Received: 19 September 2018 | Revised: 15 July 2019 | Accepted: 15 November 2019
DOI: 10.1111/exsy.12507

ORIGINAL ARTICLE

Expert Systems WILEY

BlockU: Extended usage control in and for Blockchain

Yasar Khan[1] | Toqeer Ali[2] | Megat Fariz[1] | Fernando Moreira[3] | Frederico Branco[4,5] | José Martins[4] | Ramiro Gonçalves[4]

[1]Malaysian Institute of Information Technology, University Kuala Lumpur, Kuala Lumpur, Malaysia

[2]Faculty of Computer and Information System, Islamic University of Madinah, Madinah, Saudi Arabia

[3]IJP, REMIT, Universidade Portucalense & IEETA, Universidade de Aveiro, Aveiro, Portugal

[4]CSIG, INESC TEC and University of Trás-os-Montes e Alto Douro, Vila Real, Portugal

[5]Departamento de Informática e Matemáica, Instituto Politécnico de Bragança, Bragança, Portugal

Correspondence
Frederico Branco, INESC TEC and University of Trás-os-Montes e Alto Douro, Quinta de Prados, 5001-801 Vila Real, Portugal.
Email: fbranco@utad.pt

Abstract

An electronic business transaction among untrusted bodies without consulting a mutually trusted party has remained widely accepted problem. Blockchain resolves this problem by introducing peer-to-peer network with a consensus algorithm and trusted ledger. Blockchain originally introduced for cryptocurrency that came with proof-of-work consensus algorithm. Due to some performance issues, scientists brought concept of permissioned Blockchain. Hyperledger Fabric is a permissioned Blockchain targeting business-oriented problems for industry. It is designed for efficient transaction execution over Blockchain with pluggable consensus model; however, there is limitation of rapid application development. Hyperledger introduced a new layer called Hyperledger Composer on top of the Fabric layer, which provides an abstract layer to model the business application readily and quickly. Composer provides a smart contract to extend the functionality and flexibility of Fabric layer and provides a way of communication with other systems to meet business requirements. Hyperledger Composer uses role-based access control (RBAC) model to secure access to its valuable assets. However, RBAC is not enough because many business deals require continuous assets monitoring. Our proposed model, BlockU, covers all possible access control models required by a business. BlockU can monitor assets continuously during transactions and updates attributes accordingly. Moreover, we incorporate hooks in Hyperledger Composer to implement extended permission model that provides extensive permission management capability on an asset. Subsequently, our proposed enhanced access control model is implemented with a minimal change to existing Composer code base and is backward compatible with the current security mechanism.

KEYWORDS

fabric, Hyperledger Composer, permissioned Blockchain, UCON

1 | INTRODUCTION

Before Blockchain, performing a trusted transaction in an untrusted network was either not possible or it was only possible using complex methods such as remote attestation (Ismail, Syed, & Musa, 2014; Jaeger, Sailer, & Shankar, 2006). With the advent of Blockchain, it became very simple to perform a secure transaction between two unknown and untrusted systems. Blockchain was initially introduced in 2008 by Satoshi Nakamoto for Bitcoin transactions. Bitcoin provided a distributed ledger with no trusted or central party, offering secure Bitcoin transactions among peer-to-peer parties. The order of transactions is validated using a consensus algorithm, that is, a proof-of-work algorithm (Nakamoto,

2008). Blockchain has been useful in meeting the needs of today's businesses, and it has the potential to radically change future business applications and models. According to a recent report by Gartner (2018), Blockchain's value may reach $3.1 trillion by 2030.

Each transaction among two untrusted parties involves a trusted party, such as a bank or a certification authority. To perform a transaction between unknown parties, many blockchain-based frameworks have been proposed (Abdeen, Ali, Khan, & Yagoub, n.d.; Ali, Ali, Alsaawy, Khalid, & Musa, 2019; Crosby, Pattanayak, Verma, & Kalyanaraman, 2016; Khan, Zuhairi, Ali, Alghamdi, & Marmolejo-Saucedo, 2019; Kwon, 2014; Wood, 2014) and are categorized as public and private blockchain frameworks. One of them is the Hyperledger project supported by IBM and other renown industries, for example, Intel, Cisco, and SAP. Hyperledger Fabric provides a decentralized operating system for next-generation business applications (Androulaki et al., 2018; Cachin, 2016). It overcomes performance issues that were a rather big concern in public blockchain systems. Hyperledger Fabric implements smart contracts called chaincodes among various distributed nodes; they contain the details of a contract, for example, the selling policy of a car, and are disclosed to all nodes that represent different stakeholders (e.g., buyer, seller, and manufacturer). These nodes or peers are called endorsers and are defined in the endorsement policy of the chaincode (Androulaki et al., 2018). Once the contract is finalized, only the transaction is stored on the distributed ledger. This process is conducted by the Orderer node in Hyperledger Fabric. The Orderer node maintains the order of transactions, for example, which transaction comes first and then second and so on in the ledger. The Orderer node in Hyperledger Fabric uses practical Byzantine fault tolerance as a consensus algorithm instead of a proof-of-work algorithm, which is used in Bitcoin (Castro & Liskov, 1999). Hyperledger Fabric provides various APIs for application developers to design and develop their own applications based on their business networks (Cachin, 2016). However, writing applications in Hyperledger Fabric is to an extent different due to the provided APIs. In addition, although Hyperledger Fabric uses role-based access control (RBAC) security model, it has been shown in the past that RBAC cannot perform attribute updates or the fined-grained usage control of assets (Park & Sandhu, 2004). RBAC is a very popular security model that has shaped the security domain for decades. This model is mainly used in operating systems like Windows, Linux, and Solaris and databases like Oracle. There are many permissioned blockchains like Ethereum, Hyperledger Fabric, Swatooh, Iroha, and many more. However, to the best of our knowledge, all these frameworks are using traditional access control models such discretionary access control, mandatory access control model, and RBAC. We have opted for Hyperledger Fabric, introduced by IBM in 2017 (Dhillon, Metcalf, & Hooper, 2017), for this research because it is open source and is widely used in different business applications. Hyperledger Composer provides application development APIs very easily and rapidly. Hyperledger Composer also implements smart contracts based on clients, assets, users, and access control lists. In this paper, we proposed a detailed usage control model that is more effective than the existing RBAC model that is currently implemented in Hyperledger Composer. Our newly proposed access control model has the capability to design a variety of business scenarios.

In this research, we formally constructed a usage control model for Hyperledger Composer to clarify examples based on the model. In addition, we described a detailed permission model that enables the members of a business network and the owner of assets to specify usage-based constraints while performing transactions. The new model is tailored specifically to Hyperledger Composer. The model has the capability to specify various types of policies depending on business needs, for example, allowing a specific asset to be accessed only for a specific time, with a specific computational power, or revoking an asset access from a user based on the occurrence of some defined condition. Finally, we implemented an extended usage control model on the smart contract implementation of Hyperledger Composer, which uses a blockchain to record every transaction on the distributed ledger.

1.1 | Use case

In a blockchain business network, many business deals rely on resource/asset usage control. Businesses share resources instead of buying them, and although based on the usage of assets, we can grant and revoke access to resources; this is called access decision continuity. Renting/accessing a resource for a specific time is common, such as when renting a car or releasing health records. In the example of renting a car, multiple parties are involved including the owner of the car, the renter of the car, the bank, and vehicle-tracking entities. The assets involved in this transaction are the car and the money. The renting authority rents the car for a specific time frame and zone the user can drive it in. The renter can order a car online, and the car will be reserved for them. The renter pays money to rent the car, and the asset's ownership is exchanged for a specific duration. For example, a car may be rented for 24 hr, and the driver can only drive the car in New York City. The vehicle-tracking entities track the car closely using defined GPS coordinates. If the car exits the approved zone, alerts are sent to the owner and the renter. After a few warnings, if the car is still not in the approved range, the ownership of the car may be revoked. The car will be locked or tracked further by government authorities. Similarly, if the car is not returned in the agreed amount of time, the owner can send alerts to the renter. The owner can also request the bank to deduct money from the renter's account. In this case, the bank can check the policy defined between the two parties and transfer money from the renter's account to the owner's account. If the renter account does not have enough money to comply, the bank can send a message to the vehicle-tracking system to lock the car unless the owner responds and pays the money. Once the money is paid, access can be granted again. This business model cannot be implemented in Hyperledger Composer, as an extended usage control model is needed.

The rest of the paper is organized as follows: Section 2 elaborates the background of blockchain studies, and Section 3 explains the UCON model in detail, Section 4 presents the proposed model, whereas Section 5 explains its implementation, and Section 6 concludes the paper, whereas Section 7 presents future directions of the studies.

2 | BLOCKCHAIN BACKGROUND

In this section, brief background studies are provided related to Blockchain, Hyperledger, and Hyperledger Fabric. Moreover, certain access control types that involve usage access control are also elaborated.

2.1 | Blockchain

Blockchain is a distributed ledger that was initially introduced for Bitcoin transactions. Each block has a block number that is generated using mining techniques, the details of the transaction, the previous block, and the newly generated block hash. Using a previous hash to generate a new block creates a chain of blocks known as a blockchain. There are two types of blockchains, that is, public and private blockchains. Using a blockchain, two untrusted networks can easily perform a transaction in a secure manner, as a blockchain can maintain a distributed ledger and record every transaction on the ledger nodes. A distributed ledger uses mining or a consensus protocol to solve double spending problems and malicious transactions. Once a block is created, any small change in the transaction or other details of the block are detected and further accepted by the rest of the nodes in the network.

2.2 | Hyperledger

Hyperledger is a permissioned blockchain technology with various private blockchain projects. The one we opted for in these studies is Hyperledger Fabric. Hyperledger Fabric is a pluggable and generic smart contract framework that is implemented on top of blockchain to design any kind of application, including cryptocurrency applications (though it was not developed specifically for cryptocurrency applications). Apart from Hyperledger Fabric, Hyperledger also offers Hyperledger Burrow, Hyperledger Sawtooth, Hyperledger Indy, and Hyperledger Iroha for various purposes. The following section further elaborates on the components of Hyperledger Fabric.

2.3 | Hyperledger Fabric

Initially, blockchain was designed as a public ledger, which means that every transaction performed by the blockchain network should be visible to all nodes. However, cases exist in which two or more parties do not wish to disclose their transaction details but would still like to benefit from blockchain technology. Therefore, IBM started a generic private blockchain implementation known as Hyperledger Fabric. In a private network, a new participant cannot be added unless it receives an invitation to become part of the business network.

2.3.1 | Execute-order-validate architecture

Hyperledger Fabric uses an execute-order-validate architecture (Androulaki et al., 2018), whereas traditional blockchains use an order-execute architecture. In an order-execute architecture, once a transaction is generated, it requires mining to generate a hash. In addition, it is generated by peers or minors. Once one of the peers can generate an acceptable hash, it disseminates it to the rest of other peer nodes. Further, every peer on the local system validates the block based on the value of its previous block hash. This order-execute architecture suffers from a few problems, such as sequential execution, nondeterministic code, confidential execution, fixed trust, and hard-coded consensus. To overcome these shortcomings, Hyperledger Fabric offers a permissioned, blockchain-backed smart contract that uses an execute-order-validate architecture with a chaincode, an enforcement policy, and a Hyperledger Fabric network.

In Hyperledger Fabric, a chaincode implements the smart contract. The chaincode runs in the execution phase of Hyperledger Fabric's architecture. Using the application, one can design their own logic in any programming language and perform a transaction with any other application in the network. Once a transaction is performed, it is stored on a distributed ledger. The endorsement policy can specify an endorser for a specific transaction.

Hyperledger Fabric's use of an enforcement policy is a novel concept in permissioned-based smart contracts. In a public smart contract, there is no privacy, meaning that all transactions are visible to everyone on the network. In addition, a transaction details need to be specified exchanged between two parties to successfully execute a transaction. In contrast, smart contracts have endorsers that only enable the transaction to be visible to select parties. However, once the endorsers agree on certain parameters, a final transaction is propagated and validated based on the consensus algorithm.

Every blockchain-based system involves a network of nodes with various roles, including Hyperledger Fabric. However, the nodes in the network have identities in permissioned blockchain-based systems, including client nodes, peer nodes, and ordering service nodes. Client nodes initiate transactions and send them further to the execution phase. Peer nodes perform certain tasks like maintaining the ledger and validating the transactions between endorsing nodes. Of these nodes, some endorsing nodes respond to the transaction proposals requested by the client. Ordering service nodes first complete the transactions of clients via the chaincode, and then they complete the transactions of the endorsers. Ordering service nodes handle the consensus and maintain the order by which transactions are updated on the ledger.

3 | UCON FOR HYPERLEDGER COMPOSER

The first and most popular model for usage control is UCON, which was proposed by Park and Sandhu (2004) and was later formalized by Zhang et al. (2005). The following discussion is taken mostly from this formalization. This model is based on two core concepts: decision continuity and attribute mutability.

Decision continuity refers to the concept that a decision to grant access is not a singular operation but rather a continuous action that is carried out in parallel to subjects' access. Once access has been granted, UCON's reference monitor tracks the usage and can perform several useful tasks based on it. For this purpose, the second aspect of UCON, attribute mutability, is of paramount value. Attribute mutability enables changes to a subject or object's attributes because of accessing or trying to access an object. UCON allows policy writers to define certain types of attribute mutability (Figure 1). UCON includes the following states: initial, requesting, denied, accessing, revoked, and end. The model puts each (s, o, r) triple in one of these states, where s is the subject, o is the object, and r is the right. Consider the following:

- t1 = (s1, o1, r1) state(t1) = accessing.

Subject s1 is currently exercising right r1 on object o1; the reference monitor tracks this usage as soon as s1 tries to access o1. This is the nonbypassability feature of the reference monitor, which ensures that access is mediated by the security policy enforced by the reference monitor. Immediately after trying to access o1, the state of the triple is changed to *requesting*. The reference monitor can perform one of the following actions at this stage:

- permitAccess: Allow s1 to access o1; in this case, the state changes to *accessing*.
- denyAccess: Disallow access, which changes the state to *denied*.

One of these two decisions must be made by the reference monitor, and they are, of course, mutually exclusive. The reference monitor can also perform the following action:

- preUpdate: Update the attributes of the subject or the object. This is a pre-update operation and is highly useful for keeping track of how many times a subject has tried to access an object. Hence, repeated attempts by a subject to access a restricted object can be detected by the reference monitor, after which the subject would be penalized for misbehaviour.

If the subject is denied access, the session finishes, and the triple must return to the *initial* state before performing any restricted action. However, if the state changes to accessing, the subject is free to access the object. At any point in time during access, the reference monitor might revoke the granted permission. The following four operations can potentially occur:

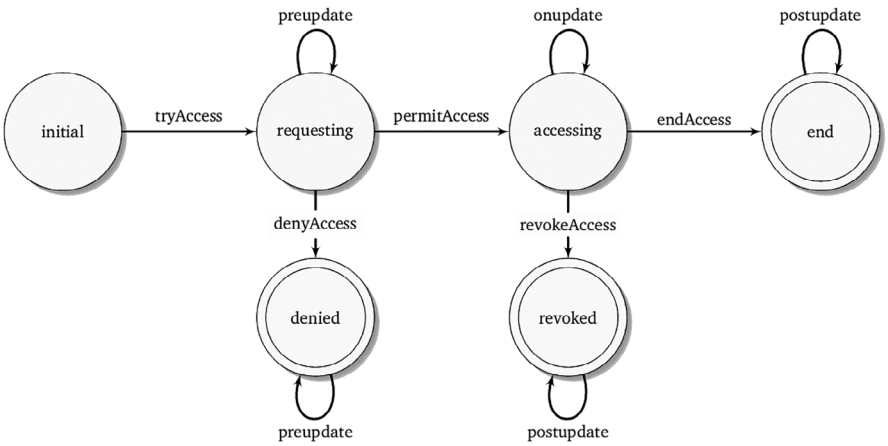

FIGURE 1 UCON model states

- endAccess: The subject voluntarily gives up the object. In this case, the reference monitor does not have to perform any task.
- revokeAccess: Because of the usage or due to changes in the operating environment, the reference monitor revokes access. In this case, the subject is no longer able to access the object, and the state of the system changes to *revoked*.
- onUpdates: During access, the reference monitor might update some attributes to keep track of the usage. This is a clean method for gauging usage.
- postUpdates: The reference monitor can perform updates to a subject or object's attributes based on whether the subject voluntarily gave up the object or if it was taken back forcefully.

In the following section, we formally define UCON's access and usage control.

4 | BLOCKU: EXTENSION OF USAGE CONTROL

Usage control must not be limited to granting access to a right only once; instead, it should constantly keep track of the usage of objects and decide continuously whether to allow a subject access to objects or not. If a subject exceeds the allowed quota or if another constraint is violated during the use of an object, the subject's right should be revoked. This simple but powerful idea enables various policy specifications and thus extends the expressiveness of access control systems immensely. With this new capability, it is possible for an organization to specify a policy that dictates that an object can be accessed only within the premises of its offices and within a certain time frame. It also allows for the specification of constraints, such as allowed usage time per session and the revocation of usage after a specified limit.

4.1 | Model definition

Hyperledger Composer's architecture is based on assets, participants, transactions, and conditions. According to our proposed usage control for Hyperledger Composer, we defined the models based on the new requirements. These models are extensions to Hyperledger Composer and can aid in the development of business applications based on blockchains.

Definition 1: *Participants and assets. We denote the set of participants in an active business network with P and the set of all assets within a transaction with A. Assets in a specific transaction can be accessed by the participant association function (ζ: A $\rightarrow$ P). Each asset is associated with a unique participant.*

Definition 2: *Permissions. We denote the set of all permissions with S. Permissions are the collection of labels that allow access to an asset. A permission requires P_s: A $\rightarrow$ A to provide access s $\in$ S, which p $\in$ P requires the user to have.*

Definition 3: *Access association with assets. The access association function (ψ: P $\rightarrow$ 2^s) returns the set of permissions that belong or were given to participant p $\in$ P at runtime.*

4.2 | Extension to the core Composer model

To define extended usage control permissions over assets, we introduce the concept of the participant state as well as updates to the operation, condition, and policies. The participant state decides which set of attributes belongs to the participant, including arbitrary characteristics about participants that are related to their permissions.

Definition 4: *Participant state t: γ (P) $\rightarrow$ com (γ(P)) is a function that maps a participant's attributes to their values. This is a dictionary of attributes that is associated with each participant. γ is a function that provides the set of attributes that belongs to a participant, and the set of all possible values for an attribute (a) is denoted with com(a). These attributes can be updated via the attribute update function.*

4.2.1 | Pre-authorization models

Before a participant accesses an asset in our Hyperledger Composer extension, we added a usage decision function.

Definition 5: *According to the above additional usage functions, our model has the following components:*

- P, A, S, ATT(S), ATT(A), befA (participants, assets, permissions, participant attributes, asset attributes, and asset attributes before authorization), and
- granted (p,a,s) = befA((ATT(p)), ATT(a), S).

Similarly, an additional step before an update is sometimes required, which is defined in our model as follows:

Definition 6: *befUpdate(ATT(P)) and befUpdate(ATT(a)) are optional operations to update the attributes of participants and assets, respectively.*

4.2.2 | Ongoing-authorization model

In the ongoing-authorization model, the composer requests asset usage without any pre-decision making. However, the authorization of assets is monitored constantly. In this model, each participant is initially allowed to use an asset; however, this model does remove their access if certain requirements are not met. In Hyperledger Composer, such a model is very useful when assets are accessed for a long duration of time. This functionality can be further extended into four different models:

- the A0 model (no pre-updates)
- the A1 model (with pre-updates)
- the A2 model (with continuous updates)
- the A3 model (with post-updates)

Definition 7: *In the A0 model, P, A, S, ATT(S), and ATT(A) remain the same as in the pre-authorization model.*

- onA (continuous authorization)
- allowed (p,a,s) $\Rightarrow$ true
- stopped (p,a,s) $\Leftarrow \neg$ conA(ATT(p),ATT(a),S)

Definition 8: *The A1 model is like the A0 model except it updates using the following pre-update processes: preUpdate(ATT(p)) and onUpdate(ATT(a)), which are optional procedures to perform update operations on ATT(p) and ATT(a), respectively.*

Definition 9: *The A2 model is also similar to the A0 model except it adds the following continuous-update processes: onUpdate(ATT(p)) and onUpdate(ATT(a)), which are optional procedures to perform update operations on ATT(p) and ATT(a), respectively.*

Definition 10: *The A3 model adds the following post-update processes to the A0 model: postUpdate(ATT(s)) and postUpdate(ATT(o)), which are optional procedures to perform update operations on ATT(s) and ATT(o), respectively.*

4.2.3 | Pre-obligation models

Hyperledger Composer introduced a new obligation feature in its architecture. Using this feature allows a participant to access an asset if and only if they provide information during the authentication process. Hyperledger Composer verifies this information using participants. However, the existing architecture of Hyperledger Composer only supports first-time authentication. Our model can check this information continuously.

Definition 11: *The pre-obligation model has the following components:*

- P,A,S,ATT(P) and ATT(A) are the same as in the pre-authorization model
- OBS, OBO, and OB (obligation subjects, obligation objects, and obligation actions, respectively)
- preB and preOBL (pre-obligation predicates and pre-obligation elements, respectively)
- preOBL $\subseteq$ OBS $\times$ OBO $\times$ OB
- preFulfilled: OBS $\times$ OBO $\times$ OB $\rightarrow$ {true, false}
- getPreOBL: P $\times$ A $\times$ S $\rightarrow$ 2preOBL, a function to select pre-obligations for a requested usage
- preB(P,A,S) = $\wedge$ (obsi, oboi, obi) $\in$ getPreOBL(P,A,S) preFulfilled(obsi, oboi, obi)
- preB(P,A,S) = true by definition if getPreOBL(P,A,S) = φ
- allowed(P,A,S) $\Rightarrow$ preB(P,A,S).

Definition 12: *The preB1 model is identical to the befB0 model except it adds the following pre-update processes: preUpdate(ATT(P)) and preUpdate(ATT(a)), which are optional procedures to change certain attributes because of pre-obligations.*

Definition 13: *The preB3 model is identical to the befB0 model except it adds the following post-update processes:*

- postUpdate(ATT(p)) and postUpdate(ATT(a)), which are optional procedures to change certain attributes because of pre-obligations.

4.2.4 | Continuous-obligation models

The continuous-obligation model is like the pre-obligation model except it requires the verification of obligations periodically or constantly. To implement this model, we introduce a time parameter (T), which depends on a time- or element-based period; for example, we can ask a participant to submit information every hour or to provide information after a specific amount of resource usage. Based on mutability issues, the continuous-obligation model can implement the following four models: the B0 model, which includes an ongoing-obligation predicate instead of a pre-obligation predicate; the B1 model; the B2 model; and the B3 model. The B1, B2, and B3 models are the same as the B0 model except that they add pre-updates, ongoing updates, and post-updates, respectively.

Definition 14: *The B0 model has the following components:*

- P,A,S, ATT(S), ATT(A), OBS, OBO, and OB are the same as in the befB model
- T, which represents a set of time or event elements
- onB and onOBL, which represent ongoing obligation predicates and ongoing obligation elements, respectively
- onOBL $\subseteq$ OBS $\times$ OBO $\times$ OB $\times$ T
- getOnOBL: P,A,S $\rightarrow$ 2OnBL, which is a function to select ongoing obligations for a requested usage
- onFulfilled: OBS $\times$ OBO $\times$ OB $\times$ T $\rightarrow$ true, false
- onB(P,A,S) = $\wedge$(obsi, oboi, obi,ti) $\in$ getOnOBL (P,A,S) onFulfilled(obsi, oboi, obi, ti)
- onB (P,A,S) = true by definition if getOnOBL (P,A,S) = θ
- allowed(P,A,S) $\Rightarrow$ true
- stopped(P,A,S) $\Leftarrow$ $\neg$onB(P,A,S).

Definition 15: *The B1 model is identical to the B0 model except it adds the following pre-update processes: preUpdate(ATT(p)) and preUpdate(ATT(a)), which are optional procedures to change certain attributes as a result of pre-obligations.*

Definition 16: *The B2 model is identical to the B0 model except it adds the following ongoing update processes: onUpdate(ATT(p)) and onUpdate(ATT (a)), which are optional procedures to change certain attributes as a result of pre-obligations.*

Definition 17: *The B3 model is identical to the B0 model except it adds the following post-update processes: postUpdate(ATT(p)) and postUpdate(ATT (a)), which are optional procedures to change certain attributes as a result of pre-obligations.*

4.2.5 | Pre-condition model

Some conditions define environmental variables through which we can enforce certain policies; for example, employees of an organization can use its resources only when they are in the premises of the organization. Such restrictions are not directly associated with any assets in Hyperledger Composer. In the pre-condition model, such conditions are checked each time the assets are accessed. The pre-condition model introduces the pre-condition predicate (preC), which must be evaluated before the requested rights are exercised. The following definitions formalize the preC model.

Definition 18: *The preC model has the following components:*

- P,A,S, ATT(S), and ATT(O) are the same as in the preA model
- preCON (a set of pre-condition elements)
- getPreCON: $P \times A \times S \rightarrow 2$ preCON
- preConChecked: preCON $\rightarrow$ true, false
- preC(P,A,S) = $\wedge$preConi$\in$getPreCON(P,A,S)preConChecked(preConi)
- allowed(P,A,S) $\Rightarrow$ preC(P,A,S).

4.2.6 | Ongoing-conditions model

In many cases, environmental restrictions must be satisfied while rights are in active use. This can be supported using the onC0 model. In the onC0 model, usage is allowed without any decision process at the time of the request. However, there is an ongoing condition predicate to check certain environmental statuses repeatedly throughout usage. As mentioned earlier, the onC0 model is intrinsically immutable. The following definitions formalize the onC0 model.

Definition 19: *The onC0 model has the following components:*

- P,A,S, ATT(S), and ATT(A) are the same as in the preA model
- onCON (a set of ongoing conditions)
- getOnCON: $P \times A \times S \rightarrow 2$ onCON; onConChecked: onCON $\rightarrow$ true, false
- onC(P,A,S) = onConi$\in$getOnCON(P,A,S) onConChecked(onCon i)
- allowed(P,A,S) $\Rightarrow$ true
- stopped(P,A,S) $\Leftarrow \neg$ onC(P,A,S).

Herein, we extended the core Hyperledger Composer model with the above additional models. This enriches future applications with a variety of permissions based on business needs.

5 | UCON INCORPORATION IN HYPERLEDGER FABRIC

Hyperledger Composer is a high-level layer in Hyperledger Fabric's architecture. Hyperledger Composer uses a generic chaincode to communicate with Hyperledger Fabric. Hyperledger Composer provides some extra layers and features that are not directly part of Hyperledger Fabric. To implement our proposed model, we extended Hyperledger Composer with extra features and introduced new transaction calls to be saved on the blockchain's trusted ledger and state database. To achieve this, we added hooks in the current JavaScript transaction processor, connecting us to the extra models we defined earlier. Similarly, we introduced automatic blockchain transactions that are stored on the blockchain. Whenever an asset or participant is added to the system using Hyperledger Composer's API, our proposed model automatically creates different transactions that are stored on the blockchain and state database. The purpose of these transactions is to monitor the activity of the participants while they are accessing and using any asset. Our model automatically responds to the global configuration that is set to monitor asset usage.

To incorporate the extended version of the access control, we updated three sections of the Composer as depicted in Figure 2.

- Access control: In ACL module, we introduced new tags that support our UCON policies. These tags are declared in the development phase of any BNA.
- Composer runtime: The second module that affected our proposed solution is the Composer runtime. Composer runtime is basically connected with Fabric, which is the backbone of the blockchain infrastructure. The newly defined access control models are linked with the Composer runtime, so it can be hooked to the application execution.
- JavaScript transactions processor: In the third module of UCON incorporation, we created a BlockU reference monitor that continuously monitors the assets and records all transactions in the blockchain against a participant. An asset can have multiple attributes, and we can define separate transactions for each of the attributes; similarly, we can create various BlockU rule for each attribute (for instance, a vehicle asset that contains attributes, such as millage, price, manufacture, and current state). Our newly introduced functions can retrieve an attribute—for example, millage—via the transaction asset vehicle millage. There are two types of restrictions on the usage of an asset. One constraint is developed in the business logic of the application; for example, a person cannot buy an asset if his payment is less than the product amount; such kind of constraints are also handled in the chaincode transaction functions.

```
rule SampleConditionalRule {
description: "Description of the ACL rule"
participant(m): "org.example.SampleParticipant"
operation: ALL //CREATE, UPDATE, DELETE or ALL
resource(v): "org.example.SampleAsset"
condition: (v.owner.getIdentifier() == m.getIdentifier())
action: ALLOW // ALLOW, DENY
// BlockU Tags
blockU: onOBL //befA, onA, befOBL, onOBL, preCON, onCON
blockUcondition: (v.blockUReferenceMonitor.usage(m.getIdentifier(), DATETIME) < 10)
}
```

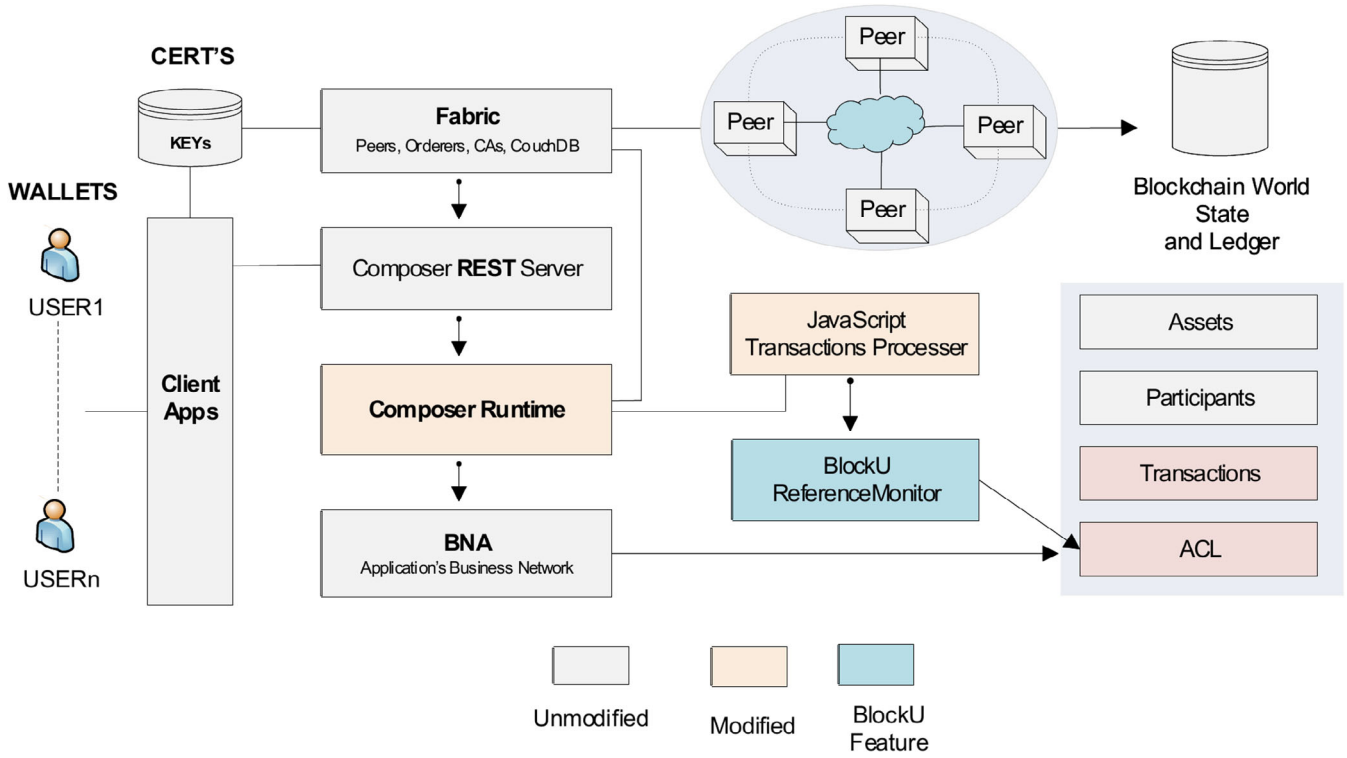

FIGURE 2 Proposed usage control framework for permissioned Blockchain

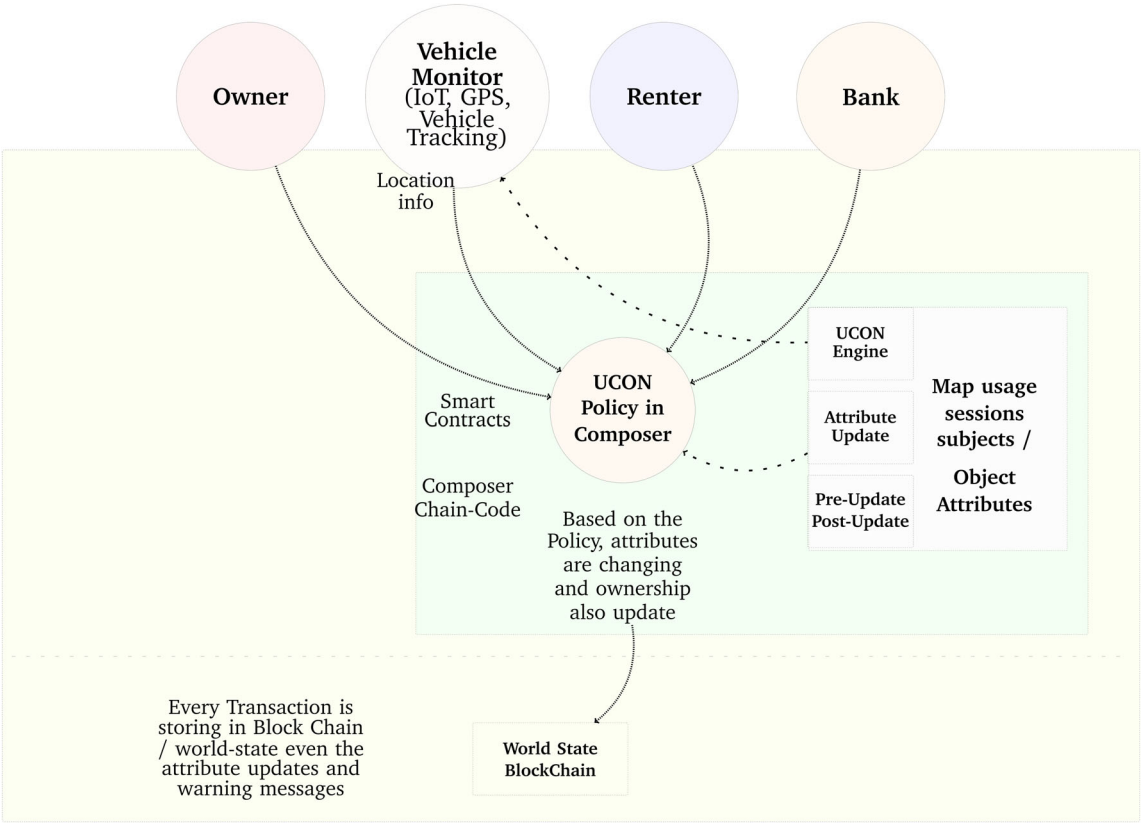

FIGURE 3 Extended usage control model for Blockchain

Our proposed model verifies each request against the generated policy before allowing or denying participants to use the assets (Figure 3). The model adds this failure request as a new transaction on the blockchain that can be seen by a different network and the system administrator who can monitor and respond to participants as and when needed. It also helps to resolve business disputes. This detailed permission model enables members of a business network and owners of assets to specify usage-based constraints while performing transactions. The new model is tailored to Hyperledger Composer. As mentioned earlier, attribute mutability includes three states in the UCON engine, that is, the first "pre-update," the second "upon update," and the third "post-update." Pre-update attributes are set before accessing an asset. In our use case, we set the start time to zero initially, ensuring that after 24 hr, the asset would be revoked. Once an asset is moved to the accessing state, the mutable attribute is updated accordingly. We also set the warnings to zero to ensure that once the number of warnings reaches the allotted time, the access would be revoked. Each state change of UCON is stored in a transparent ledger, in case of a dispute or if the bank would like to deduct money from the renter. Our second condition involves constantly monitoring the defined zone. In the case of the car rental, we set the current coordinates of the car as a pre-update attribute to constantly monitor the car's location. In addition, we set the warning attribute. In case the car travels out of range, the user receives five warnings. If the user does not respond to these warnings, access to the car is revoked, and all transactions are locked.

6 | CONCLUSION

In the recent past, permissioned Blockchain got acceptance due to the various issues in public Blockchain, such as performance, security, and other business requirements. Among these Blockchain frameworks, the Hyperledger project gained popularity due to its modular, open source architecture and huge industry support. The research done can be adopted by any permissioned Blockchain framework; however, the proof of concept is demonstrated in Hyperledger Composer. The study contributes in extending the permissions model of Hyperledger Composer by introducing BlockU. BlockU covers various business applications that need continuous access to their assets by a participant. Many business use cases cannot be implemented using the RBAC model. Usage control models are difficult to implement; thus, extending permissions are necessary to meet the needs of current and future business applications. The proposed model allows developers and end users to define their own access policies for their shared assets rather than using fixed policies developed and instantiated at the start of the application. They can also define asset's

access rules violation policy and set environmental constraints to the usage of their assets. Such violation policies can trigger consequent responses, which helps organizations with real-time monitoring and tracking the participant's activities.

7 | FUTURE WORK

To identify future work, further investigation can be made to improve the policy execution engine and to deploy it on the underlying Fabric architecture. Similarly, we aim to develop a new blockchain history track for our proposed model policies, which will be separated from the application level chaincode. A new system can be introduced, that is, system chaincode in the fabric architecture; it will have its own private data section where the proposed policies shall be stored, and its core functionality will be the execution engine logic. Similarly, a high-level language interface is required for the user to write their policies, which ultimately can be transformed to the core Hyperledger Composer engine.

CONFLICT OF INTERESTS

None.

ORCID

Toqeer Ali https://orcid.org/0000-0002-5731-6650
Frederico Branco https://orcid.org/0000-0001-8434-4887
José Martins https://orcid.org/0000-0002-7787-6305
Ramiro Gonçalves https://orcid.org/0000-0001-8698-866X

REFERENCES

Abdeen, M. A. R., Ali, T., Khan, Y., & Yagoub, M. C. E. (n.d.). Fusing identity management, HL7 and Blockchain into a global healthcare record sharing architecture.

Ali, J., Ali, T., Alsaawy, Y., Khalid, A. S., & Musa, S. (2019). Blockchain-based smart-IoT trust zone measurement architecture. *Proceedings of the International Conference on Omni-Layer Intelligent Systems*, 152–157.

Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., … Muralidharan, S. (2018). Hyperledger fabric: A distributed operating system for permissioned blockchains. *Proceedings of the Thirteenth EuroSys Conference*, 30.

Cachin, C. (2016). Architecture of the hyperledger blockchain fabric. *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, *310*, 4.

Castro, M., & Liskov, B. (1999). Practical Byzantine fault tolerance. *OSDI*, *99*, 173–186.

Crosby, M., Pattanayak, P., Verma, S., & Kalyanaraman, V. (2016). Blockchain technology: Beyond bitcoin. *Applied Innovation*, *2*(6–10), 71.

Dhillon, V., Metcalf, D., & Hooper, M. (2017). The hyperledger project. In *Blockchain enabled applications* (pp. 139–149). Berkeley, CA: Springer.

Gartner. (2018). Gartner says global artificial intelligence business value to reach $1.2 trillion in 2018. Retrieved from https://www.gartner.com/en/newsroom/press-releases/2018-04-25-gartner-says-global-artificial-intelligence-business-value-to-reach-1-point-2-trillion-in-2018

Ismail, R., Syed, T. A., & Musa, S. (2014). *Design and implementation of an efficient framework for behaviour attestation using n-call slides*. Paper presented at the Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication, p. 36.

Jaeger, T., Sailer, R., & Shankar, U. (2006). *PRIMA: Policy-reduced integrity measurement architecture*. Paper presented at the Proceedings of the Eleventh ACM Symposium on Access Control Models and Technologies, pp. 19–28.

Khan, M. Y., Zuhairi, M. F., Ali, T., Alghamdi, T., & Marmolejo-Saucedo, J. A. (2019). An extended access control model for permissioned blockchain frameworks. *Wireless Networks*, *25*, 1–12.

Kwon, J. (2014). Tendermint: Consensus without mining. *Draft v. 0.6, Fall*, *1*, 11.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

Park, J., & Sandhu, R. (2004). The UCON ABC usage control model. *ACM Transactions on Information and System Security (TISSEC)*, *7*(1), 128–174.

Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, *151*(2014), 1–32.

Zhang, X., Parisi-Presicce, F., Sandhu, R., & Park, J. (2005). Formal model and policy specification of usage control. *ACM Transactions on Information and System Security (TISSEC)*, *8*(4), 351–387.

AUTHOR BIOGRAPHIES

Yasar Khan received an M.S. degree in Computer Sciences from the Institute of Management Sciences, Peshawar, Pakistan, in 2016. After completion of his Master Degree in Computer Sciences in 2008, he joined Security Engineering Research Group, Institute of Management Sciences, Peshawar in 2009 as Android System and Architecture Developer,where he worked on many areas of Security Engineering such as Integrity Measurement Architecture, Android Permission Extension, Usage Control Engine and on Cross-Domain Access Control Management. He is currently doing his Ph.D. in Information Technology at University of Kuala Lumpur, Malaysia. His current research interests include Blockchain, IoT, malware analysis, machine learning and Cross Domain Access Control Management.

Toqeer Ali Syed is currently working as an Associate Professor in Islamic University of Madinah, KSA. He also served for two years as a Senior Lecture at University Kuala Lumpur. He is a professional, teacher and a researcher working in the field of System Security, Deep Learning, Blockchain Applications and Cloud Computing. He has been actively involved in research and teaching activities since last 10 years. He received a PhD degree in Engineering Technology (IT) from University Kuala Lumpur in 2014. He completed his Masters in Computer Sciences. He has published his research findings in several forums of international repute.

Megat F. Zuhairi is an Associate Professor at University Kuala Lumpur, Malaysian Institute of Information Technology where he has been since 2004. He currently serves as Deputy Dean Student Development and Campus Lifestyle. He was an Engineer at Marconi (M) Bhd., where he worked 2 years after he completed his BEng (Hons) in Electrical & Electronics from UNITEN. Generally, he teaches computer network subjects but every other semesters he also teaches electrical and electronics. He was a Cisco Certified Network Associate (CCNA) between 2007 and 2009 and now serves as Instructor for various courses for Cisco Network Academy. His research interests lie in the area of computer data networking, and wireless mobile ad hoc communications. Over the span of 4 years he has published more than 25 journals and conference publications including 2 books. In recent years, he has focused on wireless sensor network and data acquisition and analysis within the context of Internet of Things. He has served on roughly 15 conference programme committees as reviewer and technical members.

Fernando Moreira is the Director of the Science and Technology Department. He is graduated in Computer Science (1992), M.Sc. in Electronic Engineering (1997) and PhD in Electronic Engineering (2003), both at Faculty of Engineering of the University of Porto, and Habilitation (2018). He is a member of the Science and Technology Department at Portucalense University since 1992, currently as Full Professor and a visiting professor at the University of Porto Business School. He teaches subjects related to undergraduate and post-graduate studies. He supervises several PhD and M.Sc. students. He is a (co-)author more than 150 scientific publications with peer-review on national and international journals and conferences. He serves as a member of the Editorial Advisory Board for several journals and books. He organized several special issues from JCR journals. He has already regularly served as a member of Programme and Scientific committees of national and international conferences. He was the MSc in Computation coordinator during the last ten years. He holds editorial experience, and he is co-editor of several books. He is associated with NSTICC, ACM and IEEE. His principal research areas are mobile computing, ICT in Higher Education, Mobile learning, Social business and Digital transformation. He was awarded Atlas Elsevier Award, April 2019.

Frederico Branco is Assistant Professor at the University of Trás-os-Montes and Alto Douro and Senior Research of INESC TEC research center. He has published over 70 articles in journals and event proceedings. He is also involved in several academic works, as dissertation and thesis supervisor and degree projects responsible, and is continuously participating in several research projects. His professional career is also directly related with the industry, with particular focus in various planning and implementation projects of Information Systems, with particular attention on agri-food and services sectors. Currently holds several functions of senior management in the areas of Operations,Information Systems and Quality Management.

José Martins is currently an Invited Assistant Professor at the University of Trás-os-Montes e Alto Douro (Department of Engineering), Invited Assistant at the Polytechnic Institute of Bragança and Senior Researcher at INESC TEC research center. He has published over 90 articles in indexed journals and event proceedings focusing the Information Systems, Management Information Systems, Software Engineering and Human-Computer Interaction topics. Currently he is supervisor for several Master Degree dissertations and PhD thesis. During his research career, he has participated in several research projects and is currently a member of various research projects aimed at merging information systems and technologies with other fields of study. Throughout his professional career José has also worked as an information systems and technologies senior consultant where he directly participated in several international projects. At the present time José Martins dedicates most of his time to his lectures and to his research activities where he tries to understand the variables and (in)direct impacts of ICT adoption at individual and firm levels, and how IS solutions can be idealized, specified and developed in order to fully address their audience needs and requirements.

Ramiro Gonçalves is an Associate Professor with Habilitation at University of Trás-os-Montes e Alto Douro, in Vila Real, Portugal, and a Senior Researcher at INESC TEC Associated Laboratory, Porto, Portugal. Ramiro is currently the Executive Director of the PhD degree in Informatics and has around 200 publications (including book chapters, Scientific Citation Index journal articles, as well as publications in refereed conference proceedings).

How to cite this article: Khan Y, Ali T, Fariz M, et al. BlockU: Extended usage control in and for Blockchain. *Expert Systems*. 2020;1–12. https://doi.org/10.1111/exsy.12507