-256 hash function is used to provide data integrity.
Availability	The available of the system is maintained through BC ledgers replication. Also, platform verification is performed before user authorization to avoid attacks on avilability likes DoS, DDOS.
Authorization	"Transitive access" is identified and denied using access policies stored at BC.
Non-repudiation	Achieved through digital signature. Every transaction is cryptographically signed.

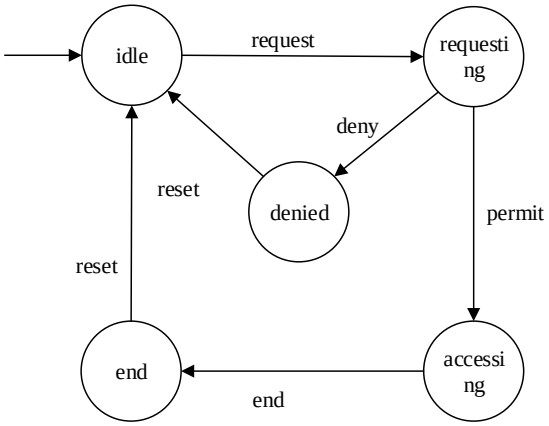
TABLE VIII
COMPARISONS BETWEEN EXITING AND BCON ARCHITECTURES

Mechanism /Property	BC based (No cross -domain access) Framew-orks [14], [16]	BC based (cross -domain access) Framew-orks [19] [20]	Central-ized (cross -domain access) Framew-orks [4] [5]	UCON BC based (cross -domain access) Framew-ork
Conflict of Interest	No	No	No	Yes
Transitive Access	No	No	No	Yes
Platform Verific-ation	No	No	No	Yes
Decentra-lized	Yes	Yes	No	Yes
Fault tolerance	Higher	Higher	Lower	Higher
Impleme-ntation	[14] Ether-eum [16] Bitcoin	[19] Eth-ereum [20] No Impleme-ntation	central-ized server	Hyper-ledger
Consensus mechanism	[14] PoW [16] proof of concept	PoW	No consens-us	Practical Byzantine fault tolerance (PBFT)
Efficiency	[14] closer to main ethereum network [16]dep-ended on bitcoin network	[19] dep-ended on ethereum network [20] No perform-ance eva-luation	depended on central entity	Higher than both ethereum and bitcoin BC

1) BC Process: In our model, there is one BC process. The BC process consists of "idle", "revoked", "pre-check", "allowed", "denied" and "end" states. The Buchi automaton [28] of BC process is shown in **Fig.11(a)**. The initial state is an "idle" state. At "idle" state, the BC process receives access request and changes the state to "pre-check" state. In "pre-check" state the BC process executes TAC algorithm and add conflicting requests to TAS. Then the BC process moves into "allowed" state and executes TAE algorithm. The TAE uses TAS and either allow or deny the user process. The user process moves into "accessing" state if allowed by TAE. Otherwise, the BC process moves into "denied" state. At "accessing" state, the user process moves into "revoked" or "end" state based on "revoke" or "end" event respectively. Then, the BC process makes a transition into "idle" state on "reset" event and repeat again for another request.



a. BC State Transition



b. User process State Transition

Fig. 11. User Process and Blockchain State Transition Diagrams

2) User Process: In our model, there are two user processes i.e., Alice and Bob. Each process consists of "idle", "request-ing", "end", "denied" and "accessing" states. Both processes have similar state transition diagrams. The buchi automaton of the user process is shown in **Fig.11(b)**. The user process moves

from "idle" state to "requesting" state to generate an access request. In "requesting" state, user process generates an access request i.e., (s_i, o(a1), r_y). At this state, the user process waits for the BC process to run TAC and TAE algorithms. If the user request is not in conflict with other users requests, then the process moves into "accessing" state. Finally, the user process moves into "end" state when either the permission is revoked by the BC process or user itself stop accessing the resource. Finally, the user process moves into "idle" state to repeat this process for another request. If the user request is in conflict with other users requests, the process makes a transition into "denied" state.

3) Adversary Process: In our model, we have defined one Adversary process. The adversary is a compromised user/IoT device. Therefore, the Adversary, Alice and Bob processes have similar state transition diagrams. The Buchi automaton of the Adversary process is shown in Fig.11(b).

4) Message channels: Our model consists of five channel variables. The messages are exchanged between processes using channel variables. We have declared four types of messages.

5) Data Objects: In our model, we have defined two data structures i.e., single and two-dimensional arrays. The single dimensional array stores platform hashes whereas the two-dimensional array stores access history of the users. The first row of the two-dimensional array contains company dataset i.e., object whereas the first column contains users i.e., subject as shown in Tables (**Table.III, Table.IV, Table.V and Table.VI**).

B. Simulation results

The proposed model simulation result is shown in **Fig.12**. We have run three user processes i.e., Alice, Bob and Adversary, and one BC processes concurrently. The Adversary process is a user process having some malicious software installed. The process execution path is shown by the vertical line. The boxes on the vertical lines show the process execution steps. The message passing between processes is shown by the cross arrow between boxes. The simulation result shows that adversary is unable to access the protected resource.

C. Never Claims / LTL formulas Verification

A never claim is used to define system behavior. It consists of a preposition or boolean expression. It is generated by SPIN model checker from user-defined LTL formula. The LTL formula is used to specify properties that must be proved by the model. We have checked the following LTL formulas against our model.

Our proposed model performs user platform verification using platform hashes before authorization. The platform hashes are stored at BC. Every request contains the current platform hash value. Hence, BC compares current platform hash with store hash. When the platform hashes match, then the user request is checked against the four conflicting execution sequences for "transitive access".

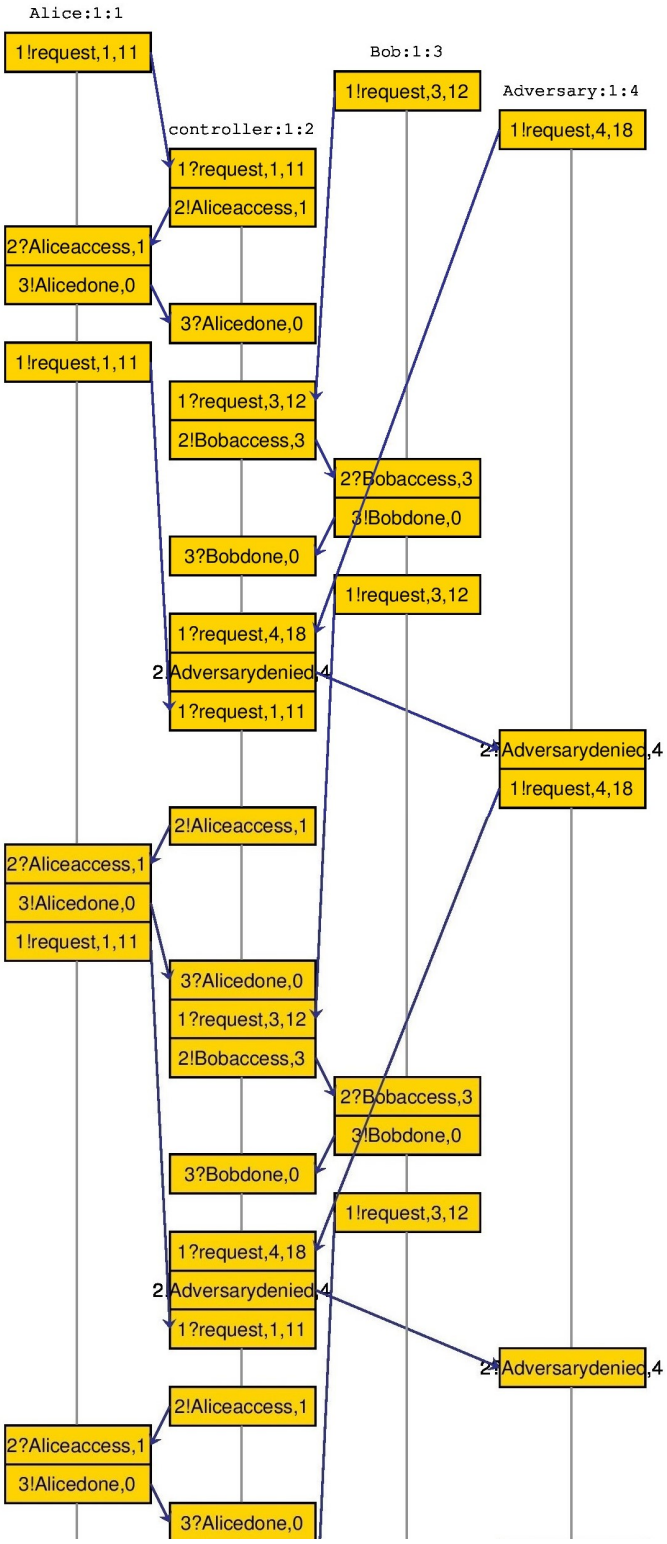


Fig. 12. Simulation

Suppose an adversary has installed malicious software on the user device. Now, the adversary wants to lunch an attack on other devices in the network. So, it sends an access request for a resource using the compromised device. The service provider forwards user request to BC for platform and access control policies verification. Therefore, in the proposed access

control mechanism the BC compares received platform hash with the stored hash. The current hash value is changed and will not match. Therefore, the adversary is denied to access the resource. The "platform verification" property describes that if the platform hashes do not match still Adversary process is able to move into "accessing_state". The SPIN model checker generated an error which means that Adversary process is not able to move into accessing state. The verification result of the SPIN model checker and "never claim" of the "platform verification" property is shown in **Table.IX** and **Fig.13** respectively.

1) Platform Verification Property (C1): {⋄ (p && q) → r}
#define p (j !== hash[3])
#define q (x == 4)
#define r (Adversary@accessing_state)

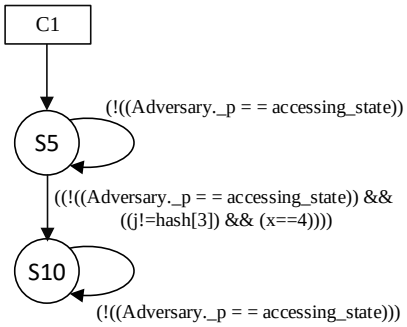


Fig. 13. Automata view of "Platform Verification" property

TABLE IX
SPIN OUTPUT: VERIFICATION OF "PLATFORM VERIFICATION" PROPERTY

pan: ltl formula C1
(Spin Version 6.4.6 – 2 December 2016)
+ Partial Order Reduction
Full statespace search for:
never claim + (C1)
assertion violations + (if within scope of claim)
acceptance cycles + (fairness disabled)
invalid end states - (disabled by never claim)
State-vector 184 byte, depth reached 32, errors: 1
16 states, stored
0 states, matched
16 transitions (= stored+matched)
2 atomic steps
hash conflicts: 0 (resolved)

Our model is safe w.r.t "transitive access" because the BC process successfully detects and denies the four conflicting execution sequences discussed in section IV(A).

2) Safety Property for Execution Sequence 1: C2: {⋄ p → !q)}
#define p (a[0].aa[2] ≡ 2)
#define q (Bobstatus ≡ 1)
This property describes the "execution sequence 1" given in section IV(A). The verification result of the SPIN model checker and "never claim" of the above property is shown in **Table.X** and **Fig.14** respectively. Suppose Bob has accessed company α_1 with "w" and Alice has accessed company α_2 dataset with "r" i.e., (a[0].aa[0] ≡ 1) and (a[1].aa[1] ≡ 2). Then, Alice accessed company β_1 with "r". Therefore, Bob is not allowed to access company β_2 because it can cause leakage of information.

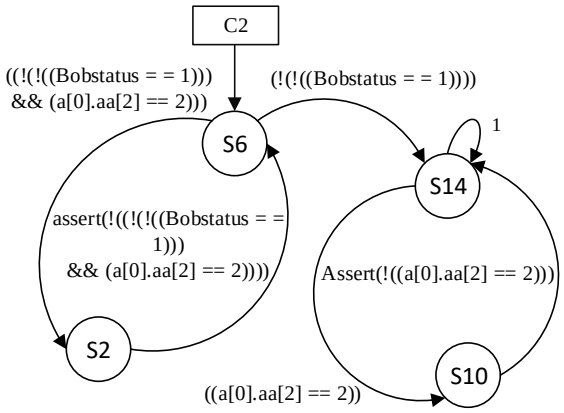


Fig. 14. Automata view of Safety Property for "Execution Sequence 1"

TABLE X
SPIN OUTPUT: VERIFICATION OF SAFETY PROPERTY FOR "EXECUTION SEQUENCE 1"

pan: ltl formula C2
(Spin Version 6.4.6 – 2 December 2016)
+ Partial Order Reduction
Full statespace search for:
never claim + (C2)
assertion violations + (if within scope of claim)
acceptance cycles + (fairness disabled)
invalid end states - (disabled by never claim)
State-vector 156 byte, depth reached 17, errors: 0
60 states, stored
36 states, matched
96 transitions (= stored+matched)
0 atomic steps
hash conflicts: 0 (resolved)

3) Safety Property of Execution Sequence 2: C3: {⋄ (p && q && r && t)}
#define p (a[0].aa[0] ≡ 2)
#define q (a[1].aa[1] ≡ 1)
#define r (Alicestatus ≡ 1)
#define t (Bob@denial_state)
This property describes the "execution sequence 2" given in section IV(A). The verification result of the SPIN model checker and "never claim" of the above property is shown in **Table.XI** and **Fig.15** respectively.

The SPIN model checker result shows that Bob request is denied. It means that only Alice can access the resource

because both requests are transitive.

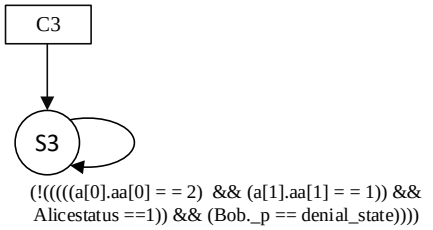


Fig. 15. Automata view of Safety Property for "Execution Sequence 2"

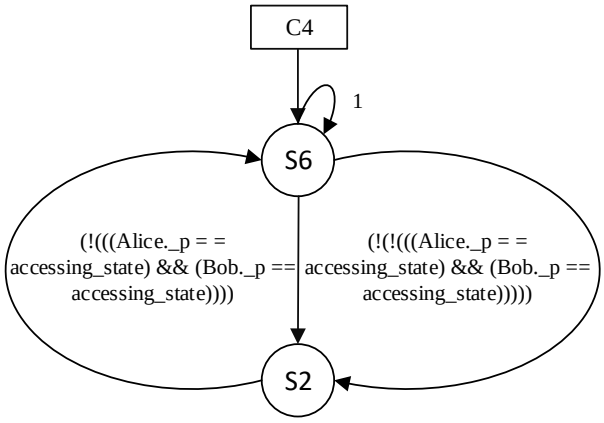


Fig. 16. Automata view of Safety Property for "Execution Sequence 3"

TABLE XI
SPIN OUTPUT: VERIFICATION OF SAFETY PROPERTY FOR "EXECUTION SEQUENCE 2"

```
pan: ltl formula C3
(Spin Version 6.4.6 – 2 December 2016)
+ Partial Order Reduction
Full statespace search for:
    never claim          + (C3)
    assertion violations + (if within scope of claim)
    acceptance cycles    + (fairness disabled)
    invalid end states   - (disabled by never claim)
State-vector 156 byte, depth reached 29, errors: 1
  30 states, stored
  1 states, matched
  31 transitions (= stored+matched)
  0 atomic steps
hash conflicts: 0 (resolved)
```

TABLE XII
SPIN OUTPUT: VERIFICATION OF SAFETY PROPERTY FOR "EXECUTION SEQUENCE 3"

```
pan: ltl formula C4
(Spin Version 6.4.6 – 2 December 2016)
+ Partial Order Reduction
Full statespace search for:
    never claim          + (C4)
    assertion violations + (if within scope of claim)
    acceptance cycles    + (fairness disabled)
    invalid end states   - (disabled by never claim)
State-vector 156 byte, depth reached 14, errors: 1
  15 states, stored
  0 states, matched
  15 transitions (= stored+matched)
  0 atomic steps
hash conflicts: 0 (resolved)
```

4) Safety Property of Execution Sequence 3: c4: [](p && q)
#define p (Alice@accessing_state)
#define q (Bob@accessing_state)
In access control, mutual exclusion means that a permission r cannot be assigned to users Alice and Bob at the same time, i.e., {(Alice, Bob), r)}. In our proposed scenario both the user requests are in TAS. Alice has accessed company β_1 with "w" and Bob has accessed company β_1 dataset with "r" i.e., (a[0].aa[2] $\equiv$ 2) and (a[1].aa[2] $\equiv$ 1). Therefore, both "Alice" and "Bob" cannot access the object simultaneously. The verification result of the SPIN model checker and 'never claim" of the above property is shown in **Table.XII** and **Fig.16** respectively.

5) Safety Property of Execution Sequence 4: C5: [](p $\rightarrow$ q&& !r)
#define p (a[1].aa[0] $\equiv$ 1)
#define q (Alice@accessing_state)
#define r (Bob@accessing_state)
Suppose Alice has accessed company β_1 with "w" and Bob has accessed company β_1 dataset with "r" i.e., (a[0].aa[2] $\equiv$ 1) and (a[1].aa[2] $\equiv$ 2). Then, Alice accessed company α_1 with "r". Therefore, Bob is not allowed to access company α_2 because it can cause leakage of information. The verification result of the SPIN model checker and 'never claim" of the above property is shown in **Table.XIII** and **Fig.17** respectively.

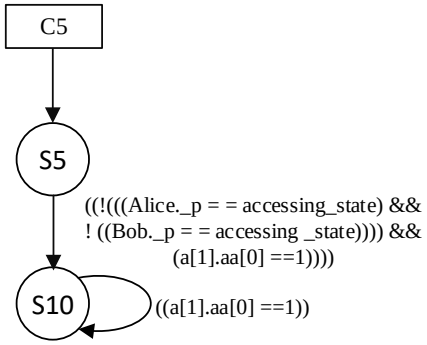


Fig. 17. Automata view of Safety Property for "Execution Sequence 4"

TABLE XIII
SPIN OUTPUT: VERIFICATION OF SAFETY PROPERTY FOR "EXECUTION SEQUENCE 4"

```
pan: ltl formula C5
(Spin Version 6.4.6 – 2 December 2016)
+ Partial Order Reduction
Full statespace search for:
        never claim             + (C5)
        assertion violations    + (if within scope of claim)
        acceptance cycles       + (fairness disabled)
        invalid end states      - (disabled by never claim)
State-vector 156 byte, depth reached 17, errors: 0
  60 states, stored
  36 states, matched
  96 transitions (= stored+matched)
  0 atomic steps
hash conflicts: 0 (resolved)
```

VI. CONCLUSION

In this paper, we have presented a BC based access control architecture for conflict of interest domains. Our proposed TACE mechanism (**Algorithm.1**) makes an authorization decision based on BC validation. PAS maintains access histories of the users whereas, TAS stores possible future "transitive accesses". Both PAS and TAS are stored on BC. We have integrated BC technology in BCON to make access control unbiased, decentralized and verifiable yet anonymous. We have proposed two transaction validation algorithms i.e., **Algorithm.2** and **Algorithm.3** to validate and update users access histories and "transitive accesses".

Furthermore, we have modeled our mechanism in PROMELA. A SPIN model checker is used to analyze our mechanism for "transitive access" enforcement. We have defined LTL properties for all possible conflicting sequences of execution. Then, the properties are checked against our promela model. The SPIN model checker results show that our promela model is secure against "transitive access". Currently, we are working to develop a smart contract for BCON architecture in hyperledger fabric. In the future, we planned to work on formal modeling and formal verification of BC.

ACKNOWLEDGMENT

The authors are thankful to Higher Education Commission (HEC) Pakistan for research funding under "National Center for Cyber Security" initiative for the project "Provable Security of Blockchain Technologies".

REFERENCES

[1] S. Subashini and V. Kavitha, "A survey on security issues in service delivery models of cloud computing," *Journal of network and computer applications*, vol. 34, no. 1, pp. 1–11, 2011.

[2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[3] Q. Alam, M. Alam, G. Ali, F. Azim, K. H. Khan, T. Ali, M. Ali, and A. Hayat, "Towards a formal framework for cross domain access control," *International Information Institute (Tokyo). Information*, vol. 15, no. 10, p. 4303, 2012.

[4] M. Alam, X. Zhang, K. Khan, and G. Ali, "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, pp. 31–40.

[5] S. Cirani, M. Picone, P. Gonizzi, L. Veltri, and G. Ferrari, "Iot-oas: An oauth-based authorization service architecture for secure services in iot scenarios," *IEEE sensors journal*, vol. 15, no. 2, pp. 1224–1234, 2014.

[6] G. J. Holzmann, "The model checker spin," *IEEE Transactions on software engineering*, vol. 23, no. 5, pp. 279–295, 1997.

[7] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper, "Simple on-the-fly automatic verification of linear temporal logic," in *Protocol Specification, Testing and Verification XV*. Springer, 1995, pp. 3–18.

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

[9] C. Yang, X. Chen, and Y. Xiang, "Blockchain-based publicly verifiable data deletion scheme for cloud storage," *Journal of Network and Computer Applications*, vol. 103, pp. 185–193, 2018.

[10] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. ACM, 2016, pp. 3–16.

[11] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake," *self-published paper, August*, vol. 19, 2012.

[12] M. Conti, E. S. Kumar, C. Lal, and S. Ruj, "A survey on security and privacy issues of bitcoin," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3416–3452, 2018.

[13] V. Buterin, "On public and private blockchains," *Ethereum blog*, vol. 7, 2015.

[14] O. Novo, "Blockchain meets iot: an architecture for scalable access management in iot," *IEEE Internet of Things Journal*, 2018.

[15] D. D. F. Maesa, P. Mori, and L. Ricci, "Blockchain based access control," in *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer, 2017, pp. 206–220.

[16] A. Ouaddah, A. A. Elkalam, and A. A. Ouahman, "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, pp. 523–533.

[17] A. Dorri, S. S. Kanhere, and R. Jurdak, "Blockchain in internet of things: challenges and solutions," *arXiv preprint arXiv:1608.05187*, 2016.

[18] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, "Blockchain for iot security and privacy: The case study of a smart home," in *Pervasive Computing and Communications Workshops (PerCom Workshops), 2017 IEEE International Conference on*. IEEE, 2017, pp. 618–623.

[19] R. Xu, Y. Chen, E. Blasch, and G. Chen, "Blendcac: A blockchain-enabled decentralized capability-based access control for iots," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2018, pp. 1027–1034.

[20] B. Tang, H. Kang, J. Fan, Q. Li, and R. Sandhu, "Iot passport: A blockchain-based trust framework for collaborative internet-of-things," in *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies*. ACM, 2019, pp. 83–92.

[21] S. Gusmeroli, S. Piccione, and D. Rotondi, "Iot access control issues: a capability based approach," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*. IEEE, 2012, pp. 787–792.

[22] M. Loock and J. H. Eloff, "A new access control model based on the chinese wall security policy model." in *ISSA*, 2005, pp. 1–10.

[23] D. F. Brewer and M. J. Nash, "The chinese wall security policy," in *Security and privacy, 1989. proceedings., 1989 ieee symposium on*. IEEE, 1989, pp. 206–214.

[24] M. T. Hammi, B. Hammi, P. Bellot, and A. Serhrouchni, "Bubbles of trust: A decentralized blockchain-based authentication system for iot," *Computers & Security*, vol. 78, pp. 126–142, 2018.

[25] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-ng: A scalable blockchain protocol," in *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, 2016, pp. 45–59.

[26] K. R. Özyılmaz and A. Yurdakul, "Designing a blockchain-based iot infrastructure with ethereum, swarm and lora," *arXiv preprint arXiv:1809.07655*, 2018.

[27] G. Holzmann, *Spin model checker, the: primer and reference manual*. Addison-Wesley Professional, 2003.

[28] J. R. Büchi, "Weak second-order arithmetic and finite automata," *Mathematical Logic Quarterly*, vol. 6, no. 1-6, pp. 66–92, 1960.

***Author Biography**

1.

GAUHAR ALI (gauharstd@uop.edu.pk) 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, block-chain, intelligent transport system, formal verification, and model checking.

2.

NAVEED AHMAD (n.ahmad@uop.edu.pk) 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.

3.

YUE CAO received the Ph.D. degree from the Institute for Communication Systems (ICS), 5G Innovation Centre, University of Surrey, Guildford, U.K., in 2013. He was a Research Fellow at ICS until 2016, He was a Lecturer with the Department of Computer and Information Sciences, Northumbria University, Newcastle upon Tyne, U.K., until 2017, where he has been a Senior Lecturer since 2017. His research interests include intelligent mobility. He is an Associate Editor of the IEEE ACCESS and KSII Transactions on Internet and Information Systems.

4.

QAZI EJAZ ALI (qaziejazali@uop.edu.pk , Phone No.: +92-91-9216732) 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.

5.

FAZAL AZIM (azimfazal@gmail.com) 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 access control, block-chain, and intelligent transport system.

6.

HAITHAM CRUICKSHANK (h.cruickshank@surrey.ac.uk) 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 is currently working in several FP6 projects such as SATLIFE, EuroNGI, and SATNEX. He also teaches 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. In addition, he is a member of the Satellite and Space Communications Committee of the IEEE ComSoc. He is active in the ETSI BSM (Broadband Satellite Multimedia) and the IETF MSEC groups.

In addition, He is Vice Chair of the COST 272 activity, which is part of the European COST research programme.

***Conflict of Interest**

AUTHOR DECLARATION

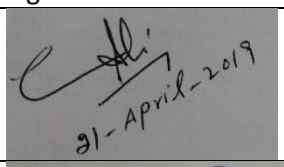
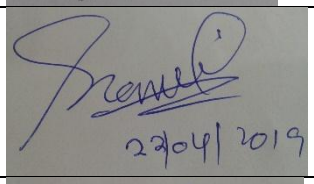
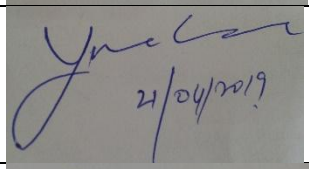
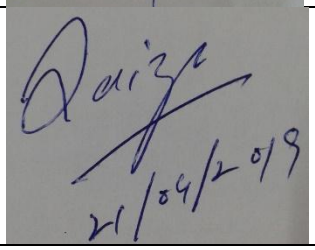
We wish to draw the attention of the Editor to the following facts which may be considered as potential conflicts of interest and to significant financial contributions to this work. [OR] We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

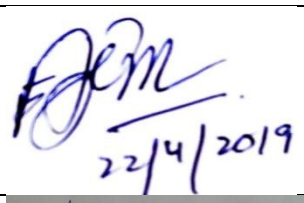
We confirm that the manuscript has been read and approved by all named authors and that there are no other persons who satisfied the criteria for authorship but are not listed. We further confirm that the order of authors listed in the manuscript has been approved by all of us.

We confirm that we have given due consideration to the protection of intellectual property associated with this work and that there are no impediments to publication, including the timing of publication, with respect to intellectual property. In so doing we confirm that we have followed the regulations of our institutions concerning intellectual property.

We understand that the Corresponding Author is the sole contact for the Editorial process (including Editorial Manager and direct communications with the office). He/she is responsible for communicating with the other authors about progress, submissions of revisions and final approval of proofs. We confirm that we have provided a current, correct email address which is accessible by the Corresponding Author and which has been configured to accept email from yue.cao@lancaster.ac.uk.

Signed by all authors as follows:

Author Name	Signature and date
Gauhar Ali	
Naveed Ahmed	
Yue Cao	
Qazi Ejaz Ali	

Fazal Azim	
Haitham Cruickshank