On the Design of Blockchain-Based Access Control Scheme for Software Defined Networks

Durbadal Chattaraj*, Sourav Saha†, Basudeb Bera‡, and Ashok Kumar Das§, *Senior Member, IEEE*
* Subir Chowdhury School of Quality & Reliability, Indian Institute of Technology, Kharagpur 721 302, India
dchattaraj@iitkgp.ac.in
†‡§ Center for Security, Theory and Algorithmic Research,
International Institute of Information Technology, Hyderabad 500 032, India
{sourav.saha,basudeb.bera}@research.iiit.ac.in, iitkgp.akdas@gmail.com

***Abstract*—Software Defined Networking (SDN) becomes a *de facto* standard for the future Internet. SDN decouples the control plane from the data plane of a proprietary network asset to ensure better programmability and security for designing more innovative future network applications. Presently, the SDN framework does not have proper access control mechanism among different entities, namely SDN applications, SDN controllers and switches. To achieve this goal, this paper proposes a blockchain-based access control scheme for the SDN framework. The proposed scheme has the capability to resist various well-known attacks and alleviate the existing single point of controller failure issue in SDN.**

***Index Terms*—software-defined networking, blockchain, consensus, security, access control.**

I. Introduction

With the profound adaptation of emerging "Information and Communication Technology (ICT)" in various domains, such as cloud, social, mobile, and big data demands proper management of a large-scale dynamic network [1]. Towards this solution, Software-Defined Networking (SDN) plays an important role [2]. SDN decouples the traditional control plane from the data plane of a proprietary network asset to ensure better scalability, efficiency, reconfigurability, security and dynamic management of existing static network infrastructure. However, adopting SDN over static networks raises several security threats [2], [3], [4].

Since the inception of SDN, most of the security threats appear to be in the two specific planes in SDN framework, namely application and control plane [3]. Due to the absence of proper authentication and access control mechanisms in application plane, an application can easily inject a malicious configuration into switches (i.e., network assets) through a controller node, which can indirectly lead to network breakdown phenomenon. As a result, this scenario reduces the overall dependability, in particular, reliability and availability of the entire network [3]. Moreover, improper accountability and detectability of different application flows through single/multiple controller(s) may cause loss of information about the network failure traces. In spite of this, improper identification of different attack patterns considering a robust debugging of the underlying network is also another aspect which is not properly analyzed in current SDN environment [3].

In the control plane, a single point of compromise (or failure) of an SDN controller makes the whole network jeopardized. Also, such kind of provision makes the whole network vulnerable to Denial-of-Service (DoS) attack. Furthermore, proper access control mechanisms among SDN applications and controllers are not found in the existing literature [3]. Eventually, in the controller-switch communication, due to the absence of a secure channel (or a robust access control mechanism) among the controllers and switches, in particular, the non-adaptation of "configuration-complex" Transport Layer Security (TLS) protocol makes the communication channel vulnerable to various attacks (for example, man-in-the-middle and eavesdropping attacks) [3]. In addition, an illegitimate switch can induce spoofing attacks, DoS and distributed DoS (DDoS) attacks by masquerading the publicly known identities of other switches [3].

To avert the above said security issues, several studies have been carried out in the recent literature [3], [5], [6], [7], [8], [9]. In addition, blockchain-based security solutions in SDN-enabled networks have also gained popularity in recent years [10], [11]. However, to the best of our knowledge, there is no concrete foundation of blockchain-based access control scheme among different components involved into the SDN framework.

This work aims to address the following objectives:

- An access control scheme is needed to overcome DoS, DDoS, man-in-the-middle (MITM), spoofing, and replay attacks of individual planes in SDN framework.
- A robust approach is required to be introduced to address the single point of SDN controller failure issue through the use of blockchain technology.

The rest of the paper is oriented as follows. Section II briefs about the proposed network and threat models. Section III presents the proposed blockchain based access control scheme. Section IV discusses security analysis and also presents performance analysis of the proposed scheme. Finally, Section V concludes the paper.

II. System Models

This section elaborates network and threat models that are utilized in discussing and analyzing the proposed scheme.

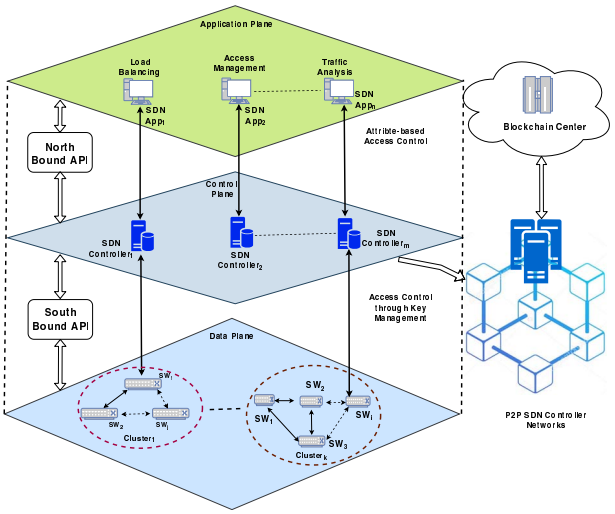

Fig. 1. System architecture of proposed protocol (adapted from [2])

A. Network Model

A blockchain-based network model for an SDN platform provided in Fig. 1 has been adapted based on [2] in the design of the proposed scheme. According to the proposed network model, the following entities exist:

Registration Authority (RA): The RA is responsible to enroll each principal of three different planes of the SDN framework.

SDN Application (SA): The SA is a software application running on the top layer of an SDN framework to manage the dynamicity of a physical network infrastructure.

SDN Controller (SC): The SC is responsible for controlling the flow of an SDN application (SA) as well as supervising the intelligent data forwarding through different propitiatory network assets, called SDN switches (SSW)s, that are the core components preferably belonging to a data plane as a network cluster. In addition, each SC can collect the real-time data from both SAs and SSWs, and construct blocks containing the transactions. The constructed blocks are then forwarded to several other controllers in the P2P controller network.

SDN Switch (SSW): The SSW is a propitiatory network asset preferably used for data forwarding from a source node to a destination node (or host). A collection of switches configured with a particular topology makes different network clusters (CLs). Each CL is controlled by a particular SC in SDN.

Blockchain Center (BC): The BC will act as a universal ledger and it stores the blocks into several cloud servers for future use.

In this model, the switches are grouped together into several disjoint clusters. Furthermore, under each cluster CL_k, for a set of switches there exists a single controller that will manage the data forwarding from one host to another host(s) (intended destinations).

TABLE I
NOTATIONS AND THEIR MEANINGS

Notation	Description
App_{id}, Con_{id}, Swi_{id}	Real identities of SDN application (SA), controller (SC) and switch (SSW)
App_{type}	Type of SDN application (SA) (i.e., traffic management, access management and security control)
$Cluster$, $Cluster_{id}$	A set of network assets (combination of controller and switches), real identity of cluster
$Cert_{app}$, $Cert_{con}$, $Cert_{swi}$	Certificates of SA, SC and SSW generated by the RA (Registration Authority)
$ECDSA.Sig(s_K : [M])$	Elliptic Curve Digital Signature Algorithm (ECDSA) signature generation on a message M with private key s_K of an entity X
$ECDSA.Ver(Pub_K : [M])$	ECDSA verification algorithm on message M with public key Pub_K of an entity X
T_x^{id}	Transaction identity created for principal x
$Flow_{id}$	Real identity of an SDN application flow
$state$	Current state of the whole network
$content$	Flow content (specifically, configuration instruction) with respect to $Flow_{id}$
$Event_{id}$	Real identity of a network event
$E_q(u, v)$	A "non-singular elliptic curve" with parameters u, v over finite (Galois) field $GF(q)$; q being a prime
G	A base point in $E_q(u, v)$ of order n
$P + Q$, $k.P$	Elliptic curve point addition and multiplication, respectively, $P, Q \in E_q(u, v)$ and $k \in Z_q^* = \{1, 2, \cdots, q-1\}$
$x * y$	Ordinary modular multiplication in Z_q
ΔT	"Maximum transmission delay associated with message"
$h(\cdot)$	"Collision-resistant cryptographic one-way hash function"

B. Threat Model

The broadly-accepted "Dolev-Yao (DY) threat model" [12] has been applied that permits an adversary $\mathcal{A}$ not only can intercept the communicated messages in between communication, but can also modify and delete the message contents, and even can inject malicious contents inside the SDN framework (specifically, between the application and control plane, and also between data plane and control plane). In addition, under this model, the end-point communicating entities (in our case, application (SA) and switch (SSW)) are not contemplated as trustworthy entities in the network. In addition, it may not be always possible to develop an SDN application (SA) without bugs. Hence, there is a possibility to inject malicious codes into the same SA. Therefore, using this SA, an adversary $\mathcal{A}$ may attempt to mount different known attacks (for example, DoS and spoofing attacks) by suppressing the actual activities of a controller (SC) or a switch (SSW). Furthermore, it is assumed that the SA and SSW are not trusted entities, whereas the SDN controller (SC) in the P2P controller network and the BC are trusted.

III. Proposed Blockchain-based Access Control Mechanism in SDN

This section provides a detailed description of the proposed blockchain-based access control scheme in SDN framework, called BACC-SDN. We list the notations along with their significance in Table I that are essential to describe the proposed BACC-SDN. The following phases are involved in BACC-SDN.

The *system initialization phase* of BACC-SDN provides the selection of all the related system parameters, such as selection of a non-singular elliptic curve and its base point, one-way cryptographic hash function, consensus algorithm for blockchain, private and public keys of the trusted RA and other entities involved in the SDN. In BACC-SDN, through the *registration phase*, the RA registers each component (application, controller, and switch) of three different planes of SDN framework as shown in Fig. 1. During the registration process, each entity is provided with necessary pre-loaded information prior to their deployment and they apply those pre-loaded information for mutual (node) authentication and key establishment of secret keys with the help of the *access control phase*.

The *access control phase* of BACC-SDN consists of two tasks. In the first task, an SDN application needs to establish a secure communication with the controller node followed by a mutual authentication process between themselves. An attribute-based access control mechanism is put forward to accomplish this task. In the second task, a key management-based access control mechanism is proposed to form a secure channel between the controller node and switch. After establishment of a secure communication between the application and controller (or switch and controller), a secure blockchain-based data access control mechanism is discussed in the next phase.

The *block addition in Blockchain Center phase* deals with the transaction and block (a set of transactions) formation policy. After collecting the data (i.e., all the interaction messages availed from the SDN applications and switches to controller) securely by the controllers, it proceeds for creating various transactions, and then it will form a block containing the encrypted transactions. The generated block is passed to one of the other controllers in the P2P Controller Network (CN), which will be responsible for mining that block into a blockchain maintained by a Blockchain Center (BC) after applying the well-known consensus algorithm, namely "Practical Byzantine Fault Tolerance (PBFT)" [13].

A. System Initialization Phase

In this phase, the RA first chooses a "non singular elliptic curve $E_q(u,v)$ of the form: $y^2 = x^3 + ux + v \pmod q$ over a finite field GF(q) with a point of infinity $\mathcal{O}$, where $(u,v) \in Z_q = \{0,1,2,\cdots,q-1\}$ and $4u^3+27v^2 \neq 0 \pmod q$". The RA then picks a base point $G \in E_q(u,v)$ of order n as large as q, where $n.G = G+ G+ \cdots G$ (n times). Next, the RA selects a "one way cryptographic hash function" $h(\cdot)$ (for instance, SHA-256 hash function [14] can be considered for sufficient security in blockchain technology).

B. Registration Phase

The RA registers all the principals, such as SA, SC, SSW and BC prior to their functionality. It selects unique identities Con_{id}, App_{id} and Swi_{id} for each SC, SA and SSW, respectively. Next, it picks its own private key $s_{RA} \in Z_q^*$ and calculates corresponding public key $Pub_{RA} = s_{RA} \cdot G$. Similarly, the RA also picks private keys s_{con}, s_{app} and $s_{swi} \in Z_q^*$, and computes their respective public keys as $Pub_{con} = s_{con} \cdot G$, $Pub_{app} = s_{app} \cdot G$ and $Pub_{swi} = s_{swi} \cdot G$ for each SC, SA and SSW, respectively.

After that the RA generates certificates $Cert_{con} = s_{con} + h(Con_{id} \,||Cluster_{id}||\, Pub_{con}||\, Pub_{RA}) * s_{RA} \pmod q$ for each SC, $Cert_{app} = s_{app} + h(App_{id} \,||App_{type}||\, Pub_{app}||\, Pub_{RA}) * s_{RA} \pmod q$ for each SA and $Cert_{swi} = s_{swi} + h(Swi_{id} \,||Cluster_{id}||\, Pub_{swi}||\, Pub_{RA}) * s_{RA} \pmod q$ for each SSW, respectively. The RA then selects a t-degree symmetric bivariate polynomial over GF(q) as $f_{con-swi}(x,y) = \sum_{i=0}^{t}\sum_{j=0}^{t} a_{ij}x^i y^j$, where $a_{ij} \in Z_q$, and calculates the polynomial shares $f_{con-swi}(Con_{id}, y)$ and $f_{con-swi}(Swi_{id}, y)$ for every SC and SSW, respectively. After that the RA assigns $\{Con_{id}, s_{con}, Cert_{con}, f_{con-swi}(Con_{id}, y)\}$, $\{App_{id}, s_{app}, Cert_{app}\}$, and $\{Swi_{id}, s_{swi}, Cert_{swi}, f_{con-swi}(Swi_{id}, y)\}$ to each SC, SA and SSW, respectively. The RA publishes all the public parameters that are available to all participants including an adversary. Finally, the RA deletes all the private keys of the principals SA, SC and SSW from its database.

C. Access Control between Application and Controller

In this section, we apply the Attribute-Based Encryption (ABE) [15] for secure communication between SDN application and its respective controller. The main reason for applying ABE here is that we only allow the respective SDN controller to retrieve its privileged application(s). An ABE consists of the four algorithms: a) Setup, b) Encrypt, c) KeyGen, and d) Decrypt, which are briefly described below [16]:

The **Setup** algorithm is supplied with a "security parameter λ" and the "universe of attributes A_i", say $\mathcal{U} = \{A_1, A_2, \cdots, A_n\}$ as inputs. It then produces a "master secret" and "public key" pair, say (MK, PK) as output.

The **Encrypt** algorithm $Enc(\mathcal{P}, Msg)$ is given an "access policy $\mathcal{P}$", a "plaintext message Msg" and the "public key PK" as inputs, and it produces the "ciphertext C as output".

The **KeyGen** algorithm is supplied with an "attribute set $\mathbb{A}$", the "master secret key MK" and "public key PK". It then produces the "user secret key (i.e., decryption key), say K_U" of $\mathbb{A}$ for the user U.

Finally, the algorithm **Decrypt** $Dec(C, \mathcal{P}, K_U, \mathbb{A})$" is supplied with a "ciphertext C generated with $\mathcal{P}$", the "secret key K_U" respective to $\mathbb{A}$ and the public key PK as inputs. It then produces the message Msg or $\perp$ (null) as output.

In this phase, the transactions communicated between the controller and switch are encrypted using the ABE **Encrypt** algorithm discussed above with their "access policy $\mathcal{P}$" and "public key PK". The receiver then decrypts the encrypted transactions using the **Decrypt** algorithm based on "secret key K_U" respective to $\mathbb{A}$ and the public key PK.

D. Access Control between Controller and Switch

The session key establishment process followed by a mutual authentication task between a SDN controller (SC_i) and a switch (SSW_j) inside a cluster, say CL_k, is summarized in Fig. 3. The step-wise illustration of the same task is elaborated

Controller SC_i	Switch SSW_j
Generate random number $r_1 \in Z_q^*$ Generate current timestamp TS_1 Compute $f_{con-swi}(Con_{id}, Swi_{id})$, $A_{con} = h(r_1 \|\| TS_1) \oplus h(f_{con-swi}(Con_{id}, Swi_{id}) \|\| TS_1)$, $B_{con} = h(Con_{id} \|\| Swi_{id} \|\| A_{con} \|\| TS_1 \|\| Cert_{con} \|\| f_{con-swi}(Con_{id}, Swi_{id}))$ $msg_1 = \{Con_{id}, Swi_{id}, Cert_{con}, A_{con}, B_{con}, TS_1\}$	
(via a public channel)	
	Check if $\|TS_1' - TS_1\| < \Delta T$? If so, compute $f_{con-swi}(Swi_{id}, Con_{id})$, $h(r_1 \|\| TS_1) = A_{con} \oplus h(f_{con-swi}(Swi_{id}, Con_{id}) \|\| TS_1)$, $B'_{con} = h(Con_{id} \|\| Swi_{id} \|\| A_{con} \|\| TS_1 \|\| Cert_{con} \|\| f_{con-swi}(Swi_{id}, Con_{id}))$ Verify if $B'_{con} = B_{con}$? If so, verify if $Cert_{con} \cdot G = Pub_{con} + h(Con_{id} \|\| Cluster_{id} \|\| Pub_{con} \|\| Pub_{RA}) \cdot Pub_{RA}$? Generate current timestamp TS_2 Generate random number $r_2 \in Z_q^*$ Compute $A_{swi} = h(r_2 \|\| TS_2) \oplus h(f_{con-swi}(Swi_{id}, Con_{id}) \|\| TS_2)$, session key $SK_{Swi,Con} = h(f_{con-swi}(Swi_{id}, Con_{id}) \|\| h(r_1 \|\| TS_1) \|\| h(r_2 \|\| TS_2) \|\| Cert_{con} \|\| Cert_{swi})$, session key verifier $SKV_{Swi,Con} = h(SK_{Swi,Con} \|\| A_{swi} \|\| Cert_{swi} \|\| Con_{id} \|\| Swi_{id} \|\| TS_2)$ $msg_2 = \{Con_{id}, Swi_{id}, Cert_{swi}, A_{swi}, SKV_{Swi,Con}, TS_2\}$
Check if $\|TS_2' - TS_2\| < \Delta T$? If so, verify if $Cert_{swi} \cdot G = Pub_{swi} + h(Swi_{id} \|\| Cluster_{id} \|\| Pub_{swi} \|\| Pub_{RA}) \cdot Pub_{RA}$? If so, compute $h(r_2 \|\| TS_2) = A_{swi} \oplus h(f_{con-swi}(Swi_{id}, Con_{id}) \|\| TS_2)$, session key $SK_{Con,Swi} = h(f_{con-swi}(Con_{id}, Swi_{id})$ $\|\| h(r_1 \|\| TS_1) \|\| h(r_2 \|\| TS_2) \|\| Cert_{con} \|\| Cert_{swi})$, session key verifier $SKV_{Con,Swi} = h(SK_{Con,Swi} \|\| A_{swi}$ $\|\| Cert_{swi} \|\| Con_{id} \|\| Swi_{id} \|\| TS_2)$ Verify if $SKV_{Con,Swi} = SKV_{Swi,Con}$?	
	(via a public channel)
Store shared secret session key $SK_{Con,Swi} (= SK_{Swi,Con})$ between controller SC_i and switch SSW_j	

Fig. 2. Summary of access control between controller SC_i and switch SSW_j

as follows. Note that we apply the polynomial-based key distribution approach by Blundo *et al.* [17] as part of the session key establishment procedure.

Step 1: SC_i generates a random number $r_1 \in Z_q^*$, picks current timestamp TS_1, and calculates $f_{con-swi}(Con_{id}, Swi_{id})$, $A_{con} = h(r_1 \|\| TS_1) \oplus h(f_{con-swi}(Con_{id}, Swi_{id}) \|\| TS_1)$ and $B_{con} = h(Con_{id} \|\| Swi_{id} \|\| A_{con} \|\| TS_1 \|\| Cert_{con} \|\| f_{con-swi}(Con_{id}, Swi_{id}))$, and sends the message $msg_1 = \{Con_{id}, Swi_{id}, Cert_{con}, A_{con}, B_{con}, TS_1\}$ to SSW_j via a public channel.

Step 2: After receiving msg_1 from SC_i at time TS_1', the switch SW_j verifies if $|TS_1' - TS_1| < \Delta T$ or not, where ΔT is the "maximum transmission delay". If the condition is true, SSW_j calculates $f_{con-swi}(Swi_{id}, Con_{id})$, $h(r_1 \|\| TS_1) = A_{con} \oplus h(f_{con-swi}(Swi_{id}, Con_{id}) \|\| TS_1)$ and $B'_{con} = h(Con_{id} \|\| Swi_{id} \|\| A_{con} \|\| TS_1 \|\| Cert_{con} \|\| f_{con-swi}(Swi_{id}, Con_{id}))$. Next, if the condition $B'_{con} = B_{con}$ fails, SW_j terminates the session.

Step 3: SSW_j then verifies the certificate of SC_i by the condition: $Cert_{con} \cdot G = Pub_{con} + h(Con_{id} \|\| Cluster_{id} \|\| Pub_{con} \|\| Pub_{RA}) \cdot Pub_{RA}$ by utilizing the known domain parameters (i.e., public information). If the certificate is verified successfully, SC_i is believed to be legitimate principal by SSW_j; else, it rejects SC_i's request. After that SSW_j generates a current timestamp TS_2 and a random number r_2, and calculates $A_{swi} = h(r_2 \|\| TS_2) \oplus h(f_{con-swi}(Swi_{id}, Con_{id}) \|\| TS_2)$, session key shared with SC_i as $SK_{Swi,Con} = h(f_{con-swi}(Swi_{id}, Con_{id}) \|\| h(r_1 \|\| TS_1) \|\| h(r_2 \|\| TS_2) \|\| Cert_{con} \|\| Cert_{swi})$ and session key verifier $SKV_{Swi,Con} = h(SK_{Swi,Con} \|\| A_{swi} \|\| Cert_{swi} \|\| Con_{id} \|\| Swi_{id} \|\| TS_2)$, and sends the message $msg_2 = \{Con_{id}, Swi_{id}, Cert_{swi}, A_{swi}, SKV_{Swi,Con}, TS_2\}$ to SC_i via a public channel.

Step 4: After getting the message msg_2 from SSW_j at time TS_2', SC_j verifies if $|TS_2' - TS_2| < \Delta T$ or not. If the condition is true, SC_i verifies the certificate $Cert_{swi}$ by the condition: $Cert_{swi} \cdot G = Pub_{swi} + h(Swi_{id} \|\| Cluster_{id} \|\| Pub_{swi} \|\| Pub_{RA}) \cdot Pub_{RA}$. If the certificate is valid, SC_i continues to calculate $h(r_2 \|\| TS_2) = A_{swi} \oplus h(f_{con-swi}(Swi_{id}, Con_{id}) \|\| TS_2)$, session key shared with SSW_j as $SK_{Con,Swi} = h(f_{con-swi}(Con_{id}, Swi_{id}) \|\| h(r_1 \|\| TS_1) \|\| h(r_2 \|\| TS_2) \|\| Cert_{con} \|\| Cert_{swi})$ and session key verifier $SKV_{Con,Swi} = h(SK_{Con,Swi} \|\| A_{swi} \|\| Cert_{swi} \|\| Con_{id} \|\| Swi_{id} \|\| TS_2)$. Finally, SC_j verifies if the condition: $SKV_{Con,Swi} = SKV_{Swi,Con}$ is valid or not. If the verification passes, both SC_i and SSW_j store the same session key $SK_{Con,Swi} (= SK_{Swi,Con})$ for their secret communications.

It is then clear that the mutual authentication and session key agreement process is accomplished between a controller (SC_i) and a switch (SSW_j). Thus, both SC_i and SSW_j establish a session key between themselves through a symmetric session key $SK_{Con,Swi} (= SK_{Swi,Con})$.

Transactions Details
(1) Transaction format of individual component (SA, SC, SSW) in SDN framework
• $T_{SA} \rightarrow \{T^{id}_{SA}, App_{id}, Pub_{app}, App_{type}, Cert_{app}, Sign_{app}\}$ • $T_{SC} \rightarrow \{T^{id}_{SC}, Con_{id}, Pub_{con}, Cluster_{id}, Cert_{con}, Sign_{con}\}$ • $T_{SSW} \rightarrow \{T^{id}_{SSW}, Swi_{id}, Pub_{swi}, Cluster_{id}, Cert_{swi}, Sign_{swi}\}$, where $Sign_{app} = ECDSA.Sig(s_{app} : [App_{id} \|\| App_{type}])$, $Sign_{con} = ECDSA.Sig(s_{con} : [Con_{id} \|\| Cluster_{id}])$, $Sign_{swi} = ECDSA.Sig(s_{swi} : [Swi_{id} \|\| Cluster_{id} \|\| Con_{id}])$
(2) Interaction between application (SA) and controller (SC)
• $T_{SA-SC} \rightarrow \{T^{id}_{SA-SC}, T^{id}_{SA}, T^{id}_{SC}\}$
(3) Interaction between controller (SC) and switch (SSW)
• $T_{SC-SSW} \rightarrow \{T^{id}_{SC-SSW}, T^{id}_{SC}, T^{id}_{SSW}\}$
(4) Application flow request to the controller (SC) before install (or uninstall) a new (or old) configuration to (or from) the switch (SSW) for a particular reason
• $T^{bf}_{appflw2con} \rightarrow \{T^{bf}_{appfl_{id}}, App_{id}, Flow_{id}, Con_{id}, Pub_{app}, content, Sign^{bf}_{appfl}\}$, where $Sign^{bf}_{appfl} = ECDSA.Sig(s_{app} : [App_{id} \|\| Flow_{id} \|\| Con_{id} \|\| content])$.
(5) Action taken by the controller (SC) with respect to the application flow request (as mentioned in previous transaction), and updation of network state
• $T^{af}_{appflw2con} \rightarrow \{T^{af}_{appfl_{id}}, Con_{id}, Flow_{id}, Swi_{id}, state, Sign^{af}_{appfl}\}$, where $Sign^{af}_{appfl} = ECDSA.Sig(s_{con} : [App_{id} \|\| Flow_{id} \|\| Con_{id} \|\| Swi_{id} \|\| state])$
(6) Current status of the network after execution of an application flow (considering both transactions $T^{bf}_{appflw2con}$ and $T^{af}_{appflw2con}$)
• $T_{appflow} \rightarrow \{T^{id}_{flow}, T^{bf}_{appfl_{id}}, T^{af}_{appfl_{id}}\}$
(7) A network event occur between controller and switch
• $T_{swi-event} \rightarrow \{T^{id}_{swi-event}, Event_{id}, Pub_{swi}, Swi_{id}, Con_{id}, event\}$
$T_{SA}, T_{SC}, T_{SSW}, T_{SA-SC}, T_{SC-SSW}, T^{bf}_{appflw2con}, T^{af}_{appflw2con}, T_{appflow}$, $T_{swi-event}$ represent different transactions constructed by the controller (SC)

Fig. 3. Formats of various transactions

E. Block Addition in Blockchain Center

In this section, we provide the details of creating a block by the controller node (SC) and then verifying that block by the P2P SC network. After successful consensus among the controller nodes in the P2P SC network, the block is added in the existing blockchain.

The formats of various transactions Tx_i are listed in Fig. 3. The structure of a block, say $Block_m$ created by a controller node SC containing various transactions is shown in Fig. 4. The signature of the block is created with the help of the elliptic curve digital signature algorithm (ECDSA) [18]. The block also contains the hash of the current block and Merkle tree root. The signature along with current hash and Merkle tree root help to achieve immutability and transparency property of the block maintained in the blockchain. Now, to verify the block $Block_m$, it requires selection of a leader, say SC_l among the available controller nodes in the P2P SC network as in [19]. After that SC_l will execute the PBFT consensus algorithm in order to verify and add that block in the blockchain maintained in the BC. This process is provided in Algorithm 1.

Block Header	
Block Version	BV
Previous Block Hash	$PBHash$
Merkle Tree Root	MTR
Timestamp	TS
Owner of Block	OB
Public key of signer (Con)	Pub_{con}
Block Payload (Encrypted Transactions)	
List of Encrypted Transactions $\#i$ (Tx_i)	$\{E_{Pub_{con}}(Tx_i)\|i = 1, 2, \cdots, t\}$
Current Block Hash	$CBHash$
Signature on $CBHash$	$BSign = ECDSA.Sig(s_{con:[CBHash]})$

Fig. 4. Formation of a block $Block_m$ by a P2P controller node

IV. Security and Performance Analysis

A. Security Analysis

In this section, we apply the threat model discussed in Section II-B:

1) Replay Attack: To protect replay attack, we have utilized both the random numbers along with current system timestamps. Therefore, the old replayed messages can be easily detected by the respective recipients by checking the timestamps attached in the received messages. Thus, the proposed BACC-SDN is resilient against "replay attack".

2) Spoofing Attack: In an SDN framework, the spoofing attack happens when a malicious switch (SSW_p) controlled by an adversary $\mathcal{A}$ impersonates another legitimate switch (SSW_q), and tries to bypass a legal access control task between SC_i and SSW_j [3]. Assume $\mathcal{A}$ controlling SSW_p first eavesdrops all the communication messages $msg_1 = \{Con_{id}, Swi_{id}, Cert_{con}, A_{con}, B_{con}, TS_1\}$ and $msg_2 = \{Con_{id}, Swi_{id}, Cert_{swi}, A_{swi}, SKV_{Swi,Con}, TS_2\}$. $\mathcal{A}$ may attempt to construct A_{swi} and $SKV_{Swi,Con}$ from the eavesdrop messages to impersonate SSW_j in the consecutive sessions. However, to construct valid A_{swi} and $SKV_{Swi,Con}$, $\mathcal{A}$ requires the knowledge of $\{f_{con-swi}(Con_{id}, Swi_{id}), r_1, r_2\}$. Hence, BACC-SDN resists spoofing attack.

3) Denial-of-Service (DoS) Attack: With our proposed access control policy, SSW establishes a secret key with the controller (SC) through a robust mutual authentication task. A malicious switch is not able to impersonate another switch by making the spoofing attack as discussed in Section IV-A2. The resistance capability from both spoofing and impersonation attacks reduce the chance of malicious activities between control and data planes, and bring down the possibility of controller compromise problem. Furthermore, an ABE policy is also put forward to make a secure communication between SA and SC. This also reduces the feasibility of spoofing and impersonation attacks. Therefore, BACC-SDN resists DoS attack.

Algorithm 1 Consensus for block verification along with addition in blockchain

Input: A block $Block_m$ as shown in Fig. 4
Output: Commitment and addition of block $Block_m$ into blockchain after successful validation

1: Assume controller node (SC_l) acts as the leader having $Block_m$.
2: Generate current timestamp TS_{SC_j} for each follower controller SC_j.
3: Encrypt voting request $VotR$ as $E_{Pub_{SC_j}}(VotR, TS_{SC_j})$ and send a message containing the same block, and $E_{Pub_{SC_j}}(VotR, TS_{SC_j})$ to each follower SC_j, $(j = 1, 2, \cdots, n_{SC}, \forall j \neq l)$.
4: After receiving the message from SC_l by each follower SC_j in the P2P SC network at time $TS^*_{SC_j}$.
5: **for** each follower SC_j **do**
6: Decrypt the message as $(VotR', TS_{SC_j}) = D_{s_{SC_j}}[E_{Pub_{SC_j}}(VotR, TS_{SC_j})]$.
7: Verify timestamp, Merkle tree root, block hash, and signature on received block $Block_m$.
8: If all are verified successfully, send the voting reply $VotL$ and block verification status $BVStatus$ as $\{E_{Pub_{SC_l}}(VotR', VotL, BVStatus)\}$ to SC_l.
9: **end for**
10: If $VTCount$ denotes the vote counter, initialize $VTCount \leftarrow 0$.
11: **for** each received response message $\{E_{Pub_{SC_l}}(VotR', VotL, BVStatus)\}$ from the responded follower SC_j **do**
12: Compute $(VotR', VotL, BVStatus) = D_{s_{SC_l}}[E_{Pub_{SC_l}}(VotR', VotL, BVStatus)]$.
13: **if** $((VotR' = VotR)$ and $((VotL = valid)$ and $(BVStatus = valid)))$ **then**
14: Set $VTCount = VTCount + 1$.
15: **end if**
16: **end for**
17: **if** $(VTCount > 2n_{f_{SC}} + 1)$ **then**
18: Send the commit response to all followers.
19: Add block $Block_m$ to the blockchain.
20: **end if**

4) Man-in-the-Middle Attack: In this attack, an adversary $\mathcal{A}$ may capture the access control request message msg_1 from public channel, and try to tamper the request message and generate another valid message Msg_1'. Without knowing the credentials $f_{con-swi}(Con_{id}, Swi_{id})$ and r_1, $\mathcal{A}$ cannot compute the different values A_{con} and B_{con}. Similarly, for message msg_2, without knowledge the credentials $f_{con-swi}(Swi_{id}, Con_{id})$ and r_2, $\mathcal{A}$ cannot also compute valid A_{swi} and $SKV_{Swi,Con}$. Hence, BACC-SDN is not prone to man-in-the-middle attack.

5) Blockchain-based verification: In BACC-SDN, the block addition by an SC in the P2P SDN controller network is based on the PBFT consensus algorithm. Suppose a verifier $\mathcal{V}$ wants to verify the block $Block_m$ as shown in Fig. 4 in the blockchain. To achieve this goal, $\mathcal{V}$ needs to compute the Merkle tree root MTR^* on the all encrypted transactions in

$Block_m$ and $CBHash^*$ on $Block_m$. If any one of MTR^* and $CBHash^*$ does not match with the stored MTR and $CBHash$, $\mathcal{V}$ rejects $Block_m$. Otherwise, $\mathcal{V}$ further proceeds to verify the block signature $BSign$ using the ECDSA signature verification algorithm. Thus, the block verification goes through a three-level verification process. To update or modify information in a block, delete a block or even insert a fake block in the blockchain, it is quite difficult task for an adversary due to inclusion of previous block hash in the next block hash value computation.

B. Performance Analysis

This section provides a performance analysis on communication and computation costs only for the access control phase of BACC-SDN between controller SC_i and switch SSW_j shown in Fig. 3. For communication cost analysis, it is assumed that the "identity, random number, elliptic curve point ($P = (P_x, P_y)$, where P_x and P_y are x and y coordinates of P respectively), hash output (here SHA-256), and timestamp" need 160, 160, (160 + 160) = 320, 256 and 32 bits, respectively. In BACC-SDN, two messages $msg_1 = \{Con_{id},\ Swi_{id},\ Cert_{con},\ A_{con},\ B_{con},\ TS_1\}$ and $msg_2 = \{Con_{id},\ Swi_{id},\ Cert_{swi},\ A_{swi},\ SKV_{Swi,Con},\ TS_2\}$ demand (160 + 160 +160 + 256 + 256 + 32) $= 1024$ bits and (160 + 160 + 160 + 256 + 256 + 32) $= 1024$ bits, which altogether need 2048 bits. For computation cost analysis in our BACC-SDN, both SC_i and SSW_j requires the same cost as T_{poly} $+7T_h$ $+2T_{ecm}$ $+T_{eca}$, and thus, the total computation cost becomes $2T_{poly}$ $+14T_h$ $+4T_{ecm}$ $+2T_{eca}$, where T_{poly}, T_h, T_{ecm}, and T_{eca} are for the time required to evaluate a t-degree polynomial, "one-way cryptographic hash function", "elliptic curve point multiplication" and "elliptic curve point addition", respectively.

V. Conclusion

In this work, a robust block-chain enabled access control protocol (BACC-SDN) is proposed for securing SDN platform. In BACC-SDN, access control between an application and a controller is achieved through the use of ABE, while the same is achieved between a controller and a switch through a newly proposed access control mechanism. The various transactions among controllers, applications and switches are added in the blocks that are formed by the controller nodes, and those blocks are added by leader selected in P2P SDN network through the PBFT consensus mechanism. The proposed BACC-SDN is then shown to be secure against potential attacks that are essential to secure an SDN network, and its performance analysis is also carried out.

In future, we aim to design a robust accounting and a routine-based auditing mechanism in order to trace-out different activities among various components involved into the SDN framework.

Acknowledgment

This work was supported by the Ripple Centre of Excellence Scheme, CoE in Blockchain (Sanction No. IIIT/R&D Office/Internal Projects/001/2019), IIIT Hyderabad, India.

References

[1] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 27–51, 2014.

[2] O. I. Abdullaziz, L. Wang, and Y. Chen, "HiAuth: Hidden Authentication for Protecting Software Defined Networks," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 618–631, 2019.

[3] W. Jiasi, W. Jian, L. Jia-Nan, and Z. Yue, "Secure software-defined networking based on blockchain," *arXiv preprint arXiv:1906.04342*, pp. 1–19, 2019.

[4] M. Bonola, G. Bianchi, G. Picierro, S. Pontarelli, and M. Monaci, "Streamon: A data-plane programming abstraction for software-defined stream monitoring," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 6, pp. 664–678, 2017.

[5] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for OpenFlow networks," in *Proceedings of the first workshop on Hot topics in software defined networks*. Helsinki, Finland: ACM, 2012, pp. 121–126.

[6] A. Tootoonchian and Y. Ganjali, "HyperFlow: A Distributed Control Plane for OpenFlow," in *Proceedings of the Internet Network Management Conference on Research on Enterprise Networking (INM/WREN'10)*. San Jose, CA: USENIX Association, 2010, pp. 3–3.

[7] K. Phemius, M. Bouet, and J. Leguay, "Disco: Distributed multi-domain sdn controllers," in *Proceedings of the Network Operations and Management Symposium (NOMS'14)*. Krakow, Poland: IEEE, 2014, pp. 1–4.

[8] S. Matsumoto, S. Hitz, and A. Perrig, "Fleet: Defending SDNs from Malicious Administrators," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*. Chicago, Illinois, USA: ACM, 2014, pp. 103–108.

[9] X. Wen, Y. Chen, C. Hu, C. Shi, and Y. Wang, "Towards a Secure Controller Platform for Openflow Applications," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. Hong Kong, China: ACM, 2013, pp. 171–172.

[10] R. Chaudhary, A. Jindal, G. S. Aujla, S. Aggarwal, N. Kumar, and K.-K. R. Choo, "BEST: Blockchain-based secure energy trading in SDN-enabled intelligent transportation system," *Computers & Security*, vol. 85, pp. 288 – 299, 2019.

[11] A. Jindal, G. S. Aujla, and N. Kumar, "SURVIVOR: A blockchain based edge-as-a-service framework for secure energy trading in SDN-enabled vehicle-to-grid environment," *Computer Networks*, vol. 153, pp. 36 – 48, 2019.

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

[13] M. Castro and B. Liskov, "Practical Byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems*, vol. 20, no. 4, pp. 398–461, 2002.

[14] W. E. May, "Secure Hash Standard," 2015, FIPS PUB 180-1, National Institute of Standards and Technology (NIST), U.S. Department of Commerce, April 1995. Available at http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf. Accessed on August 2019.

[15] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data," in *Proc. of the 13th ACM Conference on Computer and Communications Security (CCS'06)*, Alexandria, VA, USA, 2006, pp. 89–98.

[16] V. Odelu, A. K. Das, Y. S. Rao, S. Kumari, M. K. Khan, and K.-K. R. Choo, "Pairing-based CP-ABE with constant-size ciphertexts and secret keys for cloud environment," *Computer Standards & Interfaces*, vol. 54, pp. 3–9, 2017.

[17] C. Blundo, A. D. Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung, "Perfectly Secure Key Distribution for Dynamic Conferences," *Information and Computation*, vol. 146, no. 1, pp. 1–23, 1998.

[18] D. Johnson, A. Menezes, and S. A. Vanstone, "The elliptic curve digital signature algorithm (ECDSA)," *International Journal of Information Security*, vol. 1, no. 1, pp. 36–63, 2001.

[19] H. Zhang, J. Wang, and Y. Ding, "Blockchain-based decentralized and secure keyless signature scheme for smart grid," *Energy*, vol. 180, pp. 955–967, 2019.