BLOCKCHAIN-BASED AUTHENTICATION AND AUTHORIZATION FOR SOFTWARE DEFINED NETWORKS

A Dissertation

by

Majd Latah

Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Doctor of Philosophy

in the
Department of Computer Science

Özyeğin University
May 2023

BLOCKCHAIN-BASED AUTHENTICATION AND AUTHORIZATION FOR SOFTWARE DEFINED NETWORKS

Approved by:

Asst. Prof. Kübra Kalkan, Advisor
Dept. of Computer Science
Özyeğin University

Professor Fatih Alagöz
Dept. of Electrical & Electronics Eng.
Boğaziçi University

Asst. Prof. İsmail Arı
Dept. of Computer Science
Özyeğin University

Professor Albert Levi
Dept. of Computer Science
Sabancı University

Date Approved: May 12, 2023

Professor Murat Uysal
Dept. of Electrical & Electronics Eng.
Özyeğin University

To my family

ABSTRACT

Software-defined networking (SDN) is a novel networking paradigm that allows a simple and flexible management of the underlying forwarding devices through a centralized controller. However, SDN suffers from different security issues that may paralyze the whole network when the controller is under attack. Blockchain (BC) is considered a new technology that provides a decentralized distributed ledger, which can be used to protect the SDN controller from other malicious components in the network.

In this thesis, we investigate the integration between SDN and BC technology. We focus on BC-enabled authentication and authorization for SDNs. First, we propose, DPSEC, a blockchain-based data plane authentication protocol for SDNs. Second, we improve the performance of BC-enabled SDN by proposing a component-wise waiting time approach. We also utilize lattice-based signatures and Key Encapsulation Methods (KEMs) to improve the security of BC-SDN. Third, we introduce, HostSec, a blockchain-based approach that provides mutual host-controller, Packet-In/Packet-Out and host-host authentication for SDNs. Fourth, we propose, SDN-API-Sec, a blockchain-based access control method for cross-domain SDNs by utilizing BC smart contracts. The results suggest a trade-off between security and latency.

ÖZETÇE

Yazılım tanımlı ağ (Software-Defined Networking - SDN), merkezi bir denetleyici aracılığıyla yönlendirme cihazlarının basit ve esnek bir şekilde yönetilmesine izin veren yeni bir ağ paradigmasıdır. Fakat, SDN'de kullanılan denetleyiciye yönelik bir saldırı olduğunda tüm ağı durdurabilecek farklı güvenlik sorunlarına neden olabilmektedir. Blok zinciri (Blockchain - BC), merkezi olmayan dağıtılmış defter sağlayan yeni bir teknolojidir. BC teknolojisi, SDN denetleyicisini ağdaki diğer kötü amaçlı bileşenlerden korumak için kullanılabilmektedir.

Bu tezde, SDN ve BC teknolojisi entegrasyonu üzerine odaklanmaktadır. SDN için BC tabanlı kimlik doğrulama ve yetkilendirme sistemleri araştırılmaktadır. İlk olarak, SDN'deki veri düzlemi için BC tabanlı yeni bir kimlik doğrulama protokolü öneriyoruz. İkinci olarak, bileşen bazında bir bekleme süresi yaklaşımı önererek BC tabanlı SDN'in performansını iyileştiriyoruz. Ayrıca BC-SDN güvenliğini artırmak için kafes tabanlı imzalar ve anahtar kapsülleme yöntemleri kullanıyoruz. Üçüncü olarak, SDN'ler için BC tabanlı karşılıklı ana bilgisayar-denetleyicisi, Packet-In/Packet-Out ve ana bilgisayar-ana bilgisayar kimlik doğrulaması sağlayan yeni bir yaklaşım sunuyoruz. Dördüncü olarak, BC'nin akıllı sözleşmelerini kullanarak SDN etki alanları arasında çalışabilen BC tabanlı yeni bir erişim kontrol yöntemi öneriyoruz. Sonuçlar güvenlik ve gecikme arasında bir ödünleşme olduğunu ileri sürüyor.

ACKNOWLEDGEMENTS

It was a great opportunity to work under the supervision of Asst. Prof. Kübra Kalkan without whom this dissertation would not have been possible as she continuously supports and guides the student to reach the highest possible level.

In addition to his valuable comments during thesis progress meetings, I have learned a lot from Asst. Prof. İsmail Arı when I joined his graduate and undergraduate level classes and also while working as a Teaching Assistant (TA) for "Operation Systems" and "Distributed Systems and Cloud Computing" courses.

I also would like to thank Prof. Murat Uysal for his guidance and feedback during thesis progress meetings and my thesis committee members Prof. Fatih Alagöz and Prof. Albert Levi for their sincere efforts.

Finally, I am grateful to my family for their permanent support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Software-Defined Networking (SDN) is a novel networking architecture that enables network programmability based on a global centralized control design and externally configured network devices (i.e., SDN switches) [8, 9]. SDN switches are programmed according to instructions received from the SDN controller [10]. All SDN switches, which form the SDN data plane, utilize the same protocol (e.g, OpenFlow [11]) to interact with the SDN controller, which results in open, vendor independent, programmable networking [10, 12, 13]. SDN applications, which form the application plane, take advantage of the global view provided by the SDN controller [10].

The centralized design of SDN, however, creates a single point of failure [9]. It also opens the door for new attacks (e.g., unauthorized access to the SDN network), which are mainly due to lack of proper authentication and authorization (AA) mechanisms [9]. For instance, malicious SDN applications can be used to install conflict rules in SDN switches and also access private data [9].

To tackle this issue, authentication services can be used to prove the identity claimed by an entity [14, 15]. After successful authentication, authorization services are also used to allow an authenticated entity to perform specific operations within the corresponding system [14, 15]. By utilizing access control mechanisms we can avoid unauthorized access to critical resources [16].

Choosing the infrastructure for AA services is critical. Centralized AA approaches result in a single point of failure and lacks transparency, which may open the door for Man In The Middle (MITM) attacks [17]. When a centralized AA system is compromised, the attacker can control the whole network [18]. To tackle this issue,

there is a need to envision more secure and reliable SDN architecture.

Blockchain (BC) is a promising technology that introduces a decentralized, distributed ledger [19]. BC can be integrated into the SDN design in order to ensure a level of decentralization in the SDN network and also allow collaboration in cross-domain environments without including any third parties. All transactions in BC can be checked and verified in more transparent manner compared to centralized systems [20].

Clearly, by adopting BC in SDN, we can eliminate the single point of failure, and also increase the security and transparency of SDN due to utilizing a decentralized, distributed ledger. Note that this integration may imply redesigning SDN components in order to adapt the BC technology. This thesis mainly focuses on this integration to explore the potential of BC in SDN. Specifically, we take advantage of BC technology to improve the security and reliability of SDN using blockchain-enabled authentication and authorization mechanisms, which are the building blocks for secure collaborative SDN network.

In chapter 2, we provide some background information on SDN and BC technology. Then, we review existing solutions that consider the integration between SDN and BC. These works are classified into two main categorizes: 1) BC-for-SDN, and 2) SDN for BC. We analyze each category separately. Then, we provide a fine-grained taxonomy based on these two categories. We also highlight both the advantages and drawbacks of this integration. We also introduce DPSEC, as a Proof-of-Concept (PoC) for blockchain-based data plane authentication in SDN. DPSEC is our first attempt towards a BC-enabled SDN data plane. DPSEC considers both SDN switches and hosts in both intra- and cross-domain settings. DPSEC also considers attribute-based access control for SDN switches when interacting with BC. Our host authentication approach is built on top of our switch authentication approach.

In DPSEC, every switch has to wait for the final commitment of each submitted transaction, which decreases the performance of BC-SDN. To tackle this issue, in chapter 3, we propose, CWT-DPA, a solution to enhance the performance of DPSEC by separating switch-related and host-related transactions, where we propose a different waiting strategy for each type of data plane transaction and introduce a method to aggregate host-related transactions and improve the overall latency. In addition, SDN switches and controllers utilize lattice-based signatures and Key Encapsulation Methods (KEMs) to provide protection against quantum adversaries. We compare the performance of CWT-DPA with existing solutions such as SSL/TLSv1.3, DPSEC and AuthFlow [7].

While DPSEC and CWT-DPA focus more on BC-enabled SDN switches, we also need to consider solutions for BC-enabled SDN hosts with a defense solution at the SDN data plane that allows detecting malicious SDN hosts and SDN switches. This is critical as SDN hosts need to interact with the SDN controller and also other hosts. In chapter 4, we introduce, HostSec, a blockchain-based authentication framework for SDN hosts that considers the presence of malicious SDN components in the network. We take advantage of blockchain features to provide mutual host-controller, Packet-In/Packet-Out and host-host authentication methods. We also provide secure ARP (Address Resolution Protocol) and IRP (Identity Resolution Protocol) to protect layer 2 and layer 3 of the SDN network, where the records of layer 2 and layer 3 identifies are stored on the blockchain. Both SDN hosts and controllers utilize lattice-based signatures and KEMs to provide protection against quantum adversaries. HostSec can detect both host-based and switch-based Packet-In attacks and also reduces the load on the SDN controller using our BC-enabled Packet-In manager module since it can verify the messages sent to the SDN controller. We also compare the performance of HostSec with AuthFlow [7].

In chapter 5, we propose, SDN-API-Sec, a cross-domain access control method in SDNs using BC. We take advantage of blockchain technology to protect critical SDN resources. Our proposed system provides a flexible cross-domain access control method using BC-enabled attributed-based access control. We also compare our work with B-DAC [2], which relies on role-based access control. SDN-API-Sec provides a highly flexible mechanism compared with B-DAC [2] since it allows changing the access rules without changing the smart contract. In addition, SDN-API-Sec, can detect explicit and implicit conflicts as well as redundant rules.

Finally, in chapter 6, we conclude this thesis by briefly highlighting our main contributions and providing future directions in BC-SDN research.

CHAPTER II

BACKGROUND

In this chapter, we investigate the integration between SDN and BC. We also show how BC can be utilized to address the limitations of SDN architecture. First, we introduce a background on SDN and BC. Then, we introduce a detailed literature survey that covers the recent research efforts on this topic. We classify these works into two main categories namely SDN for BC and BC for SDN. Then, we provide a fine-grained taxonomy based on these two categories (This study is published in [4]). We also introduce, DPSEC, a Proof-of-Concept (PoC) for blockchain-based data plane authentication in SDN (DPSEC is published in [1]).

The authors in [21] briefly discuss the integration among SDN and BC. They summarize and present a generic framework for BC-enabled SDN based on DistBlockNet [22]. They also discuss the main security challenges in SDNs along with possible solutions. Our work, however, clearly shows that several frameworks with different underlying goals have been proposed in the literature [23, 24, 25, 22, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38]. In another study [39], the author summarizes the existing literature on the same topic. However, this work lacks the appropriate categorization of different proposed solutions. In addition, this work does not discuss the limitations of BC. In contrast, we provide a fine-grained taxonomy that discusses the topic in more detailed and comprehensive manner. We also highlight both the advantages and disadvantages of this integration.

2.1 Software-Defined Networking (SDN)

SDN has received considerable attention due to its central management and high flexibility. SDN replaces the *local* control plane in conventional hardware devices by a *global* controller, which programmatically manages packet forwarding on a per-flow level. The SDN architecture simplifies decision making and facilitates development of new protocols as well as various SDN-tailored network applications.

The controller manages the underlying data plane using the southbound application programming interface (API), where OpenFlow [11] is the most prominent example of such an API. The northbound API, on the other hand, provides a common interface for developing SDN applications [40]. An OpenFlow switch has several flow tables, each of which has matching rules along with actions that are executed when an incoming packet matches the corresponding rule. As a result, the behavior of an OpenFlow switch changes according to the rules installed by the SDN controller [40].

When no match is found, for a newly received packet, then the switch encapsulates and sends the packet to the SDN controller using Packet-In message [11, 41]. Next, the controller takes decision using Packet-Out and Flow-Mod messages [11, 41]. In SDN, security is mainly integrated in the control plane, whereas mitigation of data modification/leakage takes place in the data plane [26]. However, SDN propounds several additional threats such as single point of failure [42], DoS /Distributed DoS (DDoS) attacks [43], and illegal access through malicious SDN apps [44]. Unfortunately, these threats have a high impact that could ultimately prevent large-scale deployment of SDNs [30].

Authentication and Authorization Solutions in SDN

In [7], the authors introduced, AuthFlow, an authentication mechanism for SDNs in accordance with IEEE 801.X standard and extensible authentication protocol (EAP) where MS-CHAPv2 (Microsoft Challenge-Handshake Authentication Protocol) is used as an authentication method. AuthFlow [7] depends on a separate authenticator which acts as a RADIUS client that needs to interact with a RADIUS server to authenticate the corresponding host. AuthFlow [7] cannot detect switch-based Packet-In attacks since it only performs MAC/port matching after the authentication is completed. These data can be easily manipulated by malicious SDN switches.

In [45], the authors presented an authentication and authorization (AA) module for SDNs. The module has two modes of operation, namely pass-through mode and authentication server mode. In pass-through mode, it relies AA requests to a RADIUS server. In authentication server mode, it can handle these requests according to attached AA resources. AA utilizes traditional AA protocols such as EAP and RADIUS. In both [7, 45], the authentication process is done at the control plane, and therefore these approaches cannot detect manipulations at the data plane.

Yiğit et al. [46] presented a modified SDN architecture in which certificates are used for securing the communication among SDN entities (controller, switch, and application). This work enhanced the security of SDNs by incorporating an access control list (ACL) and hardening Transport Layer Security (TLS) configuration. However, key generation is implemented as an SDN application, which also issues and manages corresponding X.509 certificates, which eventually may result in a single point of failure.

Liyanage et al. [47] utilized Host Identity Protocol (HIP) (RFC 7401)[1] to authenticate SDN switches. Each HIP host uses its public key as its own identity. The protocol uses Diffie-Hellman to exchange keys among HIP nodes. HIP is designed as an end-to-end authentication protocol to be used along with Encapsulating Security Payload (ESP) (RFC 7402)[2]. This approach requires significant modifications to SDN architecture and its switch design since IPSec-enabled security gateways are used as an intermediary between the controller and SDN switches. Ranjbar et al. [48] improved the quality of secure sessions in SDN data plane by analyzing the handshake messages exchanged between the controller and switches. However, the overall load on the controller is increased since it needs to analyze all the handshake messages exchanged between the controller and underlying SDN switches.

In [49], the authors suggested identity-based cryptography (IBC) approach to secure SDN data plane. Public keys are derived from each user's identity, and then used to deliver symmetric session keys via symmetric key distribution [46]. Gray et al. [50] proposed a fingerprinting-based method to authenticate SDN switches. However, since it depends on static features, attackers can also imitate the behavior of legitimate switches in order to mislead the SDN controller.

2.2 Blockchain (BC)

Recently, blockchain (BC) has gained increasing attention due to its capability as a fundamental technology for cryptocurrencies such as Bitcoin [51] and Ethereum [52]. Bitcoin [51] is the first BC and also the most popular cryptocurrency, whereas Ethereum [52] represents a generalized BC platform for developing decentralized apps. BC is basically a decentralized distributed ledger that provides trustworthy services to

[1]IETF (2015). RFC 7401: Host Identity Protocol Version 2 (HIPv2) [online]. Website https://datatracker.ietf.org/doc/html/rfc7401 [accessed 07 May 2021].

[2]IETF (2015). RFC 7402: Using the Encapsulating Security Payload (ESP) Transport Format with the Host Identity Protocol (HIP) [online]. Website https://datatracker.ietf.org/doc/html/rfc7402 [accessed 07 May 2021].

members of a peer-to-peer network without relying on any central authority [28]. The decentralized distributed ledger is built and maintained through consensus among all BC members in a distributed fashion to achieve consistency while ensuring adequate fault tolerance [31]. BC systems aim to achieve higher level of security and reliability compared to most of non-BC systems [53]. In other words, BC systems can operate in potentially hostile settings [53].

In BC, series of data blocks are generated using cryptography to be secure, immutable, and tamper-proof [30]. Each block contains the hash value of the previous block header, transaction data, a timestamp, and a nonce. Consequently, multiple blocks are connected together in order to form a *chain*. Each new block is added through an agreement between BC nodes, thanks to *consensus* protocols, which define how to reach to an agreement without relying on any trusted third-party intermediaries. The structure of blockchain is shown in Figure 1.

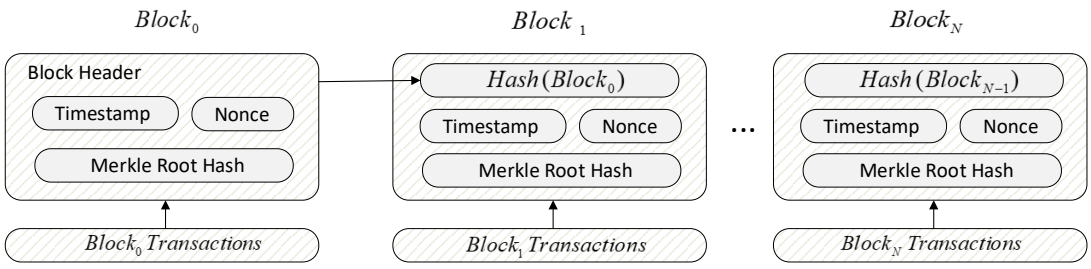


Figure 1: Blockchain structure [3].

In general, there are three types of BC, including public, private, and consortium BCs. In public BC, every node can be involved in block creation whereas in private and consortium BCs only specific nodes can participate in this operation using a valid certificate obtained by an identity management infrastructure such as certificate authority (CA) [28]. In public BCs, committed transactions are visible to everyone while the read permission is restricted to participating nodes in private BCs. In consortium BCs, the organization may decide whether the approved transactions are

public or private [54]. Private BCs are fully controlled by one organization whereas consortium BCs consist of more than one organization. Private and consortium BCs are more efficient compared to public BCs. The fact of having fewer validators results in higher transaction throughput and lower latency [54]. Hyperledger Fabric [55] is the most notable example for a business-oriented consortium BC.

In addition, BC platforms, such as Ethereum [52] and Hyperledger Fabric [55], provide smart contracts, which are automatically executed according to a pre-configured script code based on verified data provided by decentralized distributed ledgers [31]. In this context, BC systems need to maintain backward compatibility in order to validate earlier transactions and potentially to be able to interact with older versions of a deployed smart contract [53]. BC developers may also need to test and verify the security requirements of their smart contracts before being deployed [53].

Several approaches were proposed to reach consensus between BC nodes. Proof-of-Work (PoW) [51] is used in Bitcoin network. In PoW, network nodes need to solve a computationally difficult problem (i.e., cryptographic puzzle) in order to create a new block [56], where 51% of computational capability is considered to be the threshold of PoW to gain control of the BC network [54]. The PoW procedure wastes the resources of the mining nodes in order to be authentic [54]. As an alternative to PoW, Proof-of-Stake (PoS) requires the proof of ownership to validate a newly generated block, considering those with more cryptocurrencies are more trustful than those with fewer cryptocurrencies [56]. Compared with PoW, PoS saves more energy and is more effective. However, attacks might occur since the mining cost is nearly zero [54]. Practical Byzantine Fault Tolerance (PBFT) [57] can also be utilized as a consensus algorithm and can handle up to 1/3 malicious byzantine failures, where each node voting for the consensus sends its vote to the other nodes [56]. In each phase, a node will move to the next phase once it receives votes from 2/3 of all BC nodes [54]. Hyperledger Fabric [55] (in its version 0.6 and Hyperledger Sawtooth) employs

PBFT to reach consensus among BC nodes. It is possible that valid blocks might be generated simultaneously when several nodes solve the puzzle at the same time, which results in branches (or forks) [56]. PoW and PoS solve this issue by choosing the longest chain whereas PBFT handles this issue through multiple communication rounds [56]. We refer the reader to [56, 54] for further details.

2.3 The Integration between SDN and BC

BC can address the limitations of SDN in terms of security, reliability, and single point of failure by introducing a decentralized control-plane and securely sharing critical information at different levels including the application, the resource, and the flow levels. However, this would present new challenges in terms of how BC can meet the throughput and latency requirements of SDN. On the other hand, SDN can be utilized to improve the availability and to provide access control for BC nodes enforced by network devices and SDN control plane.

Distributed architectures improve load distribution and avoid the single controller failure; however, they bring additional overhead from controller-to-controller communication and increase the latency due to the need for consistent state/policy and synchronized global network view among different SDN controllers. On the other hand, decentralized architectures have attracted the attention of the research community as a solution to address the limitations of SDN. In this regard, BC is considered to be a promising approach, which acts as an out-of-band party that can be utilized to achieve a consistent and synchronized global view among different SDN controllers. BC can also ensure *security*, *dependability*, and *traceability* requirements of SDN [28]. Security is required to define who has access to what, when, and in which conditions. This system should also be highly dependable in terms of reliability and availability. It also needs to provide an undisputable proof that cannot be fabricated or forged in possible compromises.

SDN as a networking layer for BC would provide a better protection for BC nodes. For instance, it can be used to maximize the availability of participating nodes against DoS/DDoS attacks [32] and to achieve flow-based access control [33]. The existing integration approaches can be classified according to many different criteria. However, the classification we have found to be the most beneficial is primarily focusing on the utilization area. This can be a useful basis for researchers seeking to identify the range of applicability, and the mutual benefits of combining these two different technologies. Therefore, we classify this integration as BC for SDN and SDN for BC. The first category improves SDN through BC, whereas the second category makes use of SDN to enhance BC. Figure 2 illustrates both SDN and BC architectures along with the major issues that can be solved by combining these two paradigms.

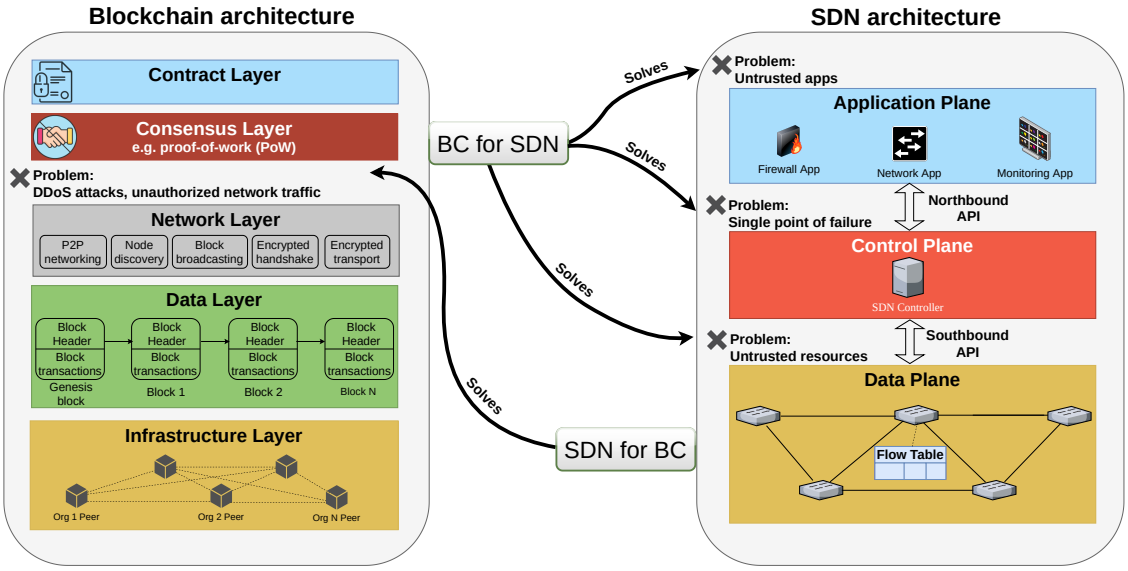


Figure 2: The integration between SDN and BC [4].

2.3.1 BC for SDN

As depicted in Figure 2, the SDN architecture consists of untrusted apps, resources, and controllers, which ultimately need to interact with each other. BC for SDN approaches make use of BC to secure traditional SDN paradigm. We categorize them into the following two groups: (1) operational solutions that focus on enhancing

the functionality of SDN and (2) structural solutions that provide core architectural refinements for the SDN paradigm.

2.3.1.1 Operational Solutions

The operational solutions utilize BC to secure the core functionality of SDN through a cooperative defense among SDN controllers, and to protect that collaboration by authenticating and establishing trust at the controller, the application, and the flow levels. Accordingly, these solutions can be classified into the following three subgroups: cooperative defense, trust enhancement, and decentralized authentication.

Cooperative defense Cooperative defense focuses on detecting sophisticated and coordinated attacks that require collaboration among different SDN controllers, which will allow a more effective mitigation near the origin of the attack and reduce the high volume of traffic exchanged across different SDN domains. BC is used to securely share critical information related to cooperative defense models and core SDN functionality such as routing among several network domains.

El Houda et al. [23] propose a cross-domain collaborative DDoS mitigation scheme based on BC smart contracts, which allow multiple SDN-based domains to securely collaborate and transfer attack information in a decentralized manner. The collaboration is managed by the smart contract's owner who can add/remove the collaborators that report suspicious IP addresses within their domains. This approach provides a secure, low-cost, and flexible solution to mitigate DDoS attacks in cross-domain SDNs.

Rathore et al. [29] introduce a decentralized and cooperative architecture for IoTs using a combination of SDN and BC as well as fog and edge computing. Attack detection is performed at the fog layer using deep learning approach. Fog nodes and a cloud server share data using BC, which is also used to regularly update the attack detection model and flow rules in the SDN switches. The cloud server acts as a manager that manages and initiates the attack detection models using BC smart

contracts.

Qiao et al. [27] propose a BC-based approach to establish trust among SDN controllers for cross-domain routing. There are two types of transactions, the first one is related to cross-domain routing, and the other is related to network state information updating. Each transaction must be recognized by at least one node in each domain.

Cooperative defense approaches take advantage of BC to secure the information shared between SDN controllers and related applications, which removes the need to use a central entity and also reduces the complexity of developing new protocols and/or modifying the existing ones. The effectiveness of these protocols also depends on global deployment and may suffer from single point of failure [23].

Trust enhancement Trust enhancement approaches make use of BC to establish trust among SDN controllers and related devices, which are connected within an SDN network. Trust management includes that each device connected to the SDN network to be registered into BC along with a trust level that changes over time based on the recent activities of that particular device. The advantage here is that BC ensures that trust scores stored within BC are secure, immutable, and tamper-proof. Moreover, trust scores are stored through consensus among all BC nodes, which are more immune to fake information submitted by malicious devices.

Kataoka et al. [25] utilize BC to prevent unsolicited traffic coming from IoT devices positioned at the edge of the network in a trustworthy, scalable, and distributed manner. To this end, they implemented a trust list across widely distributed edge networks using BC and SDN. First, SDN controllers receive the recent updates of BC, which contains profiles of validated devices. Then, they check whether the device is connected to their network. Finally, new flow rules are installed to allow that particular device to communicate with the server based on an existing service profile.

Zhao et al. [36] focus on protecting the control plane of SDN through a trust management architecture. More precisely, they propose a secure and efficient resource allocation for software-defined vehicular networks (SDVN). The main goal here is to protect SDVN against fake information submitted by malicious vehicles, which aim to deteriorate the quality of service (QoS) of other legitimate vehicles. A joint proof-of-stake and modified practical Byzantine fault-tolerance (PoS-mPBFT) algorithm is proposed to shorten the confirmation time and reduce the communication overhead of PBFT [57], where a fully trustworthy CA is used to collect, compare, and choose the leader.

Decentralized authentication Decentralized authentication approaches utilize BC to authenticate legitimate devices that are connected to an SDN network. BC securely stores the information related to legitimate devices and reduces the re-authentication overhead among different SDN controllers.

In [35], Yazdinejad et al. propose a BC-enabled handover authentication for 5G within SDN. This approach provides secure and fast authentication by employing BC, which eliminates re-authentication in repeated handovers among heterogeneous cells, in which unique characteristics of mobile users are shared between current and adjacent cells. In this work, BC manages the control plane of SDN and sends user's information to the SDN controller of that particular cell. Preshared-key handshakes are eliminated by using BC. In other words, the messages are approved by the BC instead of a third party. SDN, on the other hand, prepares the flow tables in accordance with a given policy.

Jindal et al. present SURVIVOR [38], an edge-as-a-service platform that utilizes BC to provide a secure energy trading in SDN-enabled vehicle-to-grid (V2G) environments, where BC is utilized to authenticate the energy trading transactions in the system using the PoW [51] approach. The study showed that the time required to

generate header value and validate the transaction increase as the number of transactions increases.

The authors in [26] introduce a BC-based approach for forensic purposes in SDN-based IoT environments. BC along with Linear Homomorphic Signature (LHS) algorithm are used to authenticate IoT devices based on a unique identity and Elliptic curve cryptography. An unauthenticated IoT device sends packets to the gateway, which forwards these packets to the switches and then authenticates them via the signature in SDN controllers. Event logs are used and stored on the blockchain.

Access Control BC can be used as an access control mechanism for SDN components and their associated critical assets since only authorized entities can access the BC. In addition, each entity signs related BC transactions to prove their identity. In this context, Yazdinejad et al. [37] employ BC as an access control method for the IoT devices and their associated data, where they present a secure and energy-efficient BC-enabled SDN architecture for IoT networks using a cluster structure, in which each SDN controller acts as a cluster head. The SDN controller manages several processes in each SDN domain via a private BC whereas different SDN controllers securely share data related to trusted IoT devices through a public BC. The authors replaced PoW with an authentication method in order to improve the performance of BC.

In [58], the authors proposed BCNBI, a permissioned blockchain-based access control method for SDN applications. Permissions are categorized into two main groups namely strict and normal. However, the system includes centralized authentication and authorization API, which interacts with blockchain and JSON Web Token (JWT) model. Blockchain is used only for storing SDN controller credentials as well as user and policy headers. The authentication token is generated by the centralized JWT model.

In [2], the authors introduced B-DAC, an access control framework based on BC approach. B-DAC checks the permissions of each request sent by SDN application for accessing the northbound API. For each application, a policy is defined with the right set of permissions as well as the corresponding role of the corresponding application. B-DAC includes three main steps. The first step includes initial authentication using REST API based on JWT model. The second step includes sending a new signed transaction (requestAppToken) by the corresponding application to B-DAC system, which returns a new token once the app receives the required endorsements. The third step includes authorization based on interaction between the controller and B-DAC system using tokens received from the corresponding application.

2.3.1.2 Structural Solutions

The structural solutions take advantage of BC to enhance the structure of SDN paradigm. These solutions can be also categorized into the following groups: switch-level improvement, global view protection, and hybrid approaches. Switch-level improvement utilizes BC to enhance SDN's data-plane. Global view protection shares SDN's global view through BC. Hybrid approaches apply BC technology to secure the SDN paradigm at different levels including the control and the data planes as well as the communication channel between the controller and the switch.

Switch-level improvement The switch-level improvement approaches enhance SDN's data-plane in terms of protecting flow tables and including the ability to utilize BC in SDN switches. In [22], Sharma et al. present DistBlockNet, a switch-level solution, in which flow tables are verified, validated, and securely updated. The experimental results showed that DistBlockNet is capable of detecting attacks in the IoT network in real time with low performance overheads.

Yazdinejad et al. [34] propose a BC-enabled packet parser (BPP) to support detecting attacks as well as BC structure in SDN's data plane. BPP supports the policies and rules defined by the control plane. When BPP in a given SDN switch detects a malicious behavior, it will inform the corresponding SDN controller, and then it will report that incident to other BPPs on the basis of P2P communication among other SDN switches.

Global view protection The global view protection approaches focus on using BC to securely share and synchronize SDN's global view among multiple controllers. As BC provides an immutable and tamper-proof ledger, it would prevent external attackers from tampering with the global view provided that the fraction of malicious nodes does not exceed 1/3 of the total number of BC nodes, assuming PBFT [57] is used. In other words, BC is utilized as an out-of-band party to $safely$ and *dependably* reach consensus among different SDN controllers, without affecting the core functionality of SDN network. Qiu et al. [28] use BC to collect and synchronize the global view among different SDN controllers. In this work, each controller collects its local events and OpenFlow [11] commands and then stores them into the best BC system at the edge network.

Shao et al. [30] collect view related data through features ($Features\_Reply$), statistics ($Stats\_Reply$), and $Packet\_In$ messages, and then store them on the BC in order to protect the global view of SDN. The requests for storing the global view are verified by the other controllers. To overcome the limitations of PBFT [57], they propose a Simplified PBTF (SPBFT), in which the messages are transferred and verified in a parallel manner. It also prioritizes the received requests to ensure that urgent requests are handled quickly and efficiently.

Hybrid approaches The hybrid approaches apply BC technology at different levels including: the control and data planes as well as the communication channel between the controller and the switch. Yang et al. [31] propose BlockCtrl, a BC-based architecture to secure software-defined optical networking (SDON). In BlockCtr, a customizable smart contract is installed in each controller, which automatically performs the network functions loaded in the smart contract. On the other hand, the SDN switches use BC smart contracts to choose the optimal secondary controller on failure. BlockCtr achieves better service correlation and resource utilization when compared with BFT [57], since BlockCtrl considers the traffic data stored in the BC ledger when selecting a new primary controller.

In [24], the authors use BC to address multiple security issues in SDN. They design a monolithic security mechanism for SDN based on BC, which decentralizes SDN's control plane in order to tackle the single point of failure while maintaining the global view by sharing network events among multiple controllers. High performance secure Diffie-Hellman protocol is combined with BC to authenticate the communication channel between the controller and the switch. This approach is considered as a hybrid approach since it utilizes BC to protect the global view at control plane level and to authenticate the communication channel between the controller and the switch.

The main advantages of BC for SDN

1) Provides a decentralized control-plane.

2) Authentication for application flows.

3) Securely sharing data in cross-domain SDNs without relying on third parties.

4) Ensures reliability, traceability, and auditability.

5) Improves trust management.

6) Facilities collaborative defense approaches.

The main disadvantages of BC for SDN

1) Characterized by high power consumption.

2) Additional latency and lower throughput due to computation and communication overhead.

3) Its performance depends on consensus protocols.

4) Data replication requires additional space.

5) BC transactions and smart contracts are immutable.

2.3.2 SDN for BC

In permissioned BC, only certain nodes can be involved in block creation and validation [28]. Permissioned BC uses Byzantine Fault Tolerance (BFT) protocols, such as PBFT [57], as a consensus protocol that employs state machine replication in order to cope with Byzantine failures. The advantages of permissioned blockchains are low cost and low latency [28]. However, their performance is affected when BC nodes are faulty or under attack. To address this issue, SDN can be utilized to enhance the availability of BC nodes and to protect them against unauthorized access and various network attacks. As shown in Figure 2, SDNs can be used as a framework to protect and optimize the network layer of BCs. Accordingly, we categorized the research efforts to secure BC through SDN into the following two groups:

2.3.2.1 Availability

As we mentioned previously, permissioned blockchains have a limited number of participants. Therefore, maximizing nodes' availability is a major issue in such peer-to-peer networks. To this end, Steichen et al. [32] propose ChainGuard, an SDN-based firewall for protecting permissioned BCs against DoS/DDoS attacks, which can influence how consensus is reached and can even prevent BC from working correctly. To prevent malicious actions, all traffic to BC nodes has to be forwarded by at least one of the switches controlled by ChainGuard, which directly communicates with the BC

nodes to distinguish between legitimate and malicious traffic that need to be filtered in order to protect BC nodes.

2.3.2.2 Flow-based Access Control

BC nodes need to be protected against intruders that deliberately attack the infrastructure of BC to gain access or attempt to disturb legitimate BC transactions. To address this issue, El Houda et al. [33] present ChainSecure to protect permissioned blockchains from DNS amplification attacks. ChainSecure consists of the following three schemes. First, stateful mapping scheme (SMS) that performs one-to-one mapping between DNS requests and responses. Second, entropy calculation scheme (ECS) to measure randomness of data in order to detect illegitimate DNS requests using sFlow. Third, DNS DDoS mitigation module. The experimental results showed that ChainSecure can achieve high detection rate with approximately 30% of false positives. Figure 3 depicts the taxonomy for the integration between SDN and BC.

The main advantages of SDN for BC

1) Enhances the network layer of BC.

2) Improves the availability of BC nodes.

3) Provides flow-based access control for BC nodes.

The main disadvantages of SDN for BC

1) Protects BC only from network-based attacks.

2) Attackers may exploit traditional SDN vulnerabilities to gain access to BC.

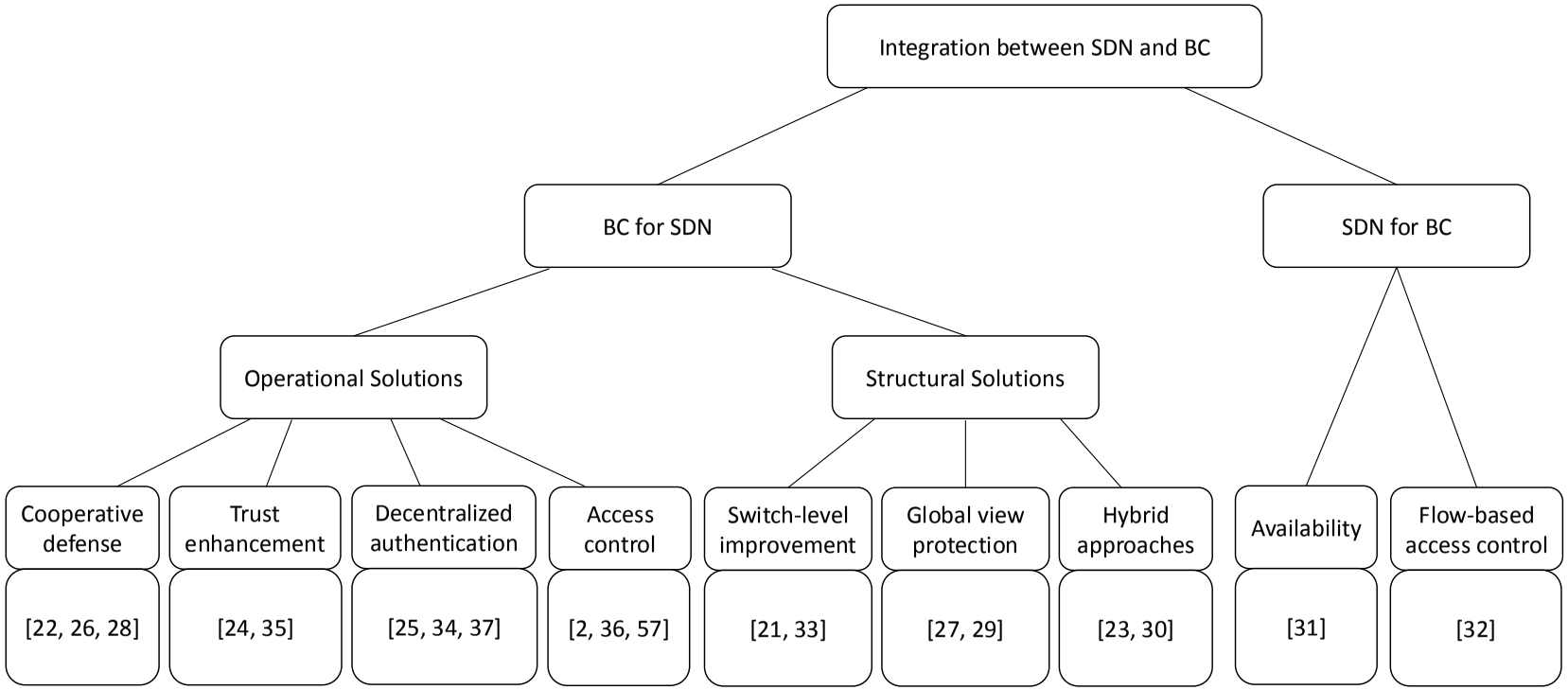


Figure 3: Taxonomy for the integration between SDN and BC [4].

2.4 Proof-of-Concept (PoC)

This section focuses on authenticating SDN's data plane since it can be used to paralyze the the SDN controller. BC can fundamentally eliminate the limitations of SDN in terms of security and reliability by its unique characteristics. This is due to the fact that BC empowers a decentralized, distributed, immutable ledger that provides trustworthy services on peer-to-peer basis without relying on third-party intermediaries [28], which may belong to multiple network and service operators [59]. We show how BC can be utilized to provide a decentralized authentication for SDN's data plane components including SDN switches and hosts requesting to join the SDN network. We chose consortium blockchain approach since we aim to authenticate SDN data plane in intra- and cross-domain settings (This study is published in [1]).

2.4.1 Threat Model

Our protocol is mainly designed to react against malicious SDN data plane. We assume that the attacker does not possess any cryptographic keys related to all SDN nodes or blockchain clients, and the signature scheme and Certificate Authority (CA) used for each SDN domain are secure (i.e., not possible to forge a signature without knowing the private key). As our focus is on authenticating SDN's data plane, we assume that the controller is authenticated through SSL/TLS[3] and is not compromised. Spoofing attacks are allowed. Therefore, internal attacks can be launched from legitimate SDN switches/hosts, which may try to act as another legitimate SDN switch/host (i.e., perform switch spoofing (1) or host spoofing (2)) and also may add unauthorized flows (3). External adversaries may utilize rogue switches/hosts to launch different attacks including switch spoofing (4) and host spoofing (5) as well as DoS/DDoS attacks (6) against the SDN controller (see Figure 4).

[3]IETF (2018). RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3 [online]. Website https://www.rfc-editor.org/rfc/rfc8446 [accessed 06 May 2021]

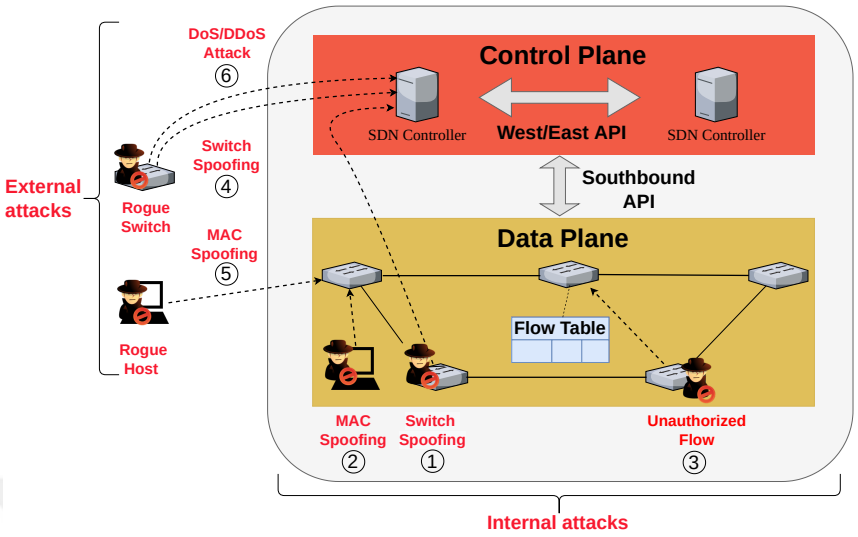


Figure 4: Threat model in DPSEC © 2020 IEEE [1].

2.4.2 Proposed Authentication Protocol

In this section, we propose DPSEC a protocol that leverages BC to authenticate the data plane of SDN. We ensure that OpenFlow events are received from *trusted* devices. Consequently, SDN controllers will only consider the events received from those trusted devices. We employ BC smart contracts to provide a decentralized and fault-tolerant authentication of SDN's data plane. Our system consists of a cross-domain SDN network denoted as $D = [D_1, ..., D_\ell]$ managed by a group of SDN controllers $C = [C_1, ..., C_\ell]$. Each SDN domain D_i includes a group of SDN switches $S_i = [s_{i,1}, ..., s_{i,j}]$ and SDN hosts $H_i = [h_{i,1}, ..., h_{i,k}]$ managed by C_i. In addition, $B_{i,j} \subset S_i$ indicates the set of border switches that exist in D_i and connect between D_i and D_j. We consider a consortium BC to securely authenticate the data plane of domain D_i consisting of $S_i \cup H_i$ over $C_{auth} \subseteq C$. Each domain D_i includes a group of pre-selected endorsing peers denoted by $E_i = [e_{i,1}, ..., e_{i,n}]$ and a group of committing peers represented by $V_i = [v_{i,1}, ..., v_{i,m}]$. A peer can act as an endorser and committer, therefore $m >= n$. The set $V = \bigcup_{k=1}^{\ell} V_k$ contains all committing peers in D. Each node $v \in V$ writes accepted transactions to its own replica of BC ledger. Assuming that less than one-third of the endorsing peers in D_i are faulty (f_i), then D_i should

consist of at least $n = 3f_i + 1$ endorsing peer to tolerate faulty peers. Thus, DPSEC requires $2f_i + 1$ peers to agree on BC transactions submitted from D_i. Thus, we define $E_{(i)accept} \subseteq E_i$ as the minimum cardinality subset of endorsing nodes required to accept a transaction T_i in D_i where $E_{(i)accept} = \{(e_1, ..., e_n) \mid e \in E_i, n = 2f_i + 1\}$. From BC perspective, both C_i and $s_{i_j} \in S_i$ are BC clients, which can submit and verify BC transactions. However, C_i has a higher level of privilege compared with $s_{i,j}$ on the basis of attribute based access control (ABAC). Each D_i has its own certificate authority CA_i managed by C_i and responsible for providing digital certificates, and delivering cryptography keys for each SDN switch $s_{i,j} \in S_i$ and host $h_{i,j} \in H_i$.

Trusted Switches

Each SDN controller $C_i \in C$ has a $Trusted\_Switches$ table, which includes information related to switches $s_{i,j} \in S_i$ authenticated through BC smart contract.

Trusted Ports

Each SDN controller $C_i \in C$ has a $Trusted\_Ports$ table, which includes port-related information for hosts connected to an authenticated switch $s_{i,j} \in S_i$.

2.4.2.1 Registration

SDN controller C_i and SDN switches in S_i are registered and enrolled to BC using the existing CA_i, which is used in order to achieve the access control policy for domain D_i and for the corresponding BC peers as well. The controller C_i has an identity with admin role whereas SDN switches $s_{i,j} \in S_i$ are registered with client role. Additionally, SDN hosts $h_{i,j} \in H_i$ are registered with user role.

2.4.2.2 Attribute-based Access Control

DPSEC utilizes attribute-based access control (ABAC) [60] in order to provide a simple, flexible and fine-grained access control for BC-enabled SDN data plane. CAs

issue certificates with attribute values, which are used to achieve access control in DPSEC. These attributes are issued by CAs according to the role of each entity in DPSEC. Therefore, each switch has a unique identify along with its corresponding attributes such as *Role*, $DPID$ and $ControllerID$ obtained from CA_i. Each host also has a unique identify along with its corresponding MAC attribute. In DPSEC, smart contracts retrieve these attributes to check whether the caller is allowed update the BC state and perform the requested functionality.

2.4.2.3 Intra-domain Switch Authentication

After successful enrollment to CA_i, each SDN switch $s_{i,j} \in S_i$ can interact with BC using its own Pub_K and certificate $Cert_{(sw)i,j}$ signed by CA_i. Next, we describe our switch authentication mechanism in the following steps (see Figure 5):

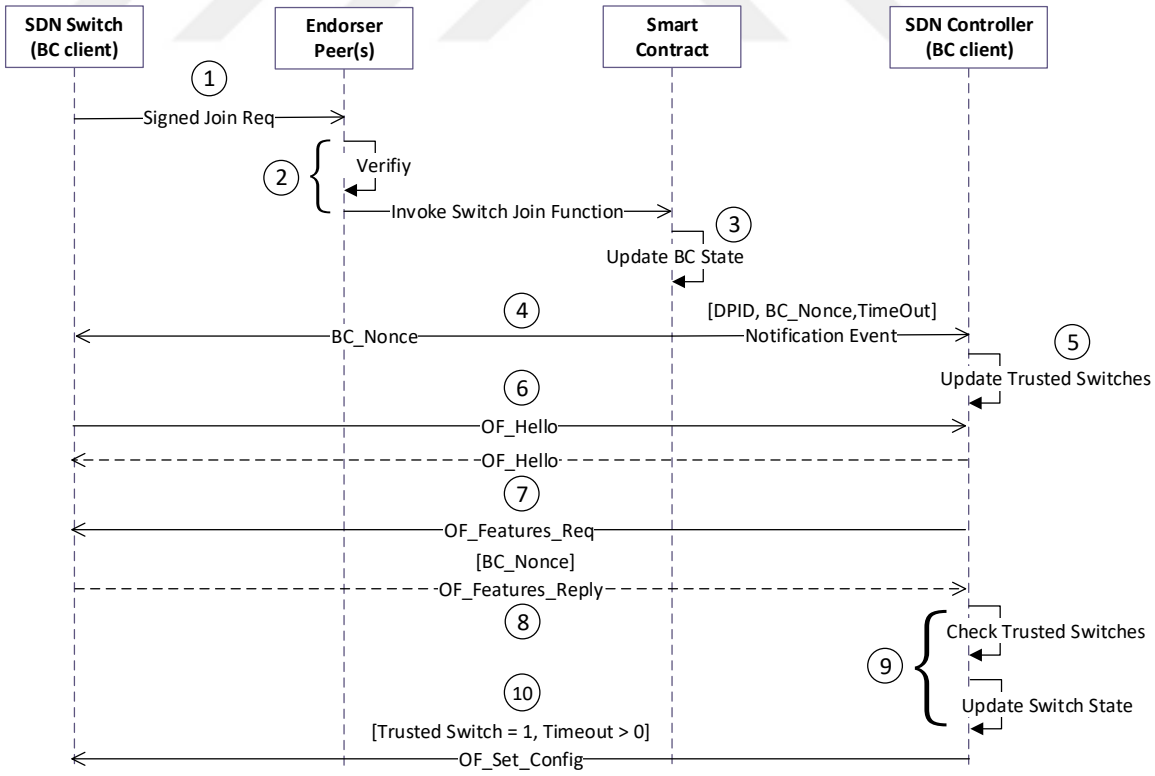


Figure 5: Switch authentication in DPSEC © 2020 IEEE [1].

Intra-domain Switch Authentication Steps

Step (1) Switch $s_{i,j}$ sends signed $Join\_Request$ along with its certificate, that also includes its Pub_K, to the at least one endorsing peer(s) $e \in E_{(i)accept}$.

Step (2) Each endorsing peer $e \in E_{(i)accept}$ verifies $Cert_{(sw)i,j}$ and it signature $Sig_{(sw)i,j}$. Then invokes a smart contract that includes $Switch\ Join\ Function$ (see Algorithm 1) responsible for updating the BC state with a new transaction that consists of a combination of $DPID$, $Timestamp$ and $TimeOut$.

Step (3) Upon acceptance, each committing peer $v \in V$ updates its own replica.

Step (4) The smart contract returns a 2 byte nonce (we call it BC_Nonce) to $s_{i,j}$ and emits a new event to the notify the controller $C_i \in C_{auth}$.

Step (5) C_i updates its own $Trusted\_Switches$ table, which includes the information received from the smart contract.

Step (6) $s_{i,j}$ initiates OpenFlow handshake using OF_Hello.

Step (7) The controller C_i responds with OF_Hello and $OF\_Features\_Req$.

Step (8) $s_{i,j}$ then replies with our modified $OF\_Features\_Reply$ message, which includes BC_Nonce received from the smart contract, where we replaced 2 padding bytes of $OF\_Features\_Reply$ with BC_Nonce since they are not utilized.

Step (9) C_i matches the information received in $OF\_Features\_Reply$ against $Trust\_Switches$ table entries, then updates $s_{i,j}$ state to $Trusted$ state.

Step (10) Finally, C_i configures $s_{i,j}$ using OF_Set_Config.

Algorithm 1: Switch Join Function [1]

Input: caller, DPID
caller BC client;
if *caller.role == client* **then**
  if *caller.dpid == DPID* **then**
    Add a new record (DPID, *Joined_BC*, timestamp);
    Emit a new event to notify C_i that $s_{i,j}$ has joined BC with generated *BC_Nonce*;
    Return *BC_Nonce* to the *caller*;
  else
    Reject the request;
    Update *caller* state to *Untrusted*;
    Emit a notification event for *caller.dpid* new status;
  end
else
  Reject the request;
end

2.4.2.4 Cross-domain Switch Authentication

As shown in Figure 6, border switches mainly connect two separate domains (D_i, D_j). Therefore, they must be endorsed by peers $e \in E_i \cup E_j$. Since we already defined $E_{(i)accept}$ and $E_{(j)accept}$, we define $E_{(i,j)accept} = E_{(i)accept} \cup E_{(j)accept}$ as the minimum

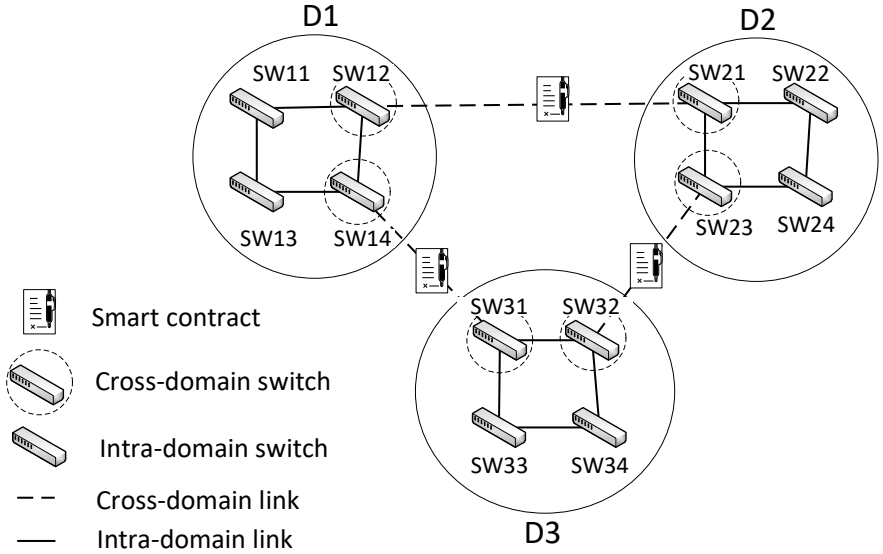


Figure 6: Cross-domain SDN in DPSEC © 2020 IEEE [1].

cardinality subset of endorsing nodes required to accept a transaction T_i over D_i and D_j. Therefore, we deploy a smart contract with an endorsement policy defined by $E_{(i,j)accept}$, where $|E_{(i,j)accept}| = 2f_i+2f_j+2$. Also, SDN controllers C_i, $C_j \in C_{auth}$ have $Trusted\_Switches$ tables that include information related to $S_{(i,j)cross} = B_{i,j} \cup B_{j,i}$.

2.4.2.5 Intra-domain Host Authentication

Next, we build our host authentication mechanism on top of our switch authentication approach, where host $h_{i,j} \in H_i$, which is connected to switch $s_{i,j} \in S_i$, can interact with BC using its own Pub_K and certificate $Cert_{(h)i,j}$ signed by CA_i as follows (see Figure 7):

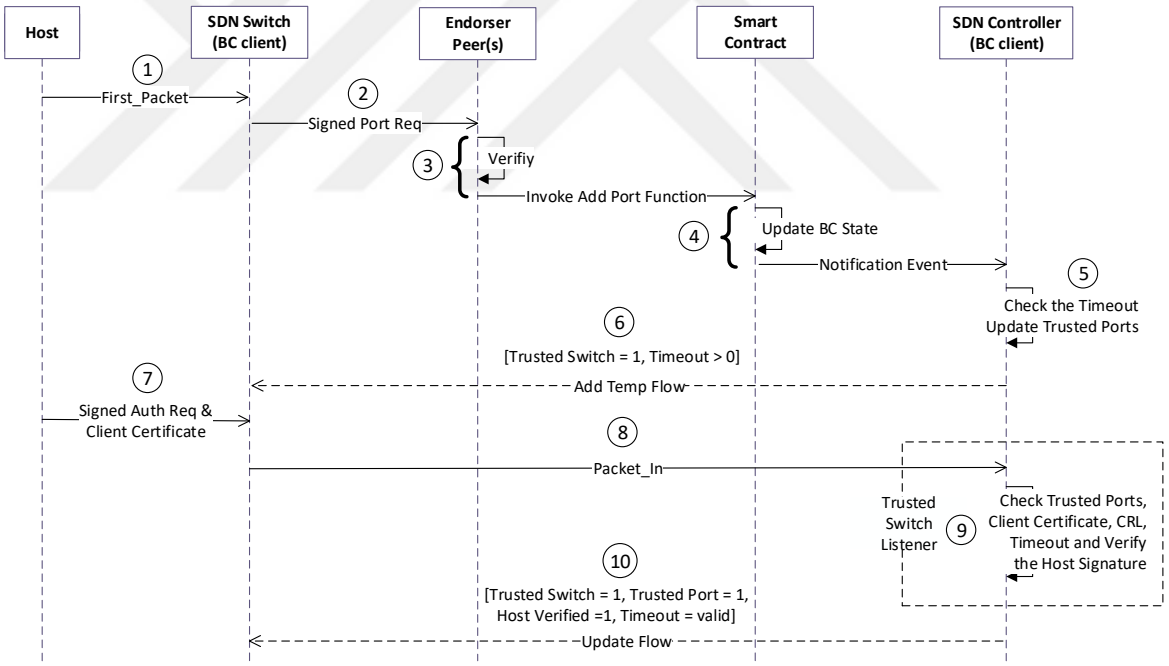


Figure 7: Host authentication in DPSEC © 2020 IEEE [1].

Intra-domain Host Authentication Steps

Step (1) Host $h_{i,k}$ sends its first packet to switch $s_{i,j} \in S_i$.

Step (2) Once $s_{i,j}$ receives the packet from $h_{i,k}$, it will send a new signed $Port\_Req$ to the corresponding endorsing peer(s) $e \in E_{(i)accept}$.

Step (3) Each endorsing node $e \in E_{(i)accept}$ verifies $Cert_{(sw)i,j}$ and its signature $Sig_{(sw)i,j}$ and invokes the corresponding smart contract, which includes *Add Port Function* (see Algorithm 2).

Step (4) Upon acceptance, each committing peer $v \in V$ updates its replica and controller $C_i \in C_{auth}$ will be notified.

Step (5) C_i will check $Timeout$ and add a new entry to its $Trusted\_Ports$ table, which includes the MAC address and the corresponding InPort.

Step (6) C_i adds a temporary flow to switch $s_{i,j}$.

Step (7) $h_{i,j}$ sends a new signed $Auth\_Req$ along with $Cert_{(h)i,j}$ to $s_{i,j}$.

Step (8) $s_{i,j}$ submits a new $Packet\_In$ message to C_i.

Step (9) C_i checks the $Trusted\_Ports$ table to determine whether the message is received from a *trusted* port/MAC pairs within specific time period. If there is no match for the packet received from the same port, then it will be removed from $Trusted\_Ports$ and also C_i will update the port record on BC to be *invalid*. Additionally, the MAC address needs to be matched with MAC attribute in $Cert_{(h)i,j}$ which also needs to be verified. C_i also checks Certificate Revocation List (CRL) to determine whether the certificate is revoked. Then, C_i needs to verify the message received from $h_{i,j}$.

Step (10) C_i sends a new $Flow\_Mod$ message to $s_{i,j}$ in order to provision a flow entry that includes Port:MAC:IP information to match against the incoming traffic.

Algorithm 2: Add Port Function [1]

```
Input: caller, DPID, MAC, InPortID
caller BC client;
if caller.role == client then
    if caller.state == Trusted then
        if caller.dpid == DPID then
            if MAC does not exist as a valid record then
                Add a new record (InPortID,MAC,DPID,D_i);
                Emit a new event to notify C_i ⊆ C_Auth that caller.dpid added a
                  new port record (InPortID,MAC,D_i);
            else
                Reject the request;
                Update h_{i,j} state to Untrusted;
                Emit a new event to notify C_i about h_{i,j} status change;
            end
        else
            Reject the request;
            Update caller state to Untrusted;
            Emit a new event to notify C_i about caller.dpid status change;
        end
    else
        Reject the request;
    end
else
    Reject the request;
end
```

2.4.2.6 Cross-domain Host Authentication

SDN hosts that request to connect to d different domains $D_{auth} = \{D_i, ..., D_j\}$, $D_{Auth} \subseteq D$ must be endorsed by $E_{(i)} \cup ... \cup E_{(j)}$. Thus, we can define the endorsement policy with minimum number of peers as $E_{(i,...,j)accept} = E_{(i)accept} \cup ... \cup E_{(j)accept}$ where $|E_{(i,...,j)accept}| = |E_{(i)accept}| + ... + |E_{(j)accept}| = 2(\sum_{y \in D_{auth}} f_y) + d$.

2.4.3 Experimental Results

In this section, we study the applicability and feasibility of our protocol. We extended the LINC-Switch [61] to interact with BC. We chose LINC in our PoC due to its

modular and flexible architecture. As an SDN controller, we use Ryu [62] with OpenFlow1.3 on Ubuntu 14.04 system, installed on VirtualBox with a system equipped with 8GB RAM and intel i7-8550u processor, where we allocated one virtual CPU for our virtual machine. We used Hyperledger Fabric 1.4 [63] for our BC part. Our consortium BC part includes 2 organization and consists of 4 peers in total, 2 peers for each. For simplicity, we focus on intra-domain authentication and assume that the number of faulty nodes $f1 = f2 = 0$. We used Mininet to emulate our SDN network. We compared the latency between SDN and DPSEC. In traditional SDN Packet-In messages are submitted directly to the SDN controller whereas in DPSEC these messages are firstly submitted to and verified by the BC, then temporary flows are added to the corresponding switch by our controller according to the Host-Auth step in our protocol. We measured the latency for each step separately, then we reported the obtained results for latency metric. Figure 8 depicts the topology used in our PoC where we gradually increased the number of SDN switches ($k = 1, 5, 10, 20, 30$) and also hosts connected to the SDN network ($i = 1, 5, 10, 20, 30$).

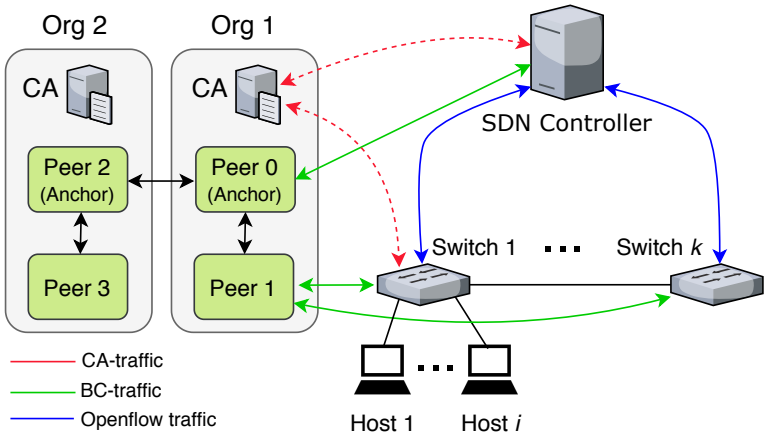


Figure 8: Topology used in our PoC © 2020 IEEE [1].

2.4.3.1 Switch-Auth Latency

Figure 9 shows that DPSEC has higher latency when compared with traditional OpenFlow handshake. The latency in DPSEC is due to the fact that each new

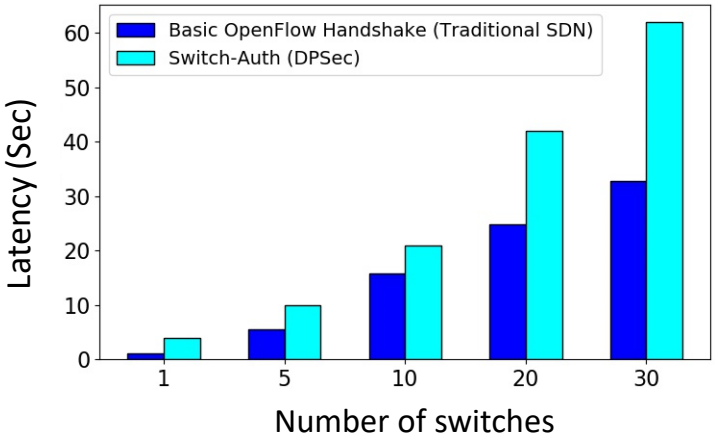


Figure 9: Comparing latency of basic OpenFlow handshake in traditional SDN and switch authentication in DPSEC © 2020 IEEE [1].

Join_Request must be verified and accepted by $2f_i + 1$ peer within the corresponding SDN domain D_i and committed by m peer in our BC, which allows fault-tolerant, decentralized and secure authentication. Moreover, since handling of *Join_Request* messages is offloaded to the smart contract, the overall load on the SDN controller is reduced compared with other approaches [48] that require analyzing of handshake messages exchanged between the SDN controller and underlying SDN switches. DPSEC provides also more flexible and secure cross-domain authentication compared with [46], which utilizes ACLs to restrict access in cross-domain networks.

2.4.3.2 Host-Auth Latency

To measure the Host-Auth latency, first we measured the time required to receive the first packet and update the *Trusted_Ports* table using *Port_Req*, which also requires updating the BC and notifying the controller. Then, we measured the time required to send and verify the *Auth_Req*.

As shown in Figure 10, the time required to complete *Port_Req* is higher than the time required to finish *Auth_Req* since *Port_Req* takes 0.94 seconds per transaction on average, whereas *Auth_Req* takes 0.5 seconds per host on average. The overall latency is 1.44 second per host on average. The major latency in host-auth is due

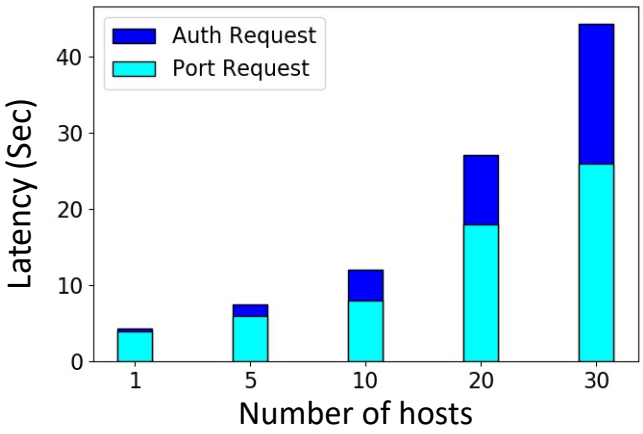


Figure 10: Latency of host authentication in DPSEC © 2020 IEEE [1].

to the fact each new *Port_Req* must be verified and accepted by at least $2f_i + 1$ peer within the corresponding SDN domain D_i and committed by m peer in our BC. However, this enables fault-tolerant, decentralized and secure cross-domain authentication based on unique blockchain characteristics. Unlike [35, 26] which focused only on host authentication, DPSEC ensures that each host is connected to a trusted switch beforehand.

2.4.4 Internal Attacks

In this subsection, we present a security analysis of internal attacks in DPSEC. These attacks, are launched from legitimate SDN data plane, where the attacker may try to gain unauthorized access on other existing components within the SDN network.

2.4.4.1 Unauthorized Flows

Since each flow is added only by trusted switches and our smart contract also checks the DPID for each trusted switch, it is not possible to add unauthorized flows. Therefore, a new flow is only added when there is a match between the caller DPID and submitted DPID.

2.4.4.2 Switch Spoofing

It is not possible to spoof SDN switches in DPSEC since each switch receives BC_Nonce from our smart contract for each time it needs to join the SDN network. Additionally, there must be a match between the caller DPID and submitted DPID at the smart contract side and also a match between DPID and submitted BC_Nonce at the controller side.

2.4.4.3 MAC Spoofing

Since our smart contract refuses to add existing valid MAC addresses, internal adversaries will be detected since spoofed MAC addresses will not be matched with $Trusted\_Ports$ entries. Additionally, its not possible to spoof any MAC address even if is not added to the BC, since $Auth\_Req$ needs to be verified through host's Pub_K along with its certificate obtained from CA_i, which also includes host's original MAC address. Therefore, MAC spoofing will also be detected by the SDN controller since the submitted MAC does not match with the MAC attribute in its certificate.

2.4.5 External Attacks

In this subsection, we present a security analysis of external attacks in DPSEC. These attacks can be launched from malicious SDN data plane to exploit or gain unauthorized access to an existing SDN network.

2.4.5.1 DoS Attacks

Our protocol significantly reduces the DoS launched from malicious SDN data plane since it authenticates SDN switches at the handshake stage. Therefore, the SDN controller will not receive events from untrusted switches. Whereas connection attempt from malicious external hosts are detected since their $Auth\_Req$ cannot be verified. Such an attack will be detected by the SDN controller at the handshake stage.

2.4.5.2 Distributed DoS Attacks

These attacks may affect the SDN network due to large number of temporary flows. The controller can monitor the switch under attack and adds flows only for verified host. Thus, avoids adding temporary flows. Additionally, DPSEC can eliminate malicious traffic generated using spoofed MAC addresses, which will reduce the overall impact of most or all of the reflection attacks.

2.4.5.3 Switch Spoofing

It is also not possible to spoof SDN switches since we enforce attribute based access control through the certificate authority and also we utilize BC_Nonce to secure the handshake stage, therefore fake switches will be detected at the handshake stage.

2.4.5.4 MAC Spoofing

It is not possible to spoof the MAC addresses in DPSEC since the packet must be received from a trusted port. Also, there must be a match between the MAC address and incoming port, otherwise it will be removed from the $Trusted\_Ports$ table. Furthermore, each $Auth\_Req$ needs to be verified through the host's Pub_K. MAC spoofing will be detected by the SDN controller when the $Packet\_In$ message is received.

2.4.6 Computational Cost

The computational cost in DPSEC can be analysed as follows. For switch authentication, each switch needs to sign $Join\_Request$ two times, since Hyperledger Fabric splits each transaction into 2 steps: 1) $Join\_Request$ proposal and 2) endorsed $Join\_Request$. This also applies for $Port\_Request$. Additionally, for intra-domain authentication $2f_i + 1$ endorsing peer needs to verify received switch certificate and signature to authenticate their communicating party and then sign the received proposal. Whereas for cross-domain authentication $2f_i + 2f_j + 2$ endorsing peers

need to verify received switch identity and then sign the proposal. Finally, each committing peer $v \in V$ needs to verify the endorsement policy of each accepted transaction. Thus, the computational cost for intra-domain switch authentication is $O((2f_i + 3)s + (2f_i + |V| + 1)v)$ where s and v denote signature generation and verification, respectively. Whereas for cross-domain switch authentication the computational cost is $O((2f_i + 2f_j + 4)s + (2f_i + 2f_j + |V| + 2)v)$. The computational cost for intra-domain host authentication is $O((2f_i + 3)s + (2f_i + |V| + 3)v)$ since the SDN controller has to verify host certificate and its signature. The computational cost for cross-domain host authentication on d different domains is $O((2(\sum_{y \in D_{auth}} f_y) + d + 2)s + (2(\sum_{y \in D_{auth}} f_y) + |V| + d + 3)v)$. In addition to the overhead produced by encryption-decryption operations due to the utilization of the SSL/TLS protocol.

2.4.7 Communication Overhead

In intra-domain settings, SDN switches have to submit signed proposal to $2f_i + 1$ endorsing peer. Whereas in cross-domain settings, they need to submit $2f_i + 2f_j + 2$ or $2(\sum_{y \in D_{auth}} f_y) + d$ transaction proposal for switch and host authentication respectively. Each proposal consists of 1096 bytes that contains header and payload. In addition to between 70-72 bytes for Elliptic Curve Digital Signature Algorithm (ECDSA) [64] signature of the corresponding switch. Endorsed transactions are submitted to $|V|$ committing peer. *Join_Request* and *Port_Request* endorsed by 1-peer are 3071, 3261-bytes, respectively. As shown in Figure 11, the size of endorsed transactions increases according to the number of endorsing peers. For host authentication the host needs also to send a packet that contains 70-72 bytes for host signature and 1500 bytes for host certificate. Different network protocol headers are also included. 20-byte header for IP, and 20-byte header for TCP with no options. Utilization SSL/TLS requires 7x4-bytes for 7-handshake headers, 4x5-bytes for 4 TLS records, 170-bytes for ClientHello. 32-bytes for SessionID. 75-bytes for ServerHello. 4x1500-bytes for

host certificate (assuming 4 certificates are used). 130-bytes for ClientKeyExchange. 2x1-bytes for 2-ChangeCipherSpec. 2x12-bytes for 2-Finished messages. Thus, 7x4 + 4x5 + 170 + 32 + 75 + 4x1500 + 130 + 2x1 + 2x12 = 6481 bytes per SSL/TLS handshake. However, when session resumption is enabled SSL/TLS overhead reduces to 4x4 + 3x5 + 170 + 32 + 75 + 2x1 + 2x12 = 334 bytes.

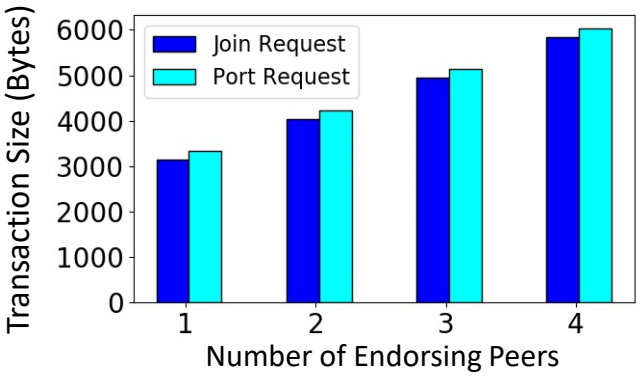


Figure 11: Transaction size according to involved endorsing peers © 2020 IEEE [1]

CHAPTER III

CWT-DPA

In this chapter, we enhance the performance of BC-SDN data plane approaches by separating switch-related and host-related transactions and propose a different waiting strategy for each type of data plane transaction. We also propose a batching technique to enhance the average latency for high-load scenarios. In addition, SDN switches and controllers utilize lattice-based signatures and Key Encapsulation Methods (KEMs) to protect against quantum adversaries. We compare the performance of CWT-DPA with existing solutions such as SSL/TLSv1.3 [65], DPSEC [1], and AuthFlow [7] (This study is published in [3]).

3.1 Limitations of DPSEC [1]

DPSEC [1] did not differentiate between host and switch transactions, resulting in performance degradation. In this chapter, we focused on improving the performance of DPSEC [1] by defining a different strategy for switch and host transactions. Existing BC-SDN models, such as our previous work DPSEC [1], suffer from high latency and security weaknesses against quantum adversaries. Consequently, these approaches cannot be used in realistic scenarios (i.e., under high loads or potential quantum attacks). Therefore, there is a need for new BC-SDN models that provide high performance and ensures protection against both classical and quantum adversaries. To this end, this chapter focuses on improving the performance of BC-SDN model and enhancing the security of BC-SDN's southbound API. Our contributions are threefold:

1. We propose CWT-DPA a BC-SDN model that separates switch-related transactions from the host-related transaction and chooses a different waiting strategy for each type of data plane BC transaction to enhance the performance of BC-SDN models. We also provide performance improvement based on batching mechanism for switches with high loads. CWT-DPA considers controller failure and chooses the highest trusted controller based on feedback received from SDN switches.

2. We utilize lattice-based signatures based on Falcon [66] and Dilithium [67] schemes and Key Encapsulation Methods (KEMs) based on NTRU [68] and Kyber [69] schemes to achieve a higher level of protection for the authentication process among switches and SDN controllers.

3. We conduct our experiments using OVS switch to measure the performance of the enhanced switch and host authentication methods. We compare the performance of CWT-DPA with existing solutions such as SSL/TLSv1.3 [65], DPSEC [1] and AuthFlow [7].

3.2 Attacks Launched From Malicious SDN's Data Plane

3.2.1 Attacks Against The SDN Controller

The SDN controller is the most critical component in the SDN network since SDN switches continuously need to communicate with the controller when no match is found in the corresponding flow table(s). Therefore, external and internal attacks against the controller may devastate the SDN network. As shown in Figure 12, spoofing attacks can be launched from malicious SDN switches by utilizing the DPID of the victim switch [70]. Moreover, SDN switches can generate fake Packet-In messages that can be utilized to initiate DoS/DDoS attacks on the SDN controller [71].

3.2.2 Attacks Against SDN Hosts

A malicious host may launch spoofing attacks on legitimate SDN hosts. Spoofed source addresses allow the attacker to hide his/her identity, and bypass source-based filtering during DoS/DDoS attacks [72].

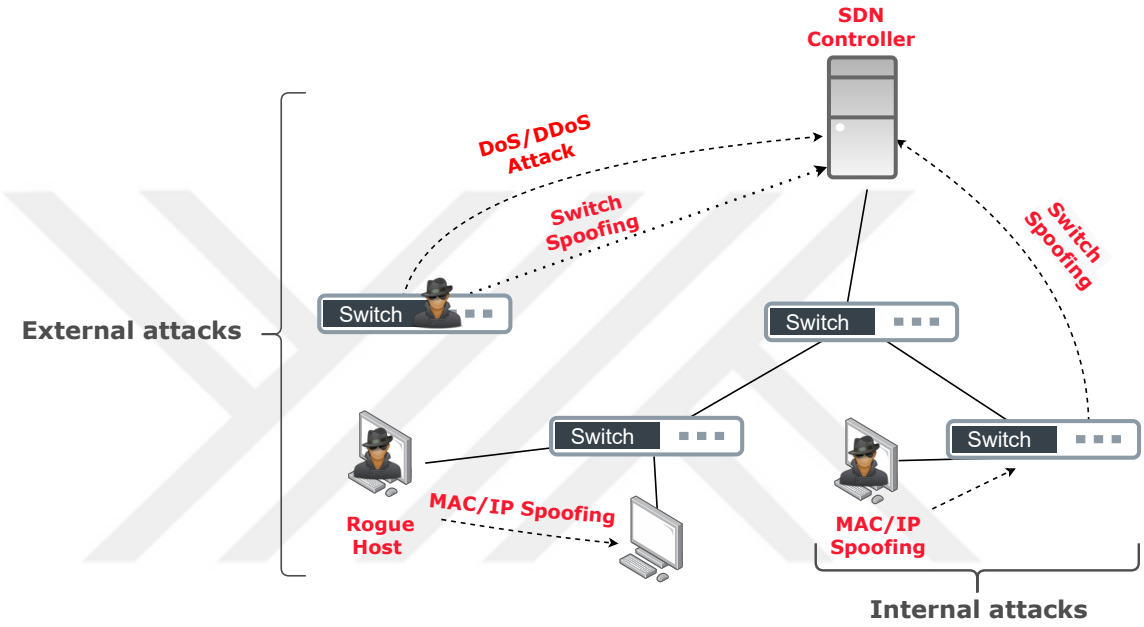
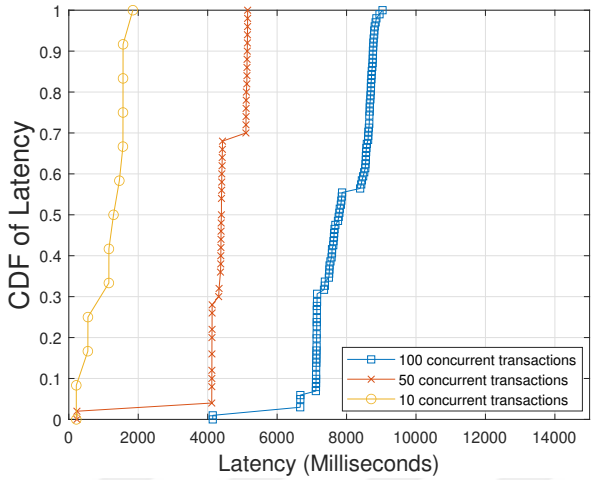


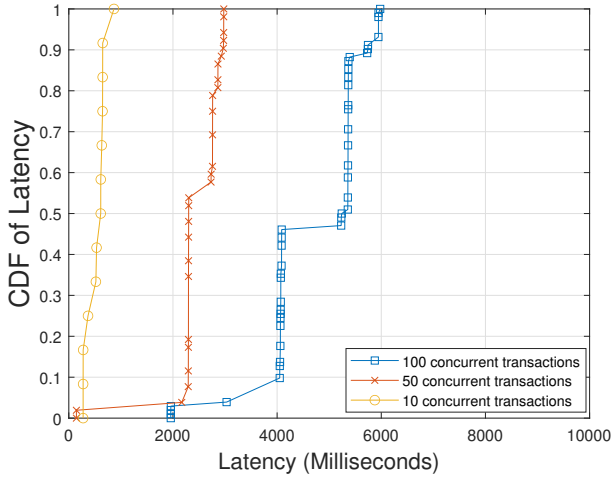
Figure 12: Attacks against the SDN network [3].

3.3 Latency of Waiting for Final Commitment and Its Effect in BC-enabled Networking

In the commit stage, BC peers send their final commitment to the corresponding BC client [73]. When the client receives the required commitments, it indicates that the submitted transaction is added to BC [73]. This also implies validation of the corresponding transaction before being added to the BC ledger. In BC-enabled networking, the latency of waiting for final commitment is a significant limitation for high performance, especially for the cases that include dependency between BC transactions. It also results in high latency in high load scenarios (i.e., receiving a high number of transactions simultaneously). Figure 13 depicts the Cumulative Distribution Function (CDF) of latency for concurrently adding a different number of



(a) Latency with waiting for final commitment



(b) Latency without waiting for final commitment

Figure 13: Comparing latency with/without waiting for final commitment [3].

transactions to BC. The latency decreases significantly when BC clients do not have to wait for the final commitment for all loads (10, 50, and 100 transactions).

Sending BC transactions asynchronously reduces the overall waiting time. However, in some scenarios, BC participates have to wait for certain transactions before they start sending their transactions. Our idea is to separate switch-related transactions from host-related transactions to enhance the performance of BC-SDN since

switch-related transactions are more trusted than host-related transactions. Switch-related transactions are more trusted since the smart contract verifies these transactions. Host-related transactions are less trusted since hosts can provide spoofed MAC addresses during the binding stage. The switch also needs to wait for the authentication results of the corresponding SDN host. Due to this property of binding-related host transactions, we use a different strategy for each type of data plane transaction.

3.4 Proposed CWT-DPA

In traditional BC-SDN approaches, such as our initial model (DPSEC [1]), the switch has to wait for the final commitment for each submitted transaction before contacting the SDN controller, which ultimately results in performance degradation, especially in case of dependencies with other SDN switches where the latency in one switch can affect the latency of remaining switches as well. The second problem is that there is no separation between switch and host transactions [1]. In this chapter, we propose CWT-DPA, which separates transactions in BC-SDN into 1) switch-related BC transactions and 2) host-related BC transactions, and propose a different waiting strategy for each case to enhance the performance of BC-SDN. Figure 14 shows our proposed model for BC-SDN, which sends switch-related transactions without waiting

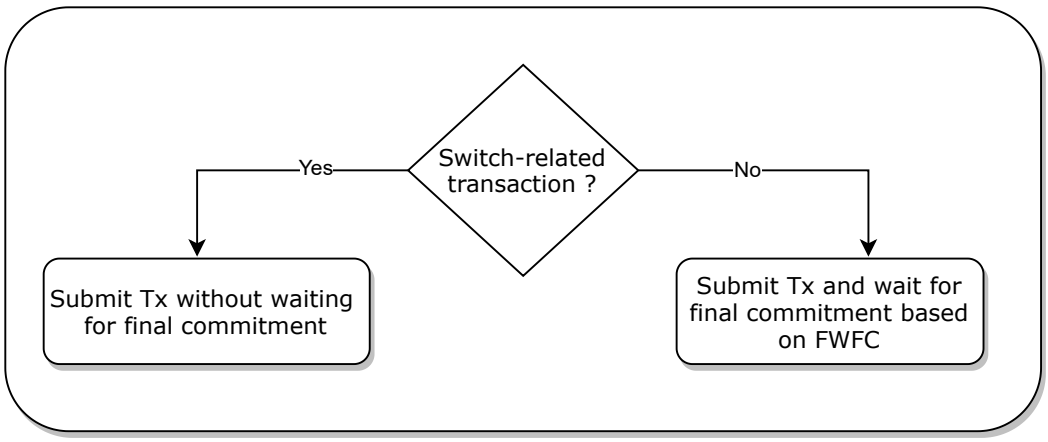


Figure 14: CWT-DPA concept [3].

for final commitment and waits only for a fraction of submitted transactions for host-related transactions based on the Fraction of Waiting of Final Commitment (FWFC) metric calculated for each switch during each epoch. As shown in Figure 15 CWT-DPA consists of four main components, which are:

- BC-Peers: provide decentralization for the authentication process in CWT-DPA. BC-Peers include the public keys and certificates of related SDN hosts, switches, and SDN controllers. BC-Peers verify and endorse transactions received from BC-gateway and SDN switches.

- BC-Gateway: receives authentication requests from SDN switches and sends authentication transactions to BC-Peers. BC-Gateway decreases the latency for switch authentication by reusing existing connections to BC-peers and submitting switch transactions without waiting for final commitment.

- CWT-DPA-enabled SDN Switch: sends switch authentication transactions to BC-gateway using CWT-DPA module. The switch also includes BC-listener to receive controller endorsements and encrypted authentication tickets from BC nodes. Host transactions are directly submitted using the BC-client component.

- CWT-DPA-enabled SDN Controller: responsible for authenticating SDN switches and SDN hosts. The controller also includes BC-listener to receive switch endorsements and the hash of authentication tickets from BC nodes. In addition, BC-client is used to interact with BC.

.

3.4.1 Switch-related Transactions

Switch transactions represent transactions submitted during the switch authentication process. In CWT-DPA, each switch sends its switch-authentication transaction without waiting for a final commitment since these transactions are critical for the

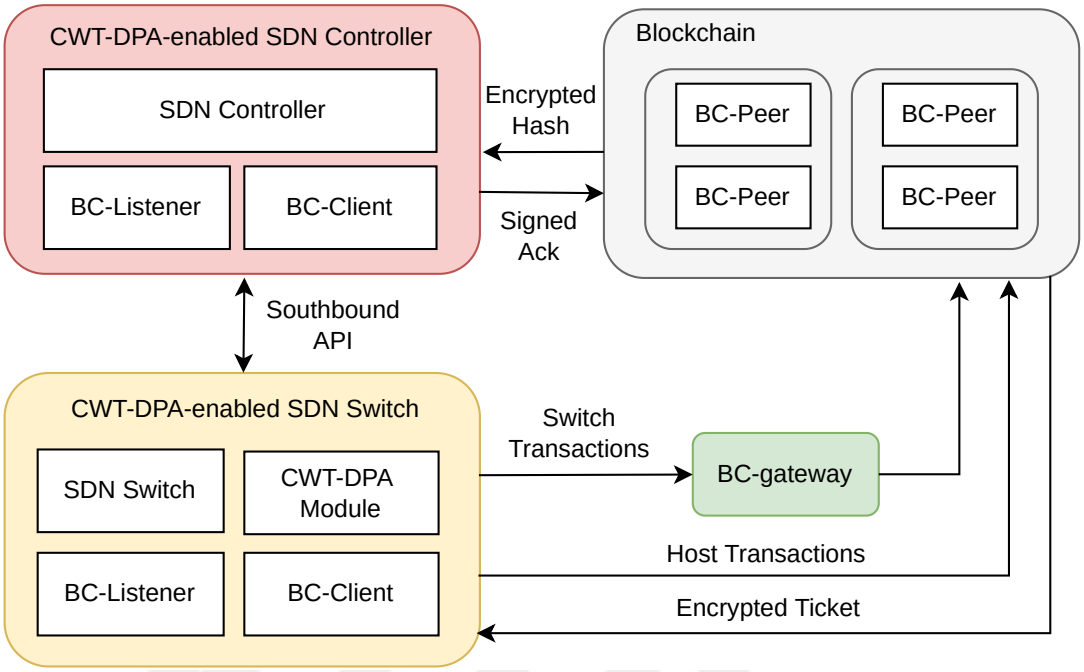


Figure 15: CWT-DPA components [3].

SDN network and may affect the performance of other related switches in the SDN network because of dependability between SDN switches. We add a BC-gateway for switch authentication to solve the dependability issues and reuse the BC connection. As shown in Figure 16, the gateway resolves the dependencies (if they exist) between SDN switches and informs the corresponding controller upon receiving and endorsing switch requests.

We update the contract of DPSEC [1] by adding a new function called $Fast\_Join\_Request$. When the request is accepted, BC peer sends a ticket encrypted with the public key of the corresponding switch. The gateway cannot use the ticket since it is already

BC-Function	Participant	Function Parameters
Add Switch	Admin	PQ Cert, DPID, PQ Public Keys, PQ Sig(DPID||PQ Public Keys)
Add Controller	Admin	PQ Cert, ContID, PQ Public Keys, PQ Sig(ContID||PQ Public Keys)
Add Gateway	Admin	PQ Cert, GwID, Public Key, PQ Sig(GwID||Public Key)
Add Host	Admin	PQ Cert, HostID, Public Key, PQ Sig(HostID||Public Key)

Table 1: Main registration functions in CWT-DPA [3].

encrypted using the switch's public key. The ticket is then decrypted by the corresponding switch and used in the authentication process. Both SDN switches and controllers utilize lattice-based signatures and Key Encapsulation Methods (KEM) to provide quantum-safe authentication and key exchange. The public keys of SDN controllers, switches, and gateways are added to smart contracts by sending a new registration transaction from the corresponding domain administrator, which has a certificate issued by the certificate authority of that corresponding domain. The certificate and each registration transaction include PQ signatures of the corresponding certificate authority and the domain administrator, respectively. Accepted registration transactions are broadcasted to SDN components through BC events. Registration functions in CWT-DPA are shown in Table 1. A high-level overview of our proposed switch authentication is shown in Figure 16.

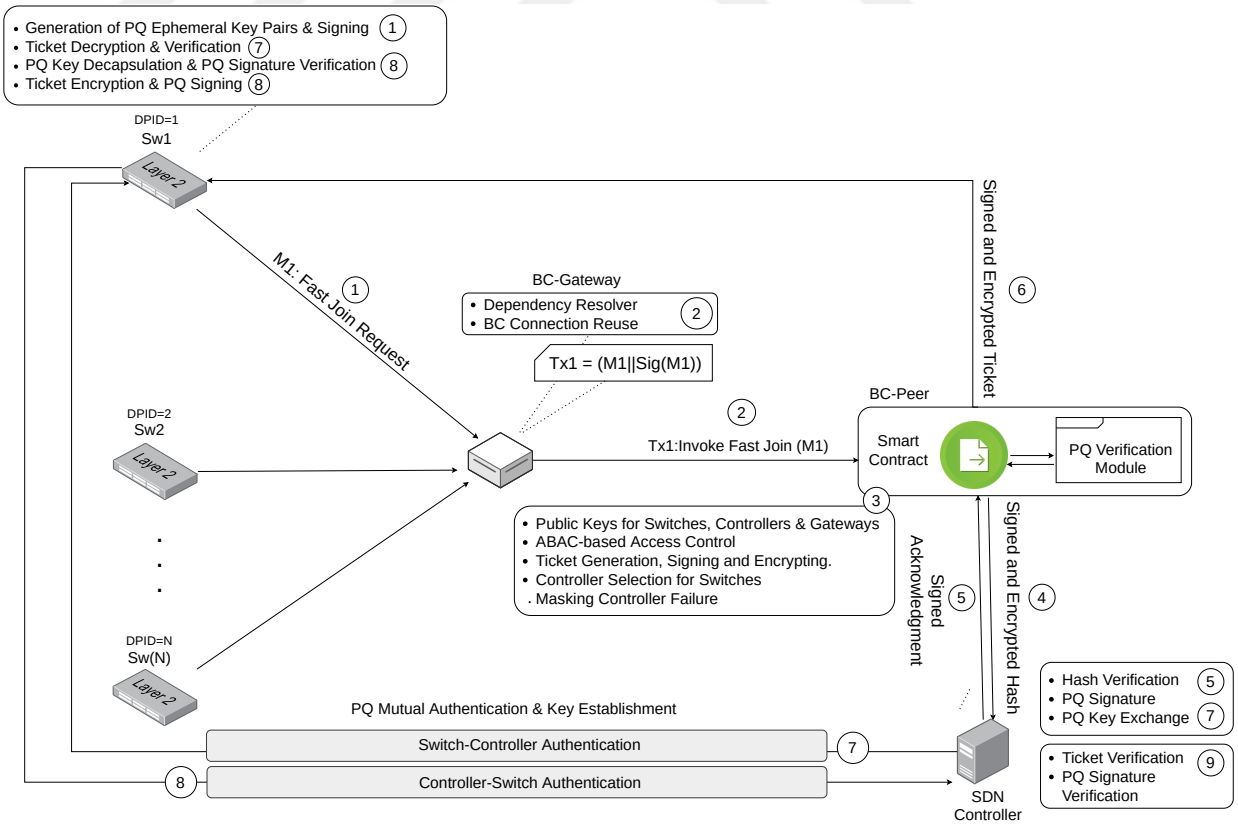


Figure 16: Overview of PQ fast BC-enabled mutual switch-controller authentication in CWT-DPA [3].

In step (1), the switch Sw_1 sends M_1 message to the BC-gateway, which consists of the switch's public key (Sw_1_PubK), Nonce, and newly generated post-quantum (PQ) ephemeral public key (Sw_1_EPubK) signed by the private key of the corresponding SDN switch (Sw_1_Sk):

M_1 (Switch → BC-Gateway)

$$M_1 = Sw_iNonce||Sw_i\_Pub_k||Sw_i\_EPub_K||PQSig(Sw_iNonce||EPub_K)$$

In step (2), BC-gateway calls $Fast\_Join\_Request$ function of the authentication contract using switch message (M_1) by sending transaction Tx_i

Tx_1 (BC-Gateway → BC)

$$Tx_i = GwNonce||M_1||Sig(GwNonce||M_1)$$

BC-gateway has two operation modes, namely, relay mode and defense mode. In the relay mode, BC gateway will not verify the requests submitted from SDN switches. In the defense mode, the BC gateway can verify switch authentication requests. The gateway also calculates the rejection rate of switch-related transactions of the corresponding switch and rejects switch-related transactions if the transaction rate passes a predefined threshold. Algorithm 3 shows the initial authentication step at the BC-gateway side.

In step (3), the contract verifies M_1 and then generates the authentication ticket. At this stage, the contract chooses the SDN controller with the highest trust value, where we describe how we calculate trust values in Section 5.1.1. Then, the hash is signed and encrypted ($M2.Controller$) by the corresponding BC peer. In step (4), $M2.Controller$ is sent to the corresponding SDN controller.

Algorithm 3: BC-Gateway [3]

```
Recv(M1);
Generate GwNonce;
if Recv(M1) and GW.Mode == Relay then
    Tx_i = GwNonce||M1||Sig(GwNonce||M1);
    Send(Tx_i);
end
if Recv(M1) and GW.Mode == Defense then
    GetSw_iPub_K(DPID);
    if (PQ Verify(Sig(Sw_iNonce||Sw_i_EPubK)) == Success) then
        if Transaction Rate < Threshold then
            Tx_i = GwNonce||M1||Sig(GwNonce||M1);
            Send(Tx_i);
        else
            Reject the request;
        end
    else
        Reject the request;
    end
end
```

In step (5), the SDN controller decrypts and verifies the hash, then acknowledges the receipt of $M2.Controller$ by signing the challenge received from the BC contract.

In step (6), the corresponding BC peer verifies the acknowledgment, then encrypts the ticket and sends it to Sw_1 using $M2.Switch$ message. If the controller fails to send its acknowledgment, the contract selects the controller with the second highest trust value, and the authentication process will continue from this point. This shows how CWT-DPA masks controller failure based on BC smart contracts during the authentication stage. Algorithm 4 shows the initial authentication step at the BC-peer side.

In step (7), Sw_1 decrypts and verifies the ticket. At the controller side, the controller verifies the ephemeral key using the PQ signature of the corresponding switch, then generates a new nonce value, a new secret, and uses PQ KEM for secret key encapsulation. The ciphertext, switch, and controller nonces are signed and sent to the corresponding switch using the M_3 message.

Algorithm 4: BC-Peer [3]

$Recv(M_1)$;
$GetSw_iPub_K(DPID)$;
if $(PQ\ Verify(Sig(Sw_iNonce||Sw_i\_EPubK))\ ==\ Success)$ **then**
  $Generate\ Ticket\ T_i, ContChallenge\ C_i$;
  $Get\ Controller(Cont\_Pub_K)$;
  $B_0 = Sw_iNonce||Sw_iPub_K||Sw_iEPub_K||Cont\_Pub_K||C_i$;
  $SC_0 = Sig(H(T_i)||H(B_0))$;
  $M_2.Controller = Sw_iNonce||Sw_iPub_K||Sw_iEPub_K||C_i||Enc(H(T_i))||SC_0||A_0$;
  $Send(M_2.Controller)$;
  $Recv(Cont_{ACK})$;
  if $(PQVerify(Sig(Cont_{ACK}))\ ==\ Success)$ **then**
    $M_2.Switch = Cont\_Pub_K||C_i||Enc(T_i)||SC_0$;
    $Send(M_2.Switch)$;
  end
end

$M3$ (Controller → Switch)

$$D_1 = PQSig(H(ContNonce||Sw_iNonce||PQCihpertext))$$

$$M_3 = ContNonce||PQCiphertext||D_1$$

In step (8), after ensuring that controller's public key is registered and pre-verified by admin transaction, CA_Cert and verifying the signature of the corresponding controller, Sw_1 signs the ticket, and the controller nonce using the PQ signing algorithm and sends the ticket along with its signature encrypted with the secret key received from the corresponding SDN controller using M_4 message.

M_4 (Switch → Controller)

$$M_4 = Enc(PQ\_Shared\_Secret, T_i||PQSig(T_i||ContNonce))$$

In step (9), the controller verifies the corresponding ticket and the required signature using the PQ signature verification algorithm and the hash received from the BC contract and accepts/rejects the connection accordingly. Algorithm 5 shows the main authentication steps on the SDN controller side. The switch receives a success message from the SDN controller, which completes the authentication process.

Algorithm 5: SDN Controller [3]

```
if Recv(M2.Controller) then
    Dec(M2);
    Verify(Sw1Nonce||Sw1PubK||H(T[Swi])||Ci);
    Cont_ACK = PQSign(Ci);
    Send(Cont_ACK) ;
    Total Endorsements[Swi] = Total Endorsements[Swi] + 1;
    if Total Endorsements[Swi] >= Required Endorsements then
        PQVerify(Sig(SwiNonce||Swi_EPubK)) ;
        Generate ContNonce;
        PQCihpertext, PQSharedSecret = PQEncapsulate(Swi_EPubK);
        D1 = PQSign(H(ContNonce||SwiNonce||PQCihpertext));
        M3 = PQCiphertext||D1||ContNonce;
        State = M3;
        Send(M3) ;
    end
end
if Recv(M4) and State = M3 then
    Dec(M4);
    Verify(H(T'[Swi]));
    PQVerify(Sig(T[Swi]||ContNonce));
    State = Success;
    Send(Success);
end
```

Algorithm 6 shows the main authentication step at the SDN switch side. In addition, a detailed description of our proposed authentication is shown in Figure 17. Our design utilizes BC gateway to resolve any potential dependencies among SDN switches and to decrease the latency of our switch authentication scheme by reusing the same connection to BC peers.

Algorithm 6: SDN Switch [3]

Generate Sw_iNonce ;
Generate Post Quantum (PQ) Ephemeral Key Pairs ($Sw_i\_EPub_k, Sw_i\_ES_k$) ;
$A_0 = PQSig(Sw_iNonce||Sw_i\_EPub_K)$;
$M_1 = Sw_iNonce||Sw_i\_Pub_k||Sw_i\_EPub_K||(A_0)$;
$State = M_1$;
$Send(M_1)$;
if $Recv(M_2)$ *and* $State = M_1$ **then**
    $Dec(M_2)$
    $Verify(Sig(H(T_i)||H(Sw_iNonce||Sw_iPub_K||Sw_iEPub_K||Cont_i\_Pub_K)))$;
    $Total\ Endorsements[Cont_i] = Total\ Endorsements[Cont_i] + 1$;
    if $Total\ Endorsements[Cont_i] >= Required\ Endorsements$ **then**
        $State = M_2$;
    end
end
if $Recv(M_3)$ *and* $State = M_2$ *and* $Pre\_verified(Cont_i\_Pub_K, AdminTx,$ *CA_Cert)* **then**
    $Verify(M_3)$;
    $PQ\_Shared\_Secret = PQDecapsulate(PQCiphertext, Sw_i\_ES_k)$;
    $M_4 = Enc(PQ\_Shared\_Secret, T_i||PQSign(T_i||ContNonce))$;
    $State = M_4$;
    $Send(M_4)$;
end
if $Recv(Success)$ *and* $State = M_4$ **then**
    $State =\ Success$;
end

On the other hand, it brings an acceptable level of centralization to the switch authentication process since the gateway is responsible only for invoking the required function in the corresponding smart contract, which is responsible for sending the results to SDN controllers and switches. Therefore, the gateway functionality is restricted to calling the corresponding smart contract. Whereas the contract itself autonomously does the remaining part. It is worth noting that the switch can detect malicious gateways based on BC transactions and the events and notifications of the corresponding contract.

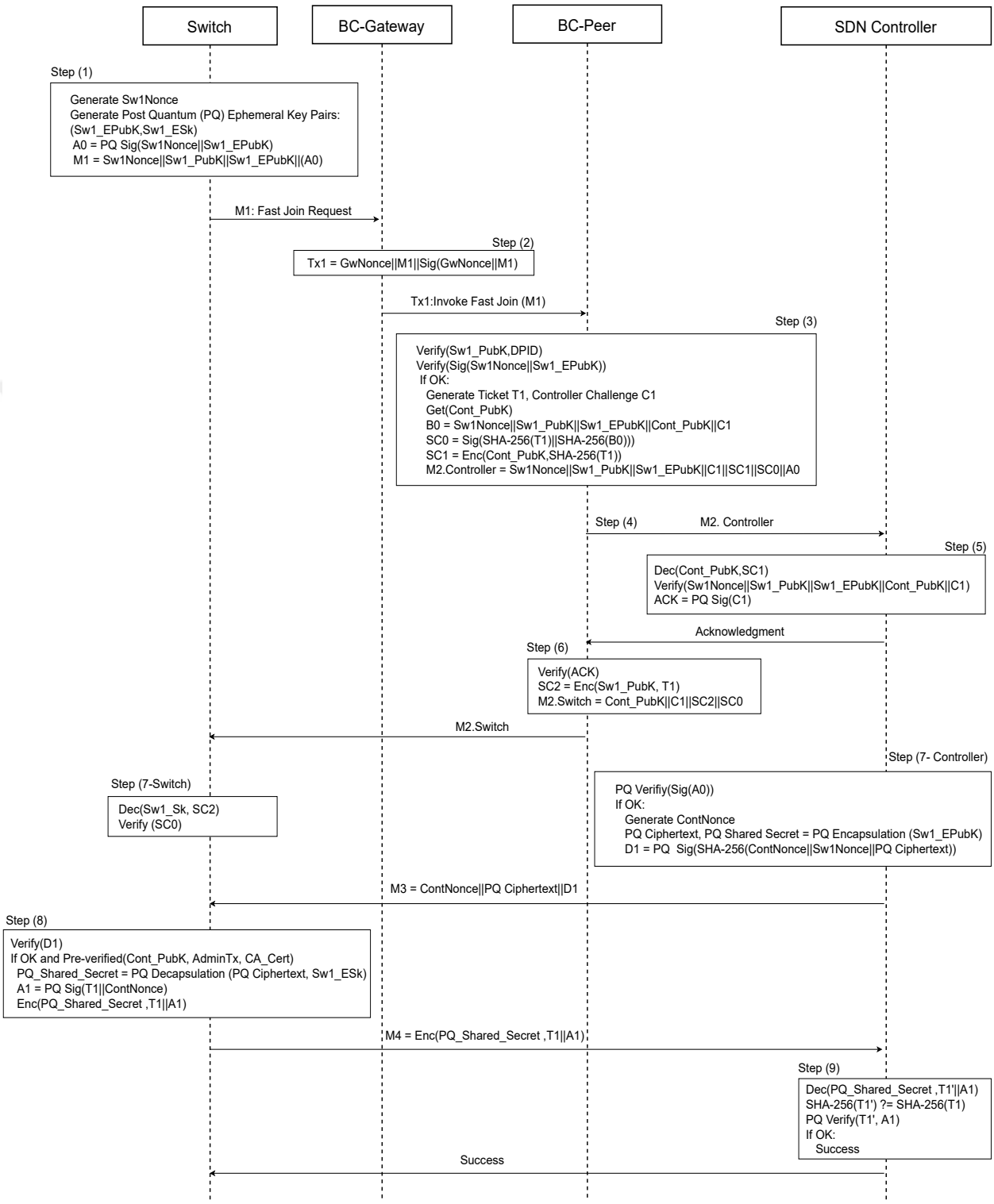


Figure 17: Detailed steps of fast BC-enabled mutual switch-controller authentication in CWT-DPA [3].

Calculation of Trust Scores for SDN Controllers

Inspired by [5, 74], trust scores are calculated based on feedback received from each SDN switch according to the Gompertz function:

$$TS_{i,j} = ae^{-be^{-cFB_{i,j-1}}} \tag{1}$$

where a, b and c are function parameters. a is the upper asymptote, b indicates the displacement along the x-axis, c controls the growth rate [5]. $FB_{i,j-1}$ represents the total feedback obtained from SDN switches for controller i from epoch 1 until epoch $j-1$ is calculated based on a modified version of [74]:

$$FB_{i,j-1} = \sum_{n=1}^{j-1} \sum_{k \in Sw_{i,n}} ln|Sw_{i,n}|FB_{k,i,n} \tag{2}$$

where $Sw_{i,n}$ is the set of SDN switches authenticated and managed by SDN controller i during epoch n. $|Sw_{i,n}|$ is the cardinality of the set $Sw_{i,n}$.

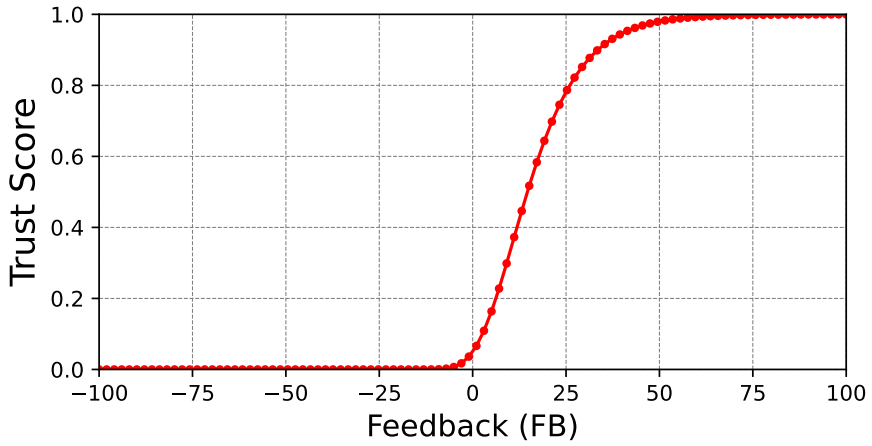


Figure 18: Feedback vs trust score where a= 1, b=3, c=0.1 (This figure is adapted from [5] © 2014 with permission from Elsevier[1]).

[1]Reprinted from Ad Hoc Networks, Vol. 12, K. L. Huang, S. S. Kanhere, and W. Hu, On the need for a reputation system in mobile phone based sensing, 130-149, Copyright (2014), with permission from Elsevier.

We multiply $FB_{i,j-1}$ by $ln|Sw_{i,n}|$ to prevent the controller to have a high trust value by authenticating a small number of SDN switches [74]. Figure 18, shows how trust scores change according to the obtained feedback (FB), which is used as an input for our Gompertz function. Each SDN switch sends feedback regarding successful/failed authentication by the corresponding SDN controller. Similar to [74], $FB_{i,j}$ can be computed as follows:

$$\begin{aligned} FB_{i,j} &= \sum_{n=1}^{j} \sum_{k \in Sw_{i,n}} ln|Sw_{i,n}|FB_{k,i,n} \\ &= \sum_{n=1}^{j-1} \sum_{k \in Sw_{i,n}} ln|Sw_{i,n}|FB_{k,i,n} + \sum_{k \in Sw_{i,j}} ln|Sw_{i,j}|FB_{k,i,j} \\ &= FB_{i,j-1} + ln|Sw_{i,j}| \sum_{k \in Sw_{i,j}} FB_{k,i,j} \end{aligned} \tag{3}$$

3.4.2 Formal Security Verification Using AVISPA Tool

AVISPA [6] is an automated tool mainly used to analyze and validate security protocols. AVISPA uses High-Level Protocol Specification Language (HLPSL) for specifying security protocols. HLPSL code is translated into equivalent intermediate format specifications, which are given as input to AVISPA backends that implement different analysis techniques. AVISPA includes the following backends. (1) On-the-fly Model-Checker (OFMC), (2) Constraint-Logic-based Attack Searcher (CL-AtSe), (3) SAT-based Model-Checker (SATMC), and (4) Tree Automata based on Automatic Approximations for the Analysis of Security Protocols (TA4SP) backend. Our simulation found that both SATMC and TA4SP did not support our protocol. Therefore, we focused on OFMC and CL-AtSe backends. Figure 19 shows the simulation of CWT-DPA authentication using OFMC and CL-AtSe backends. The results demonstrated that CWT-DPA is safe under OFMC and CL-AtSe backends against active and passive attacks, which include replay and man-in-the-middle attacks.

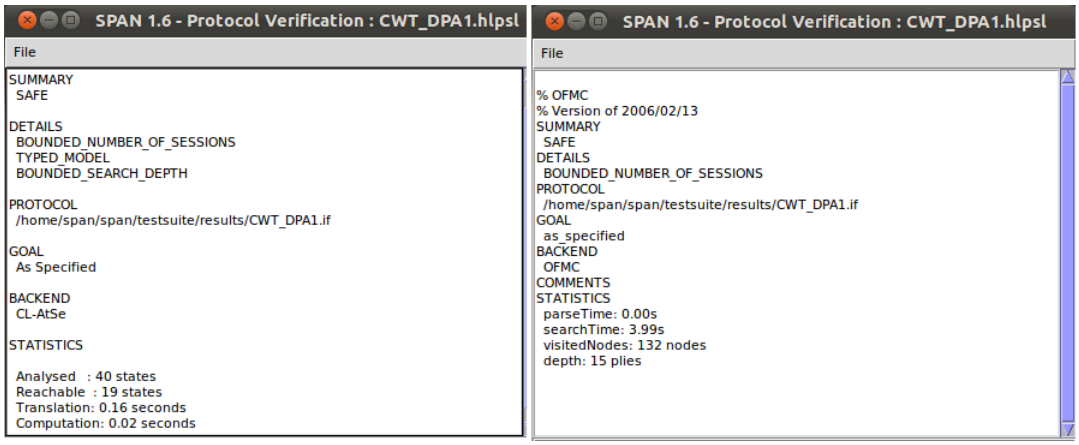


Figure 19: Verification results using AVISPA tool [6] under OFMC and CL-AtSe backends [3].

3.4.3 Host-related Transactions

3.4.3.1 Enhanced Waiting Time

Host transactions refer to transactions submitted when port bindings are changed. This occurs in two cases: 1) a new host is attached to the corresponding switch or 2) an old host is attached to a different port. Consequently, a new transaction is sent only when the binding is changed. The corresponding switch directly submits host-related transactions without utilizing our proposed gateway. These transactions are less trusted and therefore require a level of confirmation for each epoch. Our solution for host-related transactions is that the switch has to wait only for a fraction of the submitted transactions during each epoch. Since each switch has different loads and performs uniquely, we calculate the Fraction of Waiting for Final Commitment (FWFC) for each SDN switch based on the rejection rate (RR) and severity metric. Rejection rate (RR) for switch i during epoch j is calculated as follows.

$$Rejection\ Rate(RR_{i,j}) = \frac{\sum Rejected\_Transactions_{i,j}}{\sum Submitted\_Transaction_{i,j}} \tag{4}$$

When a switch sends more transactions than expected, more transactions need to be waited until the final commitment is completed, and therefore $FWFC$ should

change (i.e., increase) as well. We introduce the severity metric to address this issue. This metric deals with the case in which the switch changes its average transaction rate. Whether this change is due to high load or compromised cases, the waiting time has to be proportional to the average number of submitted transactions. Also, for security reasons, the switch has to wait for additional time to give BC peers and SDN controllers a chance to respond and perform appropriate actions without affecting the performance of other SDN switches or BC peers in general. Accordingly, the severity metric increases when the switch sends many requests higher than its overall average, which, in turn, leads to an increment of $FWFC$ as well. Conversely, this metric decreases when the switch sends several transactions less than its average, which also decreases $FWFC$ for that particular switch. The severity for switch i during epoch j is calculated as follows:

$$Severity_{i,j} = \frac{\sum Submitted\_Transactions_{i,j}}{Average\_Submitted\_Transaction_{i,(0...j-1)}} \tag{5}$$

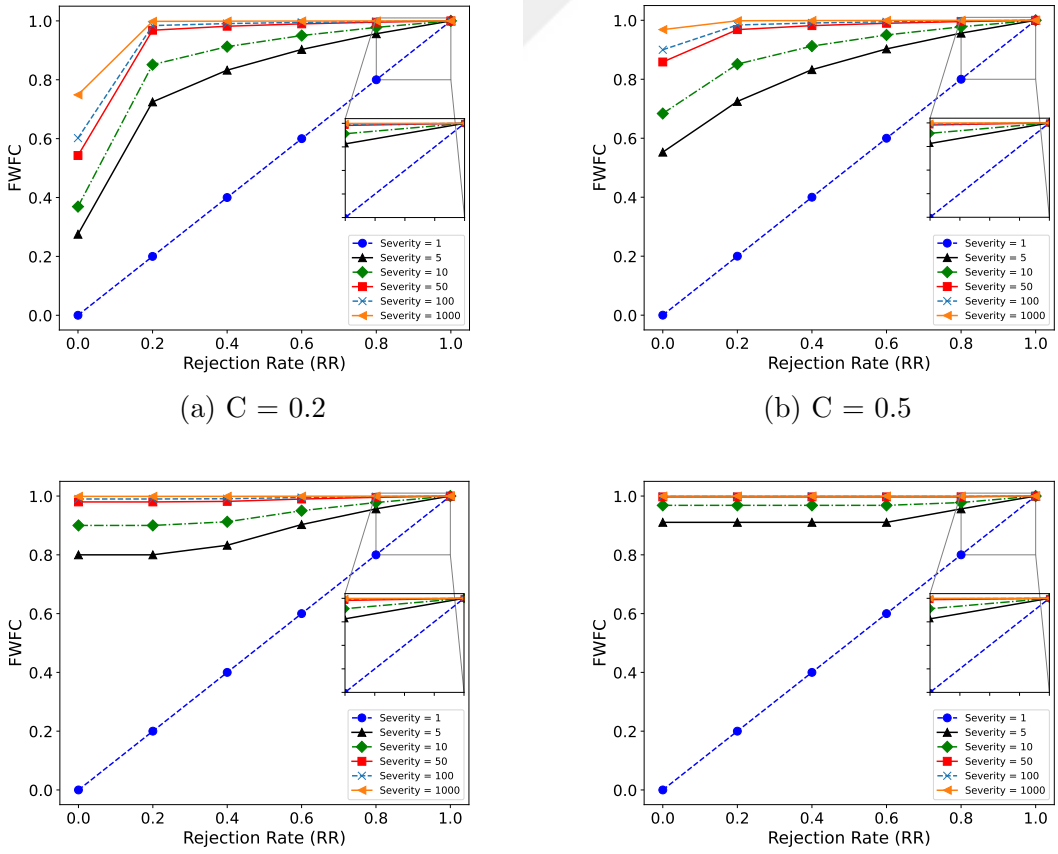
Based on Eq. (4) and Eq. (5), we define $FWFC$ for switch i during epoch j as follows:

$$FWFC_{i,j} = max\Big\{RR_{i,j-1}^{1/Severity_{i,j-1}}, 1 - (1/Severity_{i,j-1})^{C}\Big\} \tag{6}$$

$FWFC_{i,j}$ represents the fraction of transactions the switch i has to wait during epoch j. If the switch does not follow the $FWFC_{i,j}$, then the SDN controller will take the proper action in epoch $j+1$. Therefore, the length of the time window of each epoch specifies how fast malicious/compromised switches are detected in CWT-DPA.

Both RR and severity control $FWFC$. When RR or severity increases, then the switch has to wait for a higher fraction of transactions to be committed, and therefore $FWFC$ increases as well. RR is used as a directed indication, whereas severity is used as an indirect indication for the need to wait for the final commitment. When

RR is low, then $FWFC$ is defined merely based on the severity value. It is also worth noting that severity value should be larger than zero. C is a penalty parameter used to control the threshold for switches with low rejection rates and to decrease the difference between high severity values. A small C value results in a more smooth transition (i.e., small difference) among different severity values when the switch has a low rejection rate. In contrast, a large C value will ultimately lead to a sharper transition (i.e., a larger difference) between different severity values. Therefore, in critical systems, we expect C to be large to ensure that high proportion of transactions is committed when severity is increased.



(a) C = 0.2 (b) C = 0.5

(c) C = 1 (d) C = 1.5

Figure 20: Fraction of waiting for final commitment (FWFC) vs rejection rate [3].

The probability of correctness for switch i in epoch j is calculated as follows:

$$P(Correctness_{i,j}) = \Big(1 - P(Mal\_Majority_i)\Big) \times FWFC_{i,j} \tag{7}$$

Let $P(Mal\_Majority)$ be the probability that a malicious node wins voting, and let $p(m)$ be the probability that any BC node votes for a malicious node. The number of votes that the malicious node receives follows a binomial distribution. The probability that the malicious node wins is equal to the probability of receiving a minimum of $\frac{n}{2} + 1$ votes:

$$P(Mal\_Majority) = \sum_{k=\frac{n}{2}+1}^{n} \binom{n}{k} p(m)^k (1 - p(m))^{n-k} \tag{8}$$

Then, the expected correctness for switch i over n epochs is

$$E(CorrectTx_i) = \sum_{k=1}^{n} Transaction_{i,k} \times P(Correctness_{i,k}) \tag{9}$$

where $Transaction_{i,k}$ is the number of transactions submitted in epoch k by the switch i.

Figure 20 shows how $FWFC$ changes according to different RR, *severity*, and C values. The figure also shows that when RR increases $FWFC$ also increases. In addition, $FWFC$ increases for cases with the same RR value but higher severity. Finally, C determines how large the difference is among $FWFC$ values for cases with different severity and low RR values.

The best case is when $FWFC = 0$ where the switch will submit all transactions without waiting for final commitment. In contrast, the worst case is when $FWFC = 1$, which means that the switch must wait for the commitment of all submitted transactions, which also represents our baseline BC-SDN model (DPSEC [1]).

Figure 21 shows $P(Correctness)$ values for different $FWFC$ and different number of malicious nodes. $P(Correctness)$ presents guarantees on final commitment for transactions submitted by a certain switch and, therefore, can be further used as Proof of Commitment (PoC) in our blockchain. Figure 21 also shows that when malicious nodes form a majority, $P(Correctness)$ decreases significantly. In addition, low $FWFC$ values will also result in low $P(Correctness)$. This shows a trade-off between correctness and performance of CWT-DPA.

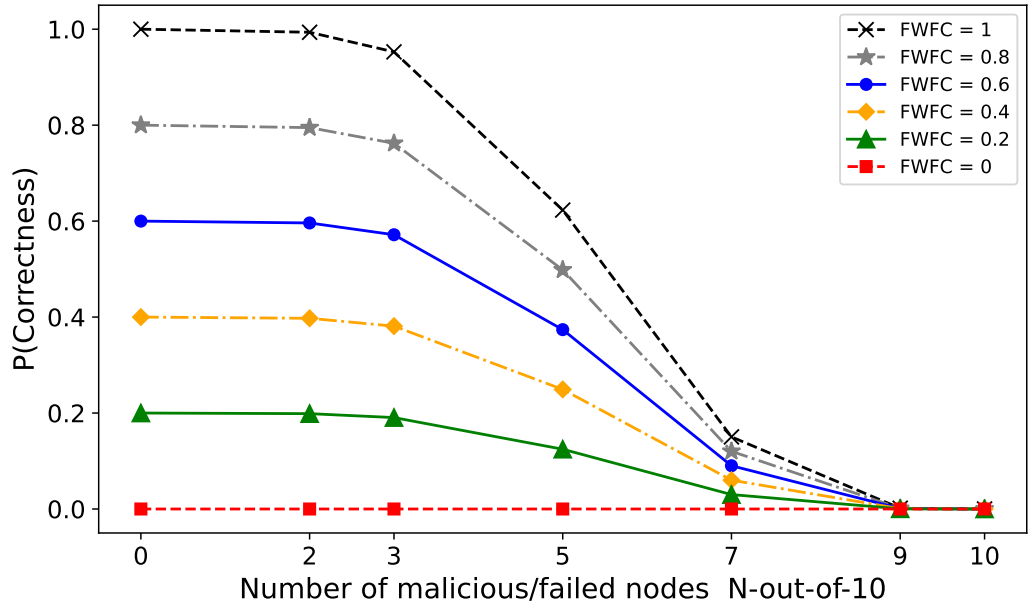


Figure 21: Probability of correctness for different FWFC and different number of malicious nodes [3].

3.4.3.2 Aggregation of Host Transactions

We propose a method to aggregate host-related transactions and improve the overall latency. We aim to send several *Port_Req* in a single transaction. The switch aggregates the transactions based on predefined batch size. The switch also sets a batch timeout, which allows it to send an uncompleted batch after passing the timeout. Although the transaction is aggregated, the smart contract needs to check each record

and adds only the valid MAC. In other words, if the MAC is blacklisted or used by another host it should not be added to the BC. We update the smart contract, to handle aggregated transactions without losing the ability to validate each MAC address separately. Our updated contract is shown in Algorithm 7. The smart contract makes use of Attribute-Based Access Control (ABAC) to achieve access control in BC-SDN. The contract checks each host record in the batch and creates two lists namely, $Accepted\_Hosts$ and $Rejected\_Hosts$, which indicate accepted and rejected host records in the batch, respectively.

Algorithm 7: Add Batch Function [3]

```
Input: caller, DPID, BatchID, Batch
if caller.role == client then
    if caller.state == Trusted then
        if caller.dpid == DPID then
            Accepted Hosts = ∅ ;
            Rejected Hosts = ∅ ;
            foreach Host : h_i (MAC_i, InPort_i) ∈ Batch do
                if MAC does not exist as a valid record then
                    Accepted Hosts ← Accepted Hosts ⋃ h_i ;
                else
                    Rejected Hosts ← Rejected Hosts ⋃ h_i ;
                end
            end
            Add a new record (BatchID, Accepted Hosts, DPID);
            Update the state of Rejected Hosts ;
            Emit a new event to notify C_i;
        else
            Reject the request;
            Update caller state to Untrusted;
            Emit a new event to notify C_i about caller.dpid status change;
        end
    else
        Reject the request;
    end
else
    Reject the request;
end
```

3.5 Experimental Results

In this section, we provide detailed experiments that demonstrate the applicability and feasibility of our protocol, along with comparisons with existing solutions. The experimental work is conducted on a physical machine with Ubuntu 20.04 OS. We used liboqs[2] library from the Open Quantum Safe project to enhance the security of switch authentication against quantum adversaries.

We extended Open vSwitch[3] to interact with BC. As an SDN controller, we use Ryu[4] with OpenFlow 1.3. We used Hyperledger Fabric[5] for our BC part. The details of our experimental setup are given in Table 2. For the SDN part, we modified the handshake for both Ryu controller and OVS switch to follow CWT-DPA specification based on liboqs[2] library. For SDN's data plane, we created CWT-DPA authentication module, which is written in C language based on liboqs library and used to generate PQ ephemeral key pairs, sign/verify PQ signatures.

#	Details
CPU	Intel i7-8550u
RAM	8 GB
Operating system	Ubuntu 20.04
Emulator	Mininet 2.3.0
SDN controller	Ryu 4.34
SDN switch	OVS 2.15.1
Blockchain	Hyperledger Fabric 1.4
Number of BC organizations	2 organizations, 2 peers per organization
Block size	100 transaction per block
Block frequency	1 block per second

Table 2: Experimental setup

[2]Open Quantum Safe project (2022). OQS [online]. Website https://openquantumsafe.org/liboqs/ [accessed 10 Jan 2022]

[3]Linux Foundation (2011). OVS [online]. Website https://github.com/openvswitch/ovs [accessed 06 May 2021]

[4]Ryu-sdn.org (2014). Ryu [online]. Website https://ryu.readthedocs.io/en/latest/ [accessed 17 May 2020]

[5]Hyperledger.org (2018). Hyperledger Fabric [online]. Website https://hyperledger-fabric.readthedocs.io/en/release-1.4/ [accessed 17 May 2020]

BC-gateway is written in NodeJS. For the controller part, we utilized liboqs-python as a wrapper for the liboqs C library. In addition, BC nodes include a PQ verification module to verify PQ signatures based on the liboqs library [75].

Our consortium BC part includes two organizations and consists of 4 peers in total, 2 peers for each. In addition to one orderer peer, responsible for determining the order of BC transactions [76]. We used Mininet to emulate our SDN network. We compared the latency of CWT-DPA with SSL/TLS [65], DPSEC [1] and AuthFlow [7]. We measured the latency for both switch and host authentication steps. Then, we reported the obtained results. Figure 22 depicts the topology used in our PoC where we gradually increased the rate at which SDN switches are added to the Mininet network. We utilized low-level APIs of Mininet to add SDN switches to the same Mininet network dynamically. Thread libraries are also used for concurrently adding the SDN switches, hosts, and their corresponding links to the SDN network.

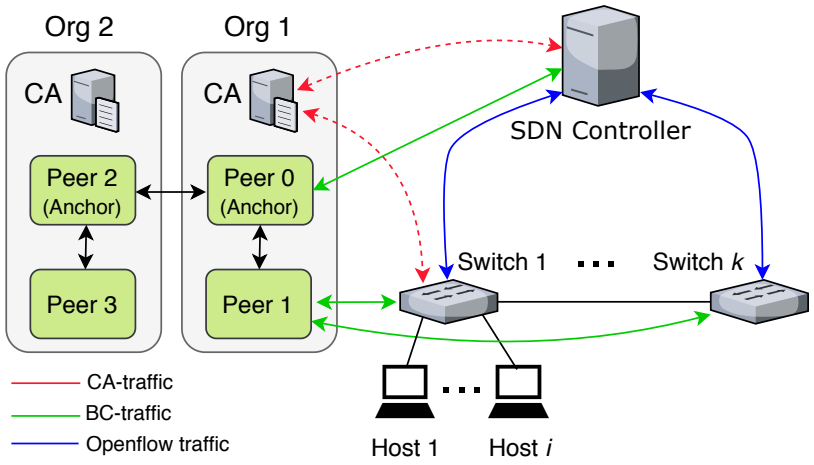


Figure 22: Topology used in our PoC © 2020 IEEE [1].

3.5.1 Switch-Auth Latency

We utilize lattice-based signatures based on Falcon1024 [66] and Dilithium3 [67], where Falcon1024 provides level 5 security according to the security categories defined by National Institute of Standards and Technology (NIST) [77]. Dilithium3, on the other hand, provides level 3 security. We also choose NTRU-HPS-2048-677 [68] and Kyber768 [69] as KEMs, providing level 3 security. As shown in Figure 23, the highest latency is achieved when NTRU-HPS-2048-677 and Falcon1024 are used. Whereas the best latency is achieved when Kyber768 and Dilithium3 are used.

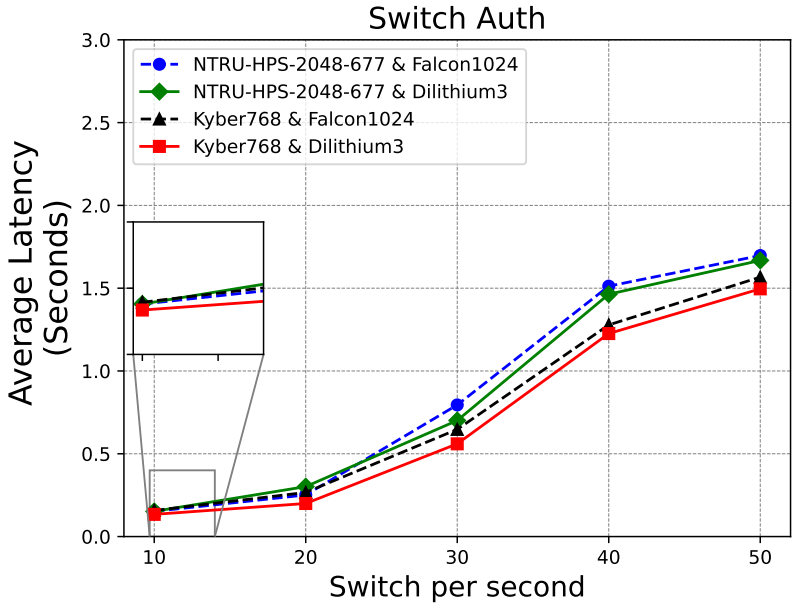


Figure 23: Latency for switch Auth in CWT-DPA [3].

3.5.1.1 Comparing with SSL/TLS and DPSEC

This section compares CWT-DPA with SSL/TLSv1.3 [65] and DPSEC [1]. We consider a combination of Kyber768 and Dilithium3 since it shows the lowest latency for CWT-DPA. SSL/TLS is the main protocol to protect the SDN's southbound API. Whereas DPSEC [1] is a BC-based protocol for authenticating the data plane of SDN.

Figure 24, shows that CWT-DPA outperforms DPSEC [1] in terms of latency. This is due to waiting for final commitments in DPSEC. Figure 25 also presents the average latency for CWT-DPA, DPSEC, and SSL/TLSv1.3, where DPSEC shows the worst average latency, which is 4.76x of CWT-DPA when a combination of Kyber768 and Dilithium3 is used.

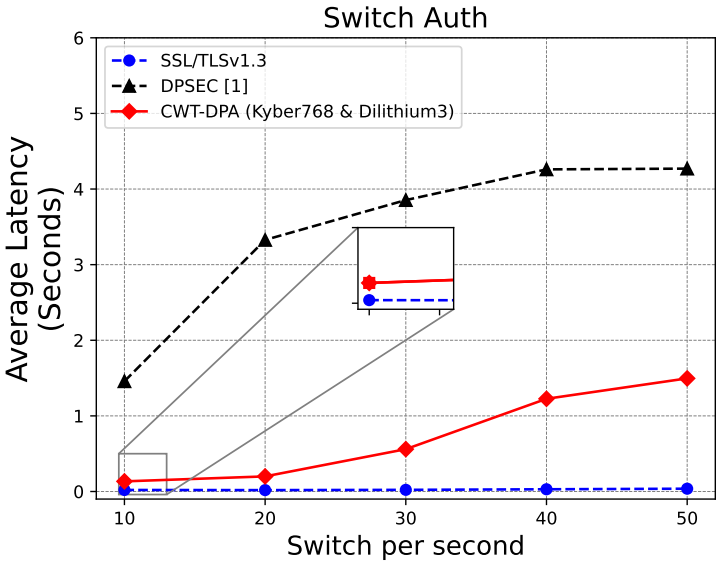


Figure 24: Comparing switch Auth latency for CWT-DPA with SSL/TLSv1.3 and DPSEC [3].

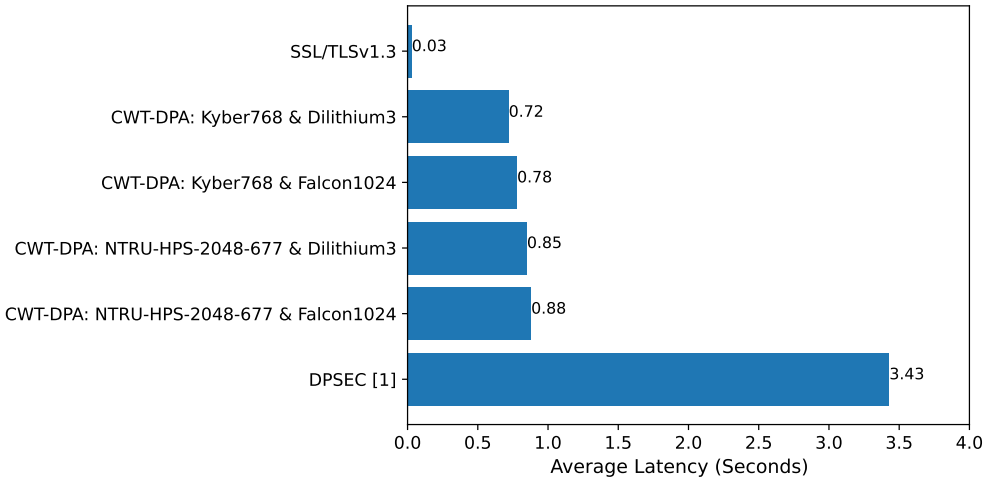
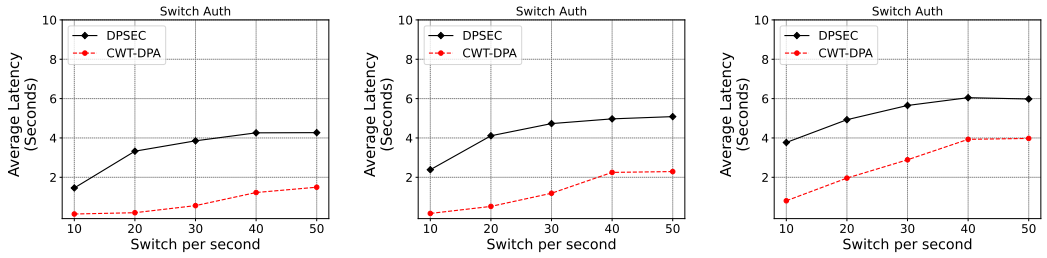


Figure 25: Comparing the average latency for switch Auth with SSL/TLSv1.3 and DPSEC [3].

On the other hand, SSL/TLSv1.3 shows the lowest latency, where the average latency of CTW-DPA based on Kyber768 and Dilithium3 is 24x of SSL/TLSv1.3. This is due to the fact that BC approaches have to send transaction proposals and collect the required endorsements from the corresponding BC peers before interacting with the authentication contract. Moreover, the interaction with BC peers is also protected using SSL/TLS, which explains the additional overhead compared to SSL/TLS without BC. In addition to encryption, signing, and verification operations at the smart contract and BC peers. The advantage of BC-enabled approaches is achieving fault-tolerant, decentralization, and access control using BC smart contracts. Adding these critical proprieties, which are not provided in TLS-based authentication, with such low latency is a significant advancement to improve the SDN paradigm.

3.5.1.2 The Effect of Increasing the Number of Organizations

We also studied the effect of increasing the number of organizations, where we considered three cases. The first case consists of 2 organizations with two peers, representing our base model. In the second case, we included five organizations with two peers for each one. In the third case, we considered ten organizations with two peers for each one. The obtained results are shown in Figure 26. For the first case, the results show that the latency for CWT-DPA remains under 2 seconds for all switch rates. For the second case, the results show that the latency for CWT-DPA remains under 3 seconds for all switch rates. Whereas in the third case, the results show that the



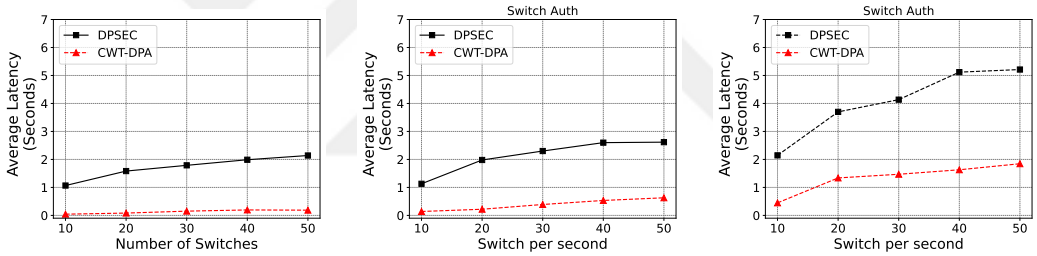
(a) 2 Organizations (b) 5 Organizations (c) 10 Organizations

Figure 26: The effect of increasing number of organizations [3].

latency increases above 3 seconds when the switch rate is higher than 30 switches per second. In the second case, CWT-DPA achieves 3.32x improvement over DPSEC [1]. In the third case, CWT-DPA achieves 1.98x improvement over DPSEC [1].

3.5.1.3 Performance After The Connection is Established

In this case, the connection is established and the switch already has interacted with BC nodes by sending BC transactions. As shown in Figure 27, the performance of both DPSEC and CWT-DPA improves. For DPSEC, the latency remains below 3 seconds in the first and second cases. Whereas in the third case the latency increases above 4 seconds when the number of switches is higher than 30. For CWT-DPA, on the other hand, the latency remains below 2 seconds for all cases.



(a) 2 Organizations (b) 5 Organizations (c) 10 Organizations

Figure 27: The average latency after the connection is established.

3.5.2 Host-Auth Latency

We follow our baseline model for the host authentication step (DPSEC[1]). In DPSEC, each host is authenticated by sending *Port_Req*, which binds the corresponding MAC/Port to the blockchain. Then, the host sends its *Auth_Req*, which includes the client certificate and signature, to complete the authentication step. To measure the latency of the host authentication in CWT-DPA, first, we measure the latency for *Port_Req*. Then, we modify *Auth_Req* to use only the client signature and compare it with the original *Auth_Req* of DPSEC [1]. Then, we measure the overall latency and compare it with DPSEC, and AuthFlow [7]. It is worth noting that the

latency occurs only for unauthenticated hosts, which need to be authenticated before utilizing the SDN network.

3.5.2.1 Port Request Latency

Enhanced waiting time Since each *Port_Req* is sent to BC, we consider three main cases for $FWFC$. The first case ($FWFC = 1$) represents the worst case for CWT-DPA, which is equivalent to DPSEC [1] since the switch has to wait for the final commitment. Whereas the second case ($FWFC = 0.5$) represents the average case for CWT-DPA, where the switch waits for final commitment for only 50% of *Port_Req* transactions during that epoch. The third case ($FWFC = 0$) represents the best case for CWT-DPA, where the switch does not have to wait for the final commitment during that epoch. Figure 28 shows the cumulative distribution function (CDF) of the latency of *Port_Req*.

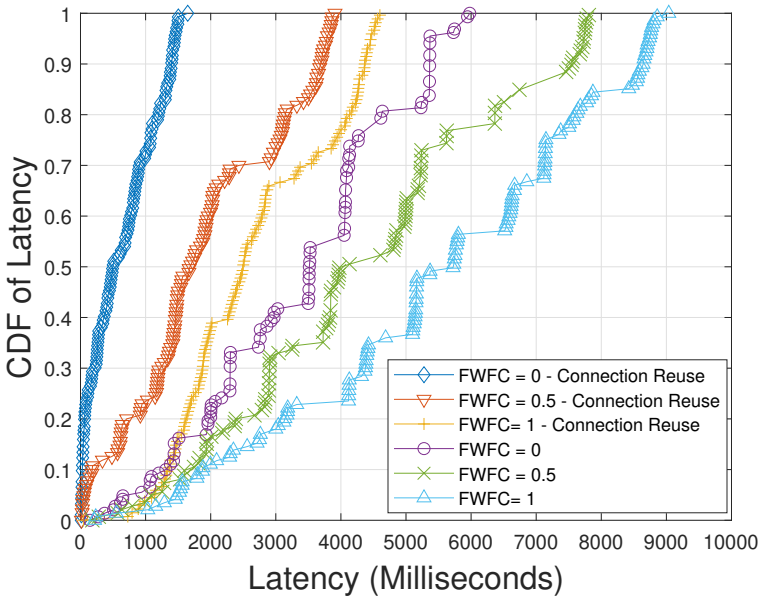


Figure 28: CDF of latency for port request [3].

The CDF shows that when connection reuse is not utilized, the worst case shows that only 36% of the requests were satisfied in less than 5000 milliseconds. Whereas the average case showed that %61 of the requests were satisfied in less than 5000

milliseconds. The best case showed that %81 of the requests were satisfied in less than 5000 milliseconds. When connection reuse is utilized %37 of the requests achieved low to moderate latency (less than 2000 milliseconds) in the worst case, whereas %60 of the requests had low to moderate latency in the average case. The best case showed that %72 of the requests had low to moderate latency (less than 1000 milliseconds) and %49 of the requests had low latency (less than 500 milliseconds). The results show that the latency significantly improved when connection reuse is utilized for all cases.

Figure 29, shows the average latency for each case with different numbers of concurrent host transactions when connection reuse is not utilized. For all cases, the latency increases when the number of concurrent host transactions increases. For the worst case (i.e., FWFC = 1), the average latency increases significantly above 4 seconds when the load is higher than 30 hosts. For the average case (i.e., FWFC = 0.5), the average latency is higher than 4 seconds only when the load is higher than 50 hosts. Whereas, for the best case (i.e., FWFC = 0), the average latency is higher than 4 seconds only when the load is higher than 70 hosts.

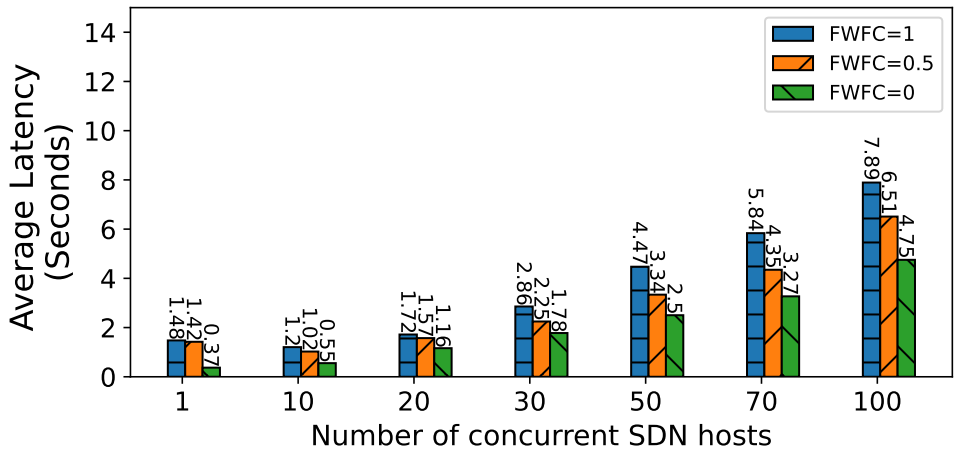


Figure 29: The average latency for port request without connection reuse [3].

Figure 30 shows the average latency for each case when connection reuse is utilized. For the worst case (i.e., FWFC = 1), the average latency is higher than 1 second for all applied loads. For the average case (i.e., FWFC = 0.5), the average latency is higher than 2 seconds only when the load is higher than 50 hosts. Whereas, for the best case (i.e., FWFC = 0), the average latency is above 1 second only when the load is higher than 70 hosts. The results show that the average latency increases proportionally to FWFC and the load on SDN switches and BC peers.

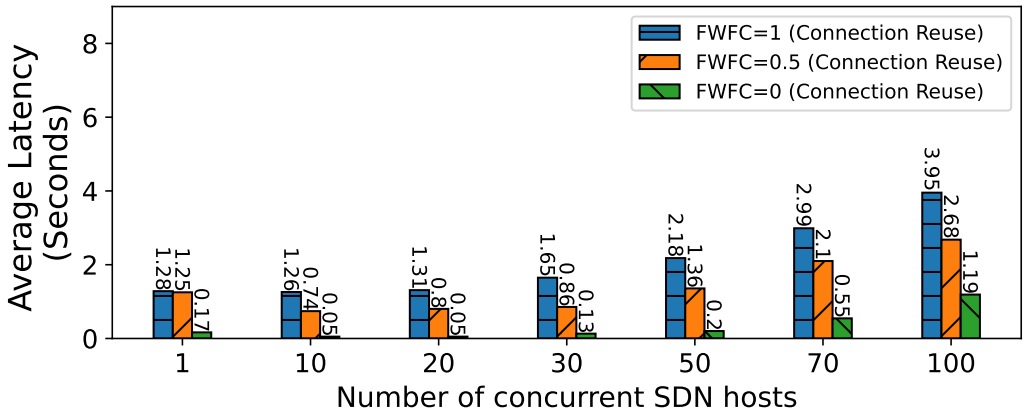


Figure 30: The average latency for port request with connection reuse [3].

Figure 31 shows the overall average for each case. Without utilizing connection reuse, in the best case (i.e., when $FWFC = 0$) the average latency is reduced by 39.23% compared to DPSEC [1] ($FWFC = 1$). Whereas, in the average case (i.e., when $FWFC = 0.5$) and without utilizing connection reuse, the average latency is reduced by 20.8% compared to $FWFC = 1$.

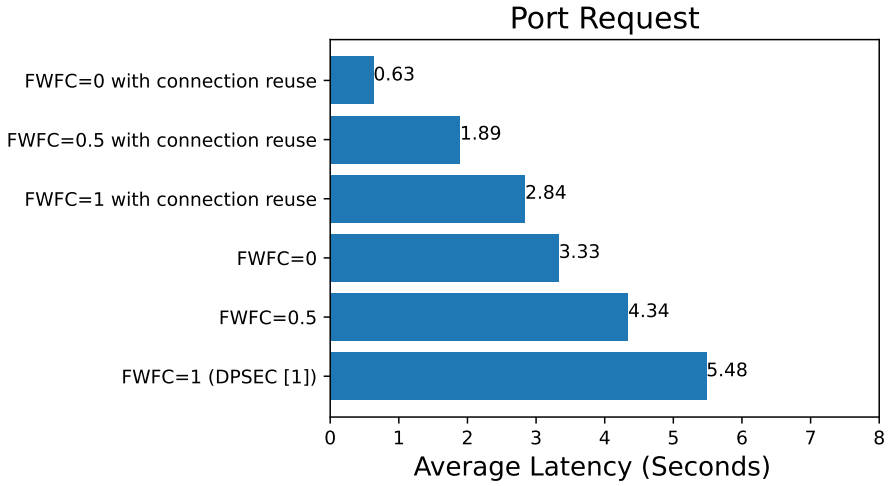
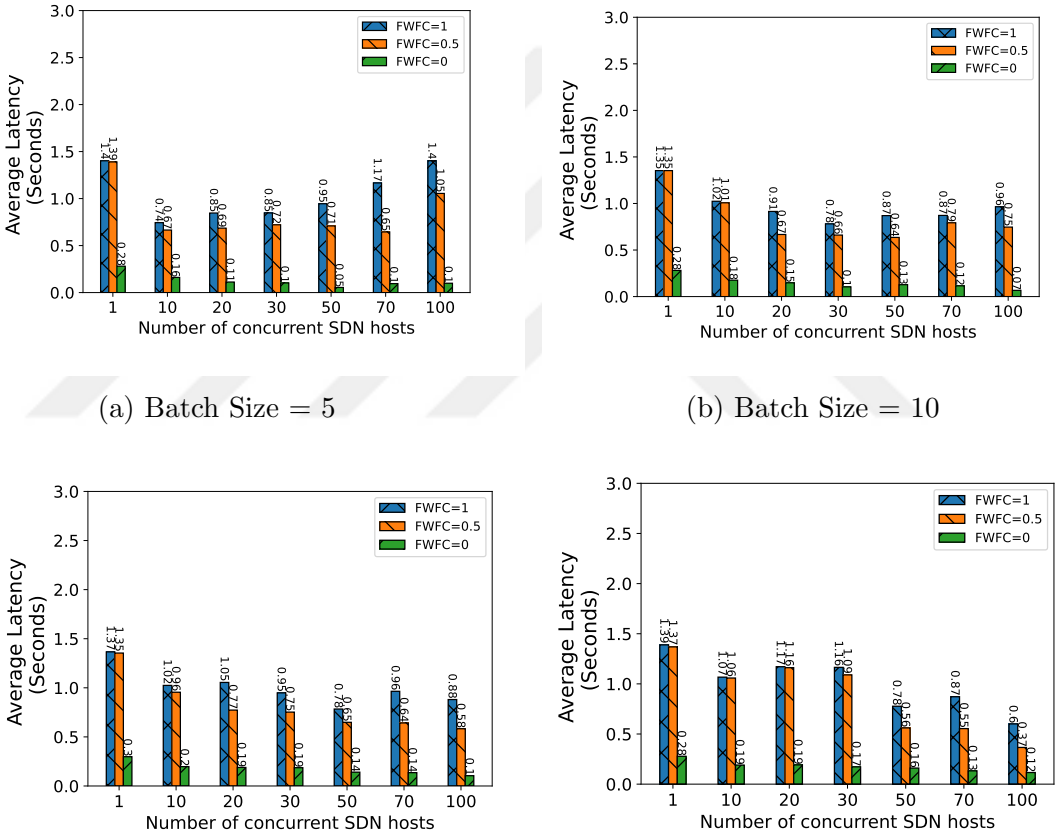


Figure 31: The overall average latency for port request [3].

By only utilizing connection reuse, the average latency is reduced by 48.17% for $FWFC = 1$ (worst case), by 56.45% for $FWFC = 0.5$ (average case), and by 81.08% for $FWFC = 0$ (best case). However, the best performance is achieved when connection reuse is combined with the CWT-DPA model. When $FWFC = 0$ and connection reuse is utilized, the average latency is reduced by 88.5% for compared to $FWFC = 1$ (DPSEC [1]) without connection reuse. Similarly, when $FWFC = 0.5$ and connection reuse is utilized, the average latency is reduced by 65.51% of DPSEC [1] without connection reuse.

Aggregation of host transactions In this section, we study the effect of aggregation of $Port\_Req$. We updated the smart contract to validate aggregated transactions. We used the same topology with different batch sizes (5,10,20,30). We considered that the switch is utilizing connection reuse to reduce the latency. We report the results for worst, average, and best cases. Figure 32 shows the average latency for different batch sizes with different loads. The results suggest an improved latency according to the applied load and the batch size. As shown in Figure 32 (a) and (b), small batch sizes are suitable in the case in which the switch has a low to moderator load.

Whereas large batch sizes, Figure 32 (c) and (d), are more appropriate for the case in which the switch has a high load. When the batch size is high, and the load is low, the latency increases since the switch has to wait for the timeout. Therefore, the choice of batch size depends on the load on that particular switch. Switches that are under high loads need to choose a large batch size. Whereas switches that have low loads need to choose a small batch size.



(a) Batch Size = 5

(b) Batch Size = 10

(c) Batch Size = 20

(d) Batch Size = 30

Figure 32: The average latency for port request with different batch sizes [3].

Figure 33, shows the average latency for each batch size and compares it with our baseline (i.e., when batching is not applied). The average latency is reduced for all batch sizes. Larger blocks achieved better latency for the average and worst cases. However, it decreased the latency for the best cases compared to smaller batch

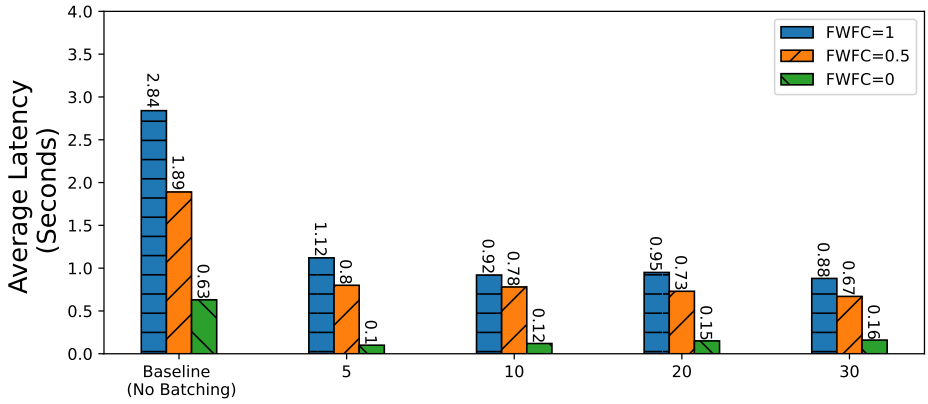


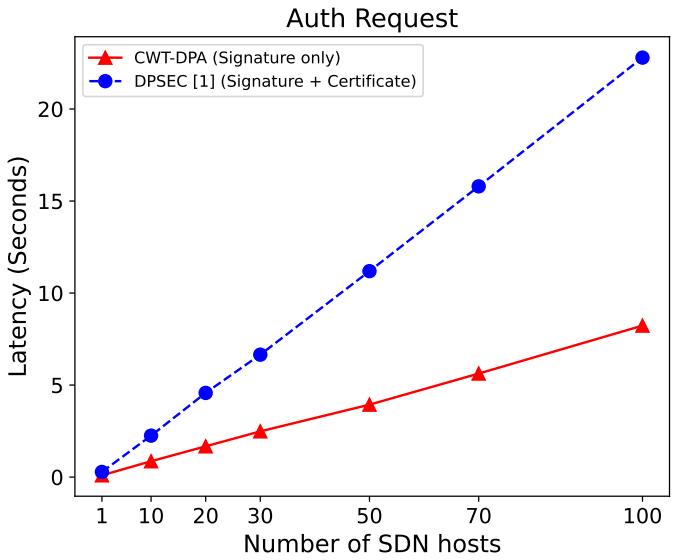
Figure 33: The overall average latency for port request with batching technique [3].

sizes. The results also show that the latency difference between the average and worst cases decreases compared to our baseline. This is due to the fact that the number of transactions the switch has to wait is significantly reduced. The minimum enhancement is 60.56%, 57.67%, and 74.6% reduced latency for the worst, average, and best cases, respectively. The maximum enhancement is 69.01%, 64.55%, and 84.13% reduced latency for the worst, average, and best cases, respectively.

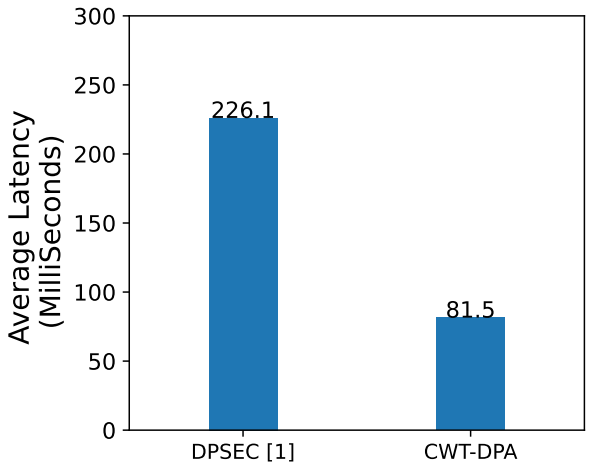
3.5.2.2 Auth Req Latency

To reduce the overhead of sending client certificate, each client's MAC/public key is stored on BC at the registration stage, where each host is binded with a tuple of MAC/public key/port. Therefore, we reduce certificate transmission and verification overhead. The smart contract rejects the requests from unknown MAC/public key pairs. Therefore, in CWT-DPA, the client is authenticated based on 1) successful binding with a smart contract and 2) successful signature verification based on Elliptic Curve Digital Signature Algorithm (ECDSA) [64] using curve secp256. To avoid reply attacks, the controller sends a challenge to the client, which has to sign and send to the corresponding controller. In addition, we send each authentication request of DPSEC [1] and CWT-DPA using EAP protocol (layer-2) packets to reduce the overhead of using upper-layer protocols. As shown in Figure 34(a), the latency of

authentication request is significantly reduced compared to DPSEC [1], where the *Auth_Req* of CWT-DPA is 0.36x of DPSEC [1]'s *Auth_Req*. Figure 34(b), shows the average latency for the *Auth_Req* of CWT-DPA and DPSEC [1] where the average latency is decreased by 63.95%.



(a) Auth request latency



(b) The average latency for Auth request

Figure 34: Comparing latency of Auth request [3].

3.5.2.3 *Full Host Authentication*

In this part, we compare the overall latency of CWT-DPA with DPSEC [1], and AuthFlow [7]. To make a fair comparison, we applied connection reuse for all methods. AuthFlow [7] provides authentication using the RADIUS server that utilizes MS-CHAPv2 as an authentication method. CWT-DPA and DPSEC [1] perform binding at the data plane level. AuthFlow [7] performs binding at the control plane level. Therefore, in both scenarios, the authentication requests are first sent to the controller, then forwarded to the authenticator component (RADIUS client), and finally authenticated by RADIUS server [7] based on MS-CHAP-v2, which also requires the client to receive a new challenge in each authentication request.

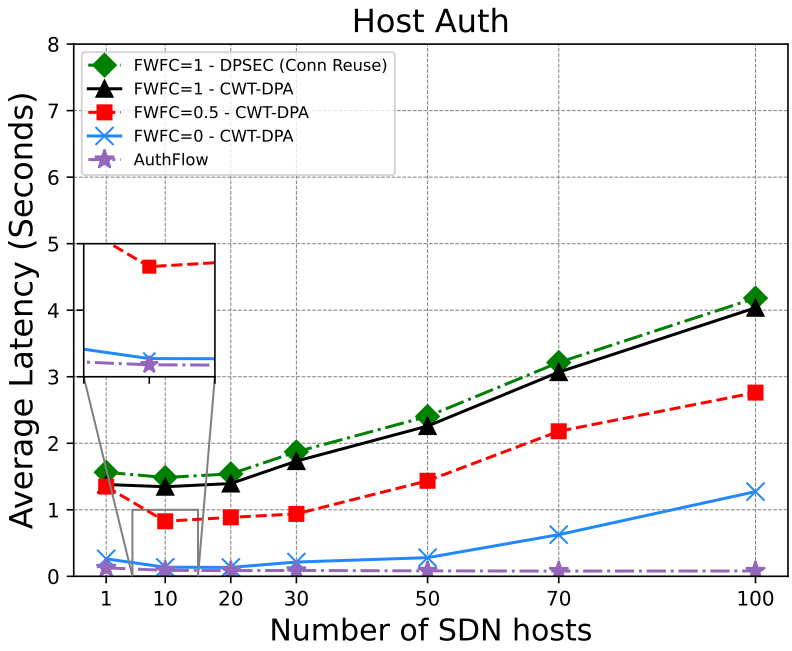


Figure 35: Average latency for full host authentication [3].

Comparing with DPSEC [1] As shown in Figure 35, DPSEC [1] has the highest latency due to waiting for the final commitment and sending the client certificate. The average latency of CWT-DPA in the best case remains under 1.5 seconds for all

loads. For the average case, the latency increases above 2 seconds when the load is higher than 50 hosts. In the worst case, the latency is higher than 2 seconds only when the load exceeds 30 hosts. On the other hand, the average latency for DPSEC[1] increases above 2 seconds when the load is higher than 30 hosts. Figure 36 also shows that when batching is applied for CWT-DPA, the average latency remains under 1.5 second for all cases and loads.

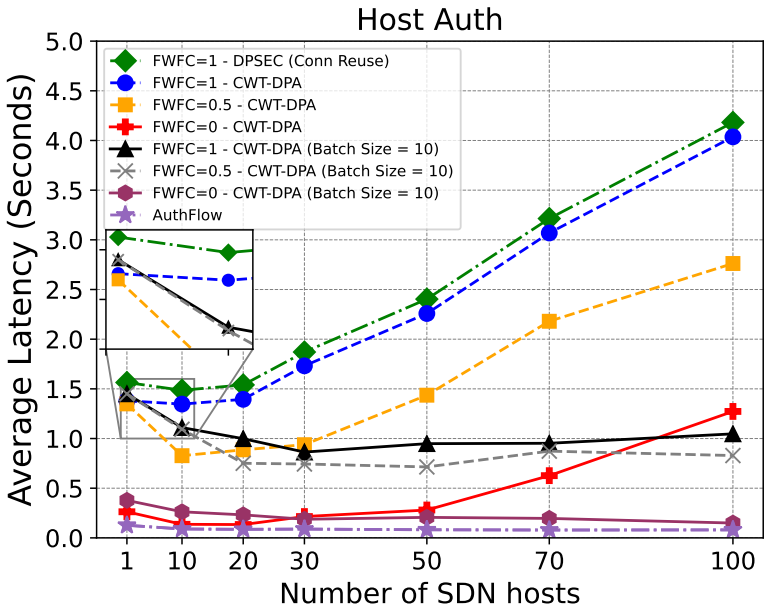


Figure 36: Average latency for full host authentication with batching technique [3].

The average latency is shown in Figure 37. In the worst case ($FWFC = 1$) of CWT-DPA, the average latency decreases by 4.88% compared with DPSEC [1]. The average case shows 35.5% decreased latency compared with DPSEC [1]. Whereas the best case shows 76.87% decreased latency compared with DPSEC [1]. When batching is used, the worst case of CWT-DPA achieves a 67.10% decrease compared to DPSEC [1]. The average and best cases also show 71.98% and 93.16% decreased latency compared with DPSEC [1].

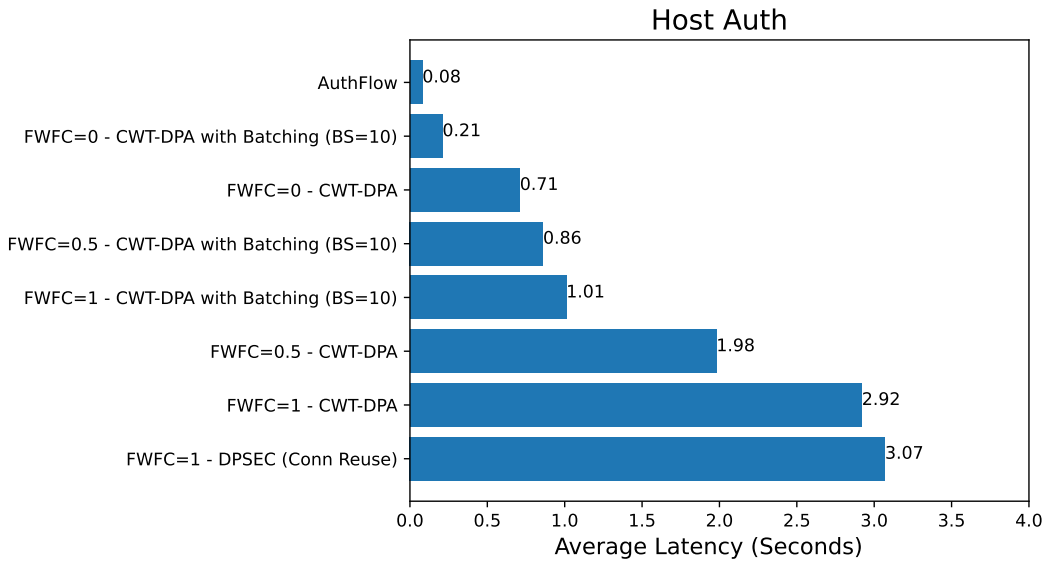
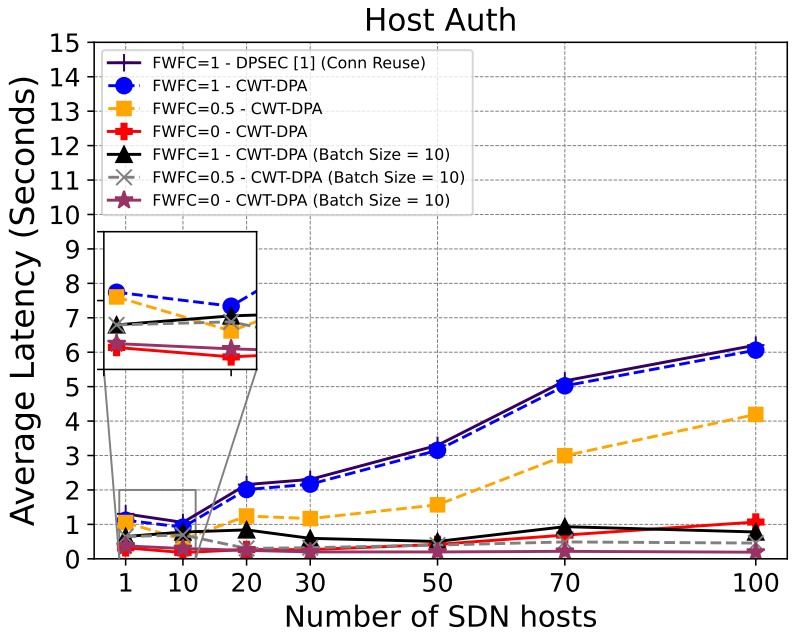


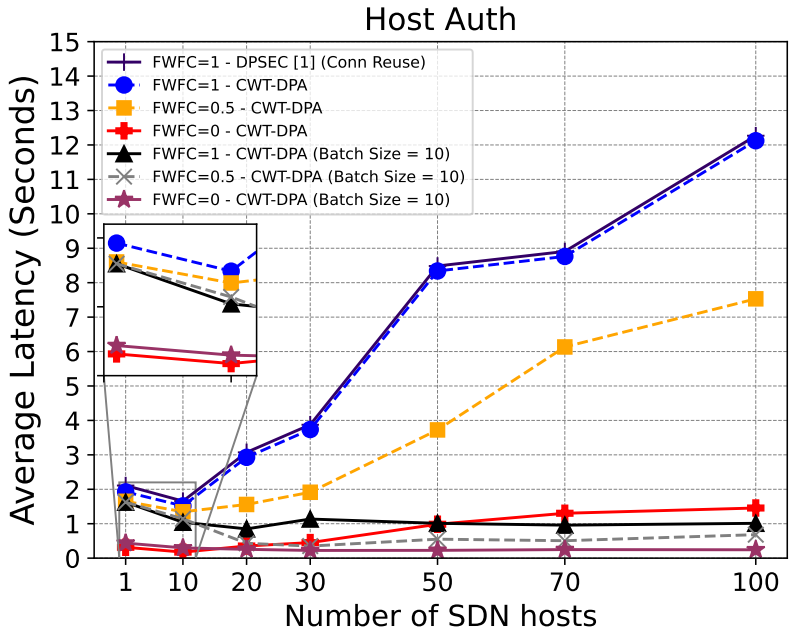
Figure 37: The overall average latency for full host authentication [3].

Comparing with AuthFlow [7] As shown in Figure 36, AuthFlow [7] outperforms CWT-DPA in terms of latency. The latency is due to endorsing and sending *Auth_Req* of the corresponding host to BC and concurrently sending the requests increases the average latency, as we have shown before. The worst case of CWT-DPA is 36.05x of the latency of AuthFlow [7]. The average case is 24.45x, whereas the best case is 8.87 of AuthFlow [7]. This latency can be significantly decreased when batching is applied. When batch size is set as 10, the average case is reduced by 10.75x of AuthFlow [7]. At the same time, the worst and best cases are reduced to 12.63x and 2.63x of AuthFlow, respectively. Unlike AuthFlow [7], the host authentication of CWT-DPA is designed on top of a switch authentication approach with built-in fault-tolerant and attribute-based access control. Therefore, we ensure that each host is connected to a trusted switch beforehand. CWT-DPA also provides strong authentication based on client signature, ensures protection for SDN controllers by performing the binding process at the data plane level, and provides fault-tolerance and decentralization in SDNs. In addition, SDN switches are aware of the authentication process and the state of other SDN switches and hosts.

The Effect of Increasing the Number of Organizations We also studied the effect of an increased number of organizations, where we considered 5 and 10 organizations, each consisting of 2 peers. As shown in Figure 38, the best results are achieved



(a) Case 1: 5 Organizations



(b) Case 2: 10 Organizations

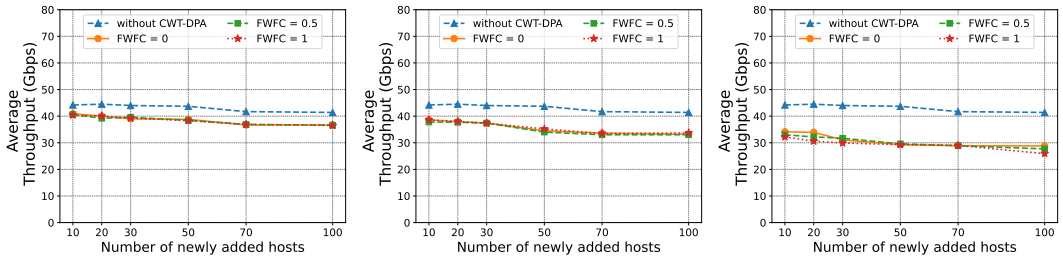
Figure 38: The effect of increasing the number of organizations [3].

when batching is activated. For five organizations, the latency remains less than 1 second for all cases when batching is applied. The best case shows that CWT-DPA achieves a 21.18x improvement over DPSEC [1]. CWT-DPA achieves 10.86x and 6.17x improvements in the average and worst cases, respectively. For ten organizations, the average latency remains less than 2 seconds for all cases when batching is applied. The best case shows that CWT-DPA achieves 35.37x improvement over DPSEC[1] when batching is applied. CWT-DPA achieves 16.31x and 8.59x improvements over DPSEC[1] in the average and worst cases, respectively.

The Effect of BC transactions on network performance We also study the effect of BC transactions on network performance. In this case, the switch has to perform networking operations such as packet forwarding, and also interacts with BC nodes due to some events, such as new hosts being connected to the corresponding switch. We give a scenario in which two authenticated hosts (i.e., client and server) need to exchange Transmission Control Protocol (TCP) traffic generated by the Iperf tool [78]. At the same time, new hosts are connected to the corresponding switch, which generates new BC transactions according to our proposed system. We measured the network throughput using the Iperf tool, where we considered three cases for BC (2, 5, and 10 organizations).

The obtained results are shown in Figure 39. As expected, increasing the number of BC organizations results in decreased network throughput. In the first case, the throughput decreases below 40 Gigabits per second (Gbps) when the number of newly added hosts exceeds 30. The second case shows that the throughput remains above 30 Gbps. In the third case, the throughput decreases below 30 Gbps when the added hosts are higher than 20.

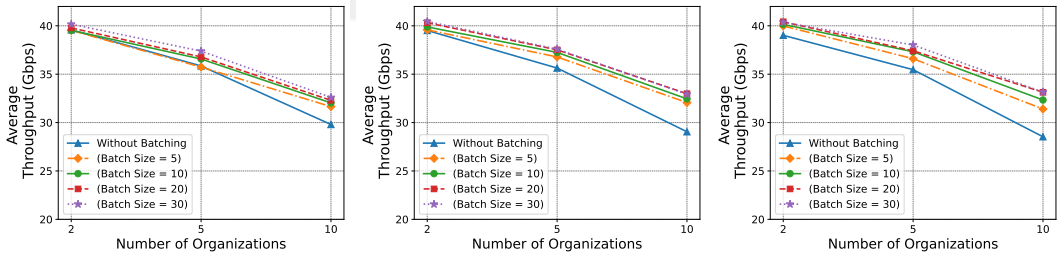
We also study the effect of applying the batching technique on network throughput. We compare the obtained results with our baseline (i.e., when batching is not



(a) 2 Organizations (b) 5 Organizations (c) 10 Organizations

Figure 39: The effect of BC on network performance [3].

applied). As shown in Figure 40, when batching is applied network throughput is increased for different batch sizes. The average throughput remains above 30 Gbps for all cases. Overall, the results confirm that the network performance is reduced when the switch has to communicate with other BC nodes. However, this task is required only when the binding is changed. In other words, hosts attached to the same port do not require sending a new transaction to BC.



(a) FWFC = 0 (b) FWFC = 0.5 (c) FWFC = 1

Figure 40: The effect of applying the batching technique on network throughput [3].

3.6 Discussion

The main advantages of CWT-DPA are as follows. First, CWT-DPA significantly reduces the latency of BC-SDN models compared with DPSEC [1]. Second, our switch authentication approach protects against quantum adversaries. Third, the host authentication is built on top of our switch authentication method, which provides fault-tolerant and access control and ensures that each host is connected to a trusted switch beforehand. Fourth, CWT-DPA performs a BC-based binding process that

can be also used as a defense mechanism to prevent other hosts from using the same MAC (i.e., spoofing attacks). Therefore, BC is used to share the information across different SDN switches based on smart contracts without frequently interacting with each SDN controller, which reduces the load on that SDN controller. Fifth, since the binding process is performed at the data plane level, the controller is protected against malicious events. Finally, each host is binded to a specific MAC address. In contrast, in AuthFlow [7], the client can have an unlimited number of MAC addresses as long as it has a legitimate identity.

3.7 Summary

In this chapter, we presented CWT-DPA, a component-wise waiting time to enhance the performance of BC-SDN models by separating the switch and host transactions. We conduct our experiments using the OVS switch and compare the performance of CWT-DPA with existing solutions such as SSL/TLSv1.3 [65], DPSEC [1], and AuthFlow [7] approaches. CWT-DPA showed better performance, in terms of latency, compared to DPSEC [1] when used to authenticate SDN switches. In addition, our switch authentication approach based on CWT-DPA protects quantum adversaries.

For the host authentication part, CWT-DPA showed better latency than DPSEC [1]. Both CWT-DPA and AuthFlow [7] achieves low latency compared to DPSEC [1]. However, CWT-DPA has higher latency compared to AuthFlow [7]. When batching is used, the average case is reduced to 10.75x of AuthFlow [7]. CWT-DPA provides more protection for SDN controllers, in comparison with AuthFlow [7], since SDN switches are aware of the authentication process, securely share the information of SDN hosts using BC, and also ensure fault-tolerant, immutability, and a level of decentralization for SDN's data plane based on BC-smart contracts.

CHAPTER IV

HOSTSEC

The security for end user equipment (hosts) is a main concern in SDNs since it affects directly the SDN controller and also other hosts connected to the SDN network. In this chapter, we utilize BC technology to provide an authentication framework that takes into account host/switch manipulations. To this end, we proposed HostSec framework to provide mutual authentication for (a) host-controller communication and (b) host-host communication. This also implies providing (c) Packet-In/Packet-Out authentication mechanism to ensure that the data in these two critical OpenFlow events are sent from verifiable SDN hosts/controllers. The main host-related attacks in SDN paradigm are shown in Figure 41.

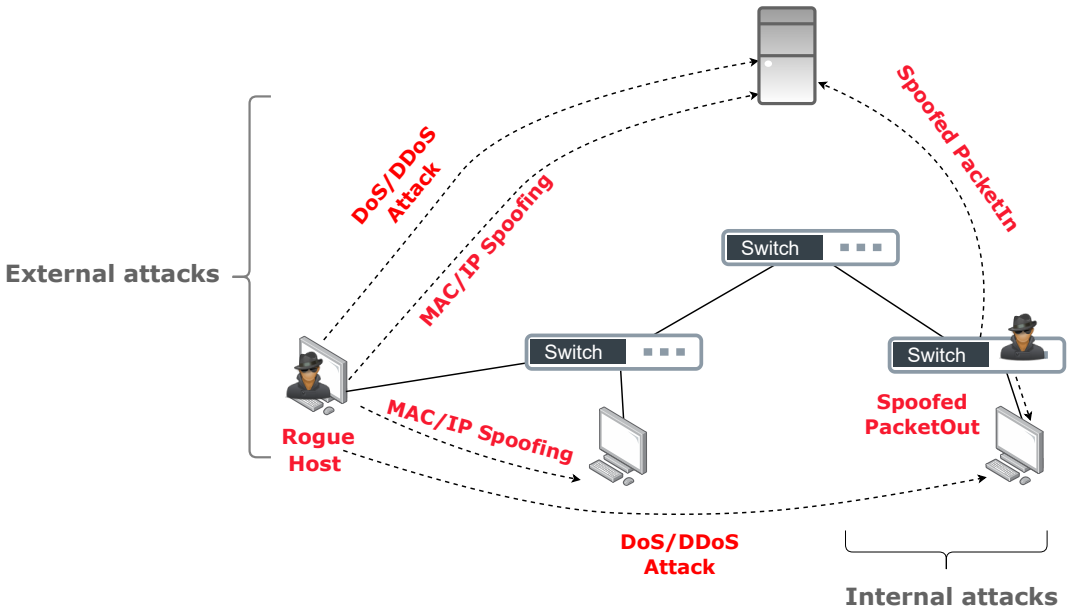


Figure 41: Host-related attacks in SDN data plane (Adapted from [3]).

DoS/Distributed DoS attacks can be launched against SDN components, where attackers may also spoof the MAC/IP addresses of other hosts and send spoofed packets towards the SDN controller or other hosts in the SDN network [79, 72]. Attackers can also generate Packet-In attacks towards the controller side [9, 79]. Consequently, different attacks target main components in the SDN network such as controllers, and hosts. Our contributions in this chapter are:

- We provide, HostSec, a blockchain-based framework that supports mutual host-controller, Packet-In/Packet-Out, and host-host authentication. SDN hosts and controller utilize lattice-based signatures and Key Encapsulation Methods (KEM) to provide post quantum resiliency for SDN components.
- We used AVISPA tool [6] to provide formal security verification for our proposed authentication mechanisms.
- We implement HostSec and report our experimental results based on a combination of Mininet emulator, Ryu SDN controller [62], Open vSwitch (OVS) [80], and Hyperledger Fabric [63], where Open Quantum Safe Library (liboqs) [75] is utilized for implementing lattice-based signatures and Key Encapsulation Methods (KEMs) to provide protection against quantum adversaries.

HostSec framework not only considers authentication to achieve more secure host-controller communication but also it provides secure key exchange by taking into account quantum adversaries. HostSec also considers potential data/packet manipulations through spoofed Packet-In/Packet-Out messages.

Furthermore, HostSec framework utilizes blockchain to improve the security of layer 2 and layer 3 in the SDN network and builds a novel host-host authentication approach on top of secure BC-enabled Address Resolution Protocol (ARP) and Identity Resolution Protocol (IRP).

4.1 Proposed Authentication Framework

In this section, we describe our threat model as well as proposed authentication framework. As we mentioned previously, our framework focuses on host-related threats in the SDN paradigm. To address these threats, we need to consider all host communications in SDN, which can be mainly related to host-controller communication or host-host communication. Our system aims to provide an acceptable level of fault tolerant, immutability, and security using blockchain characteristics. To this end, we utilize blockchain (BC) smart contracts to add the required level of decentralization to the authentication process. BC smart contracts allow only authorized SDN entities to interact with BC through attribute-based access control (ABAC).

4.1.1 Threat Model

Our threat model considers malicious SDN data and control planes. We assume that the attacker does not possess any cryptographic keys associated with any SDN nodes or BC clients, and the signature scheme and Certificate Authority (CA) are also secure (i.e., attacker cannot forge a signature without obtaining the secret key). We assume a Dolev-Yao attacker [81]. In other words, the attacker has full control over the transmission channel.

4.1.2 The Main Components in HostSec Framework

As shown in Figure 42, HostSec consists of four main components which are:

- HostSec Client: runs on the corresponding SDN host and provides quantum-resistant mutual authentication and key establishment with the HostSec controller as well as other SDN hosts. It also supports quantum-resistant mutual Packet-In/Packet-Out authentication. It also provides quantum-resistant Address Resolution Protocol (ARP), Identity Resolution Protocol (IRP) as well as HostSec protected UDP and TCP. In addition, HostSec establishes a secure communication channel with the SDN controller that allows sending private data and authenticated Packet-In/Packet-Out messages.

- HostSec-enabled SDN Switch: The switch plays an active role to defend against malicious hosts and reduces the load on the SDN controller using Packet-In manager component.

- HostSec Smart Contract: provides decentralization for the authentication process in HostSec framework. It includes the public keys and certificates of related SDN hosts, switches and SDN controllers. The contract verifies transactions submitted from HostSec-enabled SDN switches. Main functionalities includes: (i) registration for SDN components, (ii) secure binding of SDN hosts to related switches, (iii) secure ARP, IRP to protect layer 2 and 3 of SDN network, which enables secure host-host authentication, and (iv) secure flow match search to support Packet-In/Packet-Out authentication.

- HostSec-enabled SDN Controller: responsible for authenticating SDN hosts and related Packet-In messages. The controller allows post quantum secure authentication.

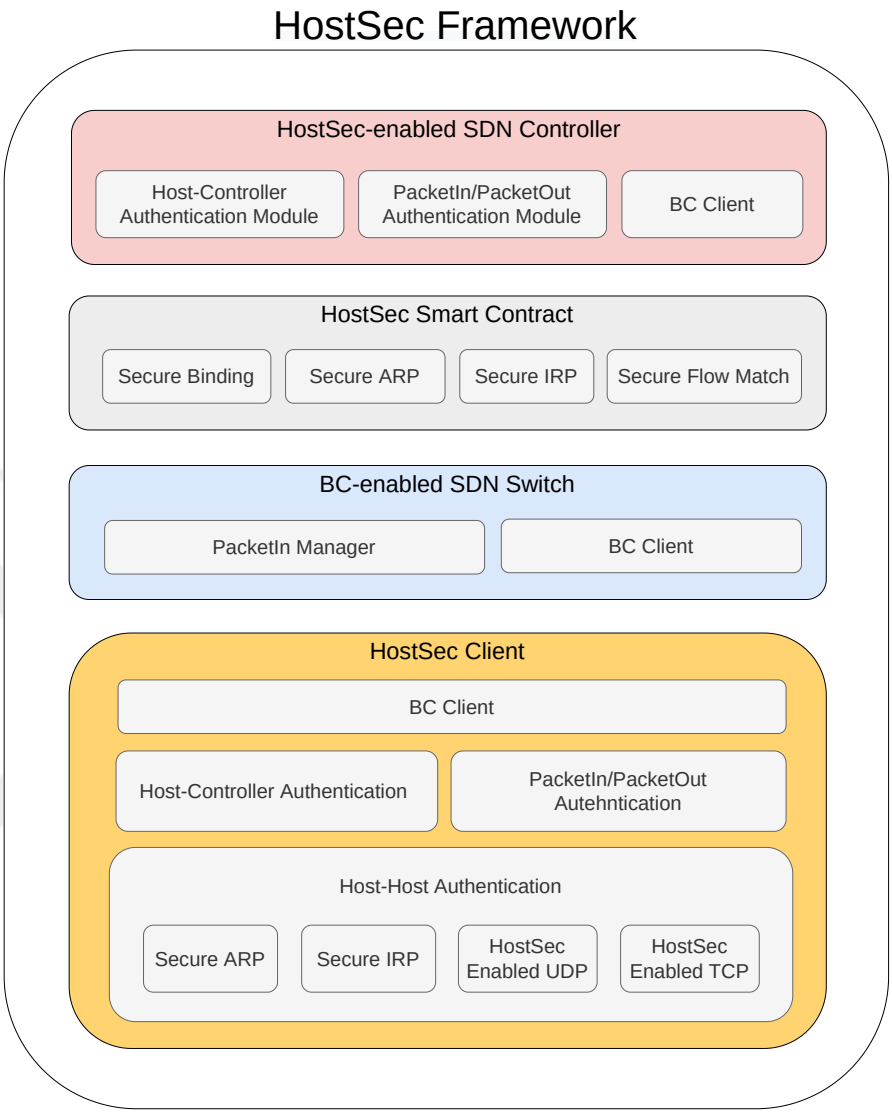


Figure 42: Main components of HostSec framework

4.1.3 Mutual Host-Controller Authentication

We utilize lattice-based signatures and Key Encapsulation Methods (KEMs) to provide quantum-safe [82] authentication and key exchange among SDN hosts and controllers, where we choose Dilithium3 [67] as our signature scheme and Kyber768 [69] as our KEM scheme. The public keys of related SDN hosts, switches and SDN controllers are added to the corresponding authentication smart contract by a new

transaction submitted by the corresponding certificate authority or domain administrator, which also has a certificate issued by the same certificate authority. We consider hybrid signatures when interacting with BC depending on the client operation (i.e., write or read). A write operation, which is highly critical in any BC system since it modifies the BC ledger, requires post-quantum signatures using Dilithium3 [67] scheme. Whereas a read operation is performed using a classical algorithm such as Elliptic Curve Digital Signature Algorithm (ECDSA) [64] since it does not modify the BC ledger. Figure 43 depicts the main steps in our host-controller authentication protocol.

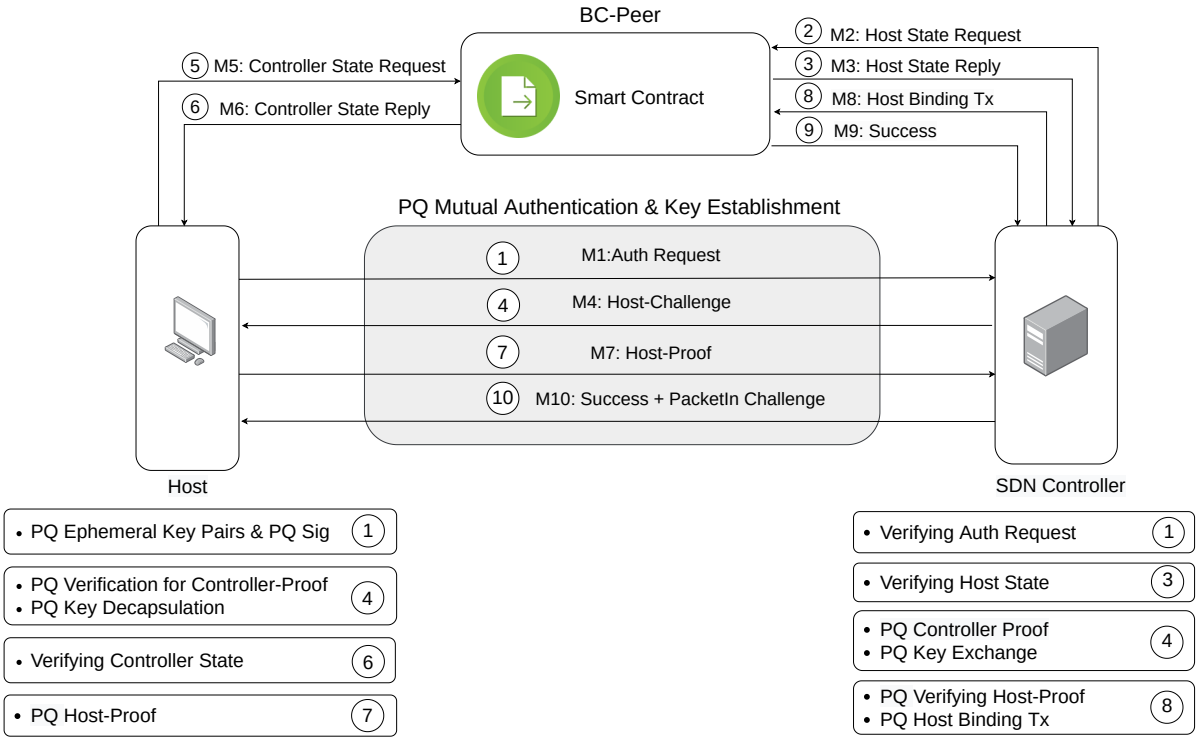


Figure 43: Overview of our proposed BC-enabled host-controller authentication

Host-Controller Authentication Steps

In step (1), the host sends a new signed $Auth\_Request$ to the SDN controller, which consists of the host's public key ($Host_1\_PubK$), Nonce and newly generated post quantum (PQ) ephemeral public key ($Host_1\_EPubK$) signed by the private key of the corresponding host ($Host_1\_Sk$).

In step (2), the controller will query the host information recorded on the smart contract, which includes registration information signed with PQ key of the corresponding admin.

In step (3), the contract replies with the registration information.

In step (4), the controller verifies the host information and the received nonce and its ephemeral public key and then signs the host challenge and generates a new signed challenge along with its certificate and a new secret key generated based on PQ KEM, which will be used in the next Packet-In messages. Then, send them to the corresponding host.

In step (5), the host will query controller information stored on the contract.

In step (6), the contract replies with the registration information.

In step (7), once the certificate and the signature are verified, the host will decapsulates the secret key and generates a nonce and a new host proof message, which includes its signature over the controller nonce and the new nonce.

In step (8), the controller verifies the host proof and verifies the signature of the corresponding host. Then, the controller sends a new transaction to bind the host to a specific switch.

In step (9), the contract returns a successful binding message.

In step (10), the controller sends a successful authentication message along with a new challenge for next Packet-In message.

A detailed description for the authentication steps is shown in Figure 44.

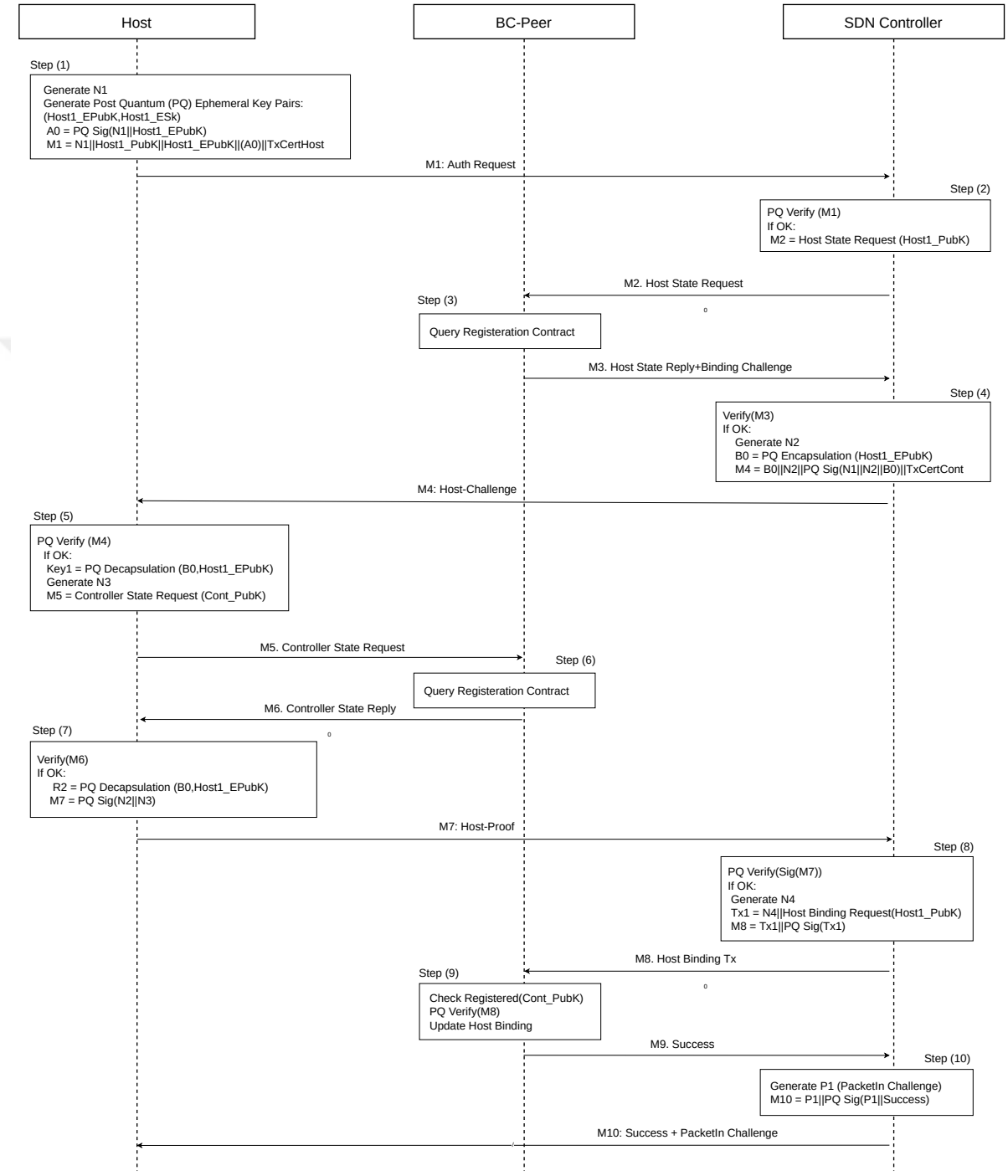


Figure 44: Detailed steps of our proposed BC-enabled host-controller authentication

Formal Security Verification Using AVISPA Tool We used AVISPA [6] to verify the security goals of our protocol. The results demonstrated that HostSec is safe under OFMC and CL-AtSe backends.

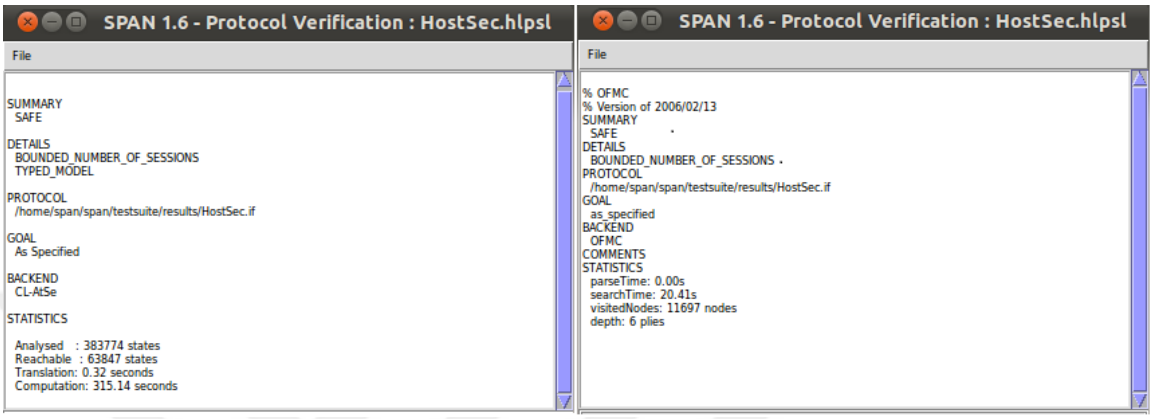


Figure 45: Verification results using AVISPA tool [6] under OFMC and CL-AtSe backends.

4.1.4 Packet-In/Packet-Out Authentication

Packet-In/Packet-Out messages are sent by the corresponding SDN switch, which can manipulate/fabricate these messages in order to launch attacks against the SDN controller or deceive the SDN host with a fabricated controller message. We assume that Packet-Out messages will be sent to the same host that resulted in sending a new Packet-In. AuthFlow [7], cannot detect switch-based Packet-In attacks since the switch is considered as a trusted component and it does not include any authentication method after the host is binded to the corresponding port other than MAC/Port pairs. To address this issue, it is important to verify and authenticate each message sent to the SDN controller and/or sent to the host through the SDN switch. We rely on BC smart contracts to store flows of the corresponding SDN switch on BC where the host must verify that there is no match for the required flow in its associated switch before sending its signed flow request. The Packet-In manager component detects malicious hosts and switches before sending the message to the SDN controller. Packet-In manager represents a data plane defense approach against malicious SDN

hosts and switches, which illegitimately generate spoofed packets that result in new Packet-In events. The Packet-In manager verifies public keys of SDN hosts based on the registration smart contract. In addition, it will verify the host based on a new challenge and signature. The Packet-In manager also receives its own version of the flow tables of the corresponding switch in order to detect malicious SDN switches, where a malicious SDN switch may send malicious Packet-Out messages even when a match exists in order to launch attacks against the SDN controller. Figure 46 shows the main steps in our Packet-In/Packet-Out authentication protocol.

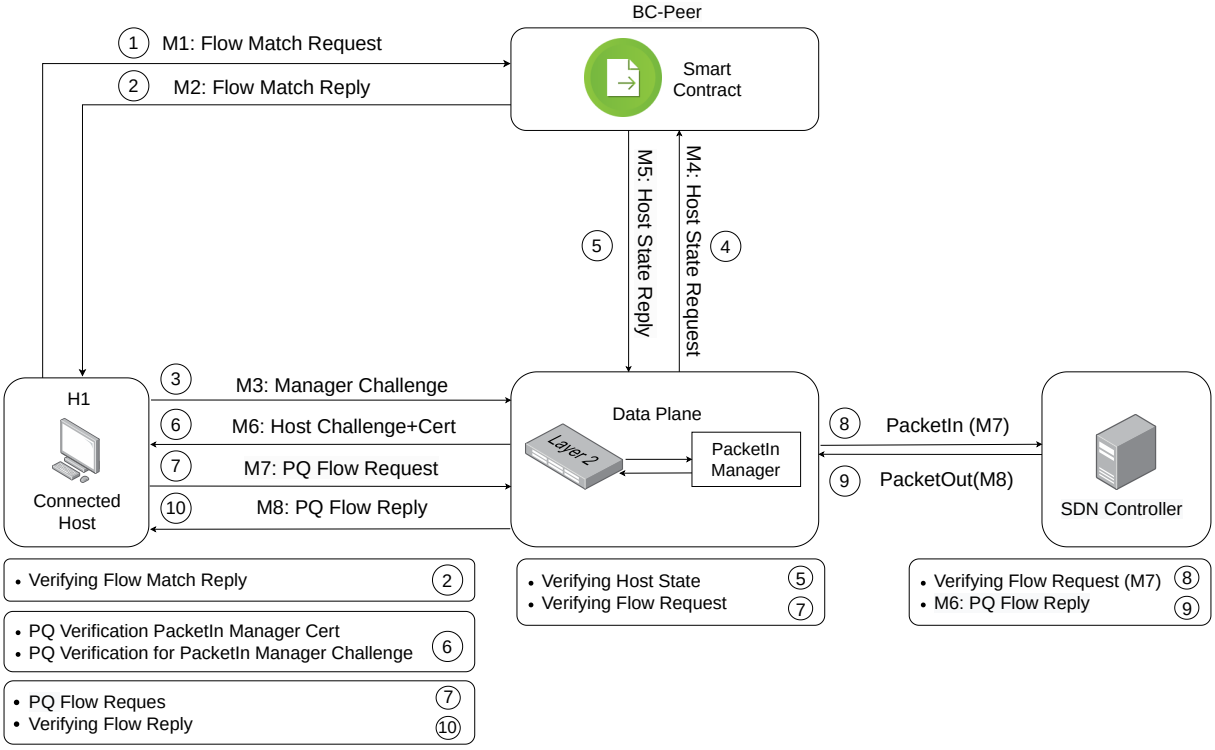


Figure 46: Overview of our proposed BC-enabled Packet-In/Packet-Out authentication

Packet-In/Packet-Out Authentication Steps

In step (1), the host sends a new *Flow_Match_Request* that includes information related to requested MAC/IP/Service to the corresponding smart contract.

In step (2), the contract will check if a match is found and return the result to the corerspoding SDN host.

In step (3), if no match is found, the host generates and signs a new challenge to be sent to the corresponding Packet-In manager.

In step (4), the Packet-In manager will use its own copy of flow table to verify that no match is found and then will query the host information recorded on the smart contract.

In step (5), the contract replies with the registration information, which allows the Packet-In manager to verify the host.

In step (6), the Packet-In manager will generate a new signed challenge and sent it, along with its certificate, to the corresponding host.

In step (7), the host verifies the Packet-In manager response. Then, it will generate a new nonce and a new *Flow_Request* signed and submitted to the Packet-In manager. Hash-based message authentication code (HMAC) is also appended to the message derived from the previously established secret key to prove that the host was authenticated before.

In step (8), the Packet-In manager verifies received *Flow_Request*. Once verified, a new Packet-In message will be sent to the corresponding controller.

In step (9), the controller verifies *Flow_Request* received from the host. Once verified, a new Packet-Out message will be sent to the corresponding switch.

In step (10), the switch will forward the Packet-Out message to the corresponding host, which verifies the controller signature.

A detailed description for the authentication steps is shown in Figure 47.

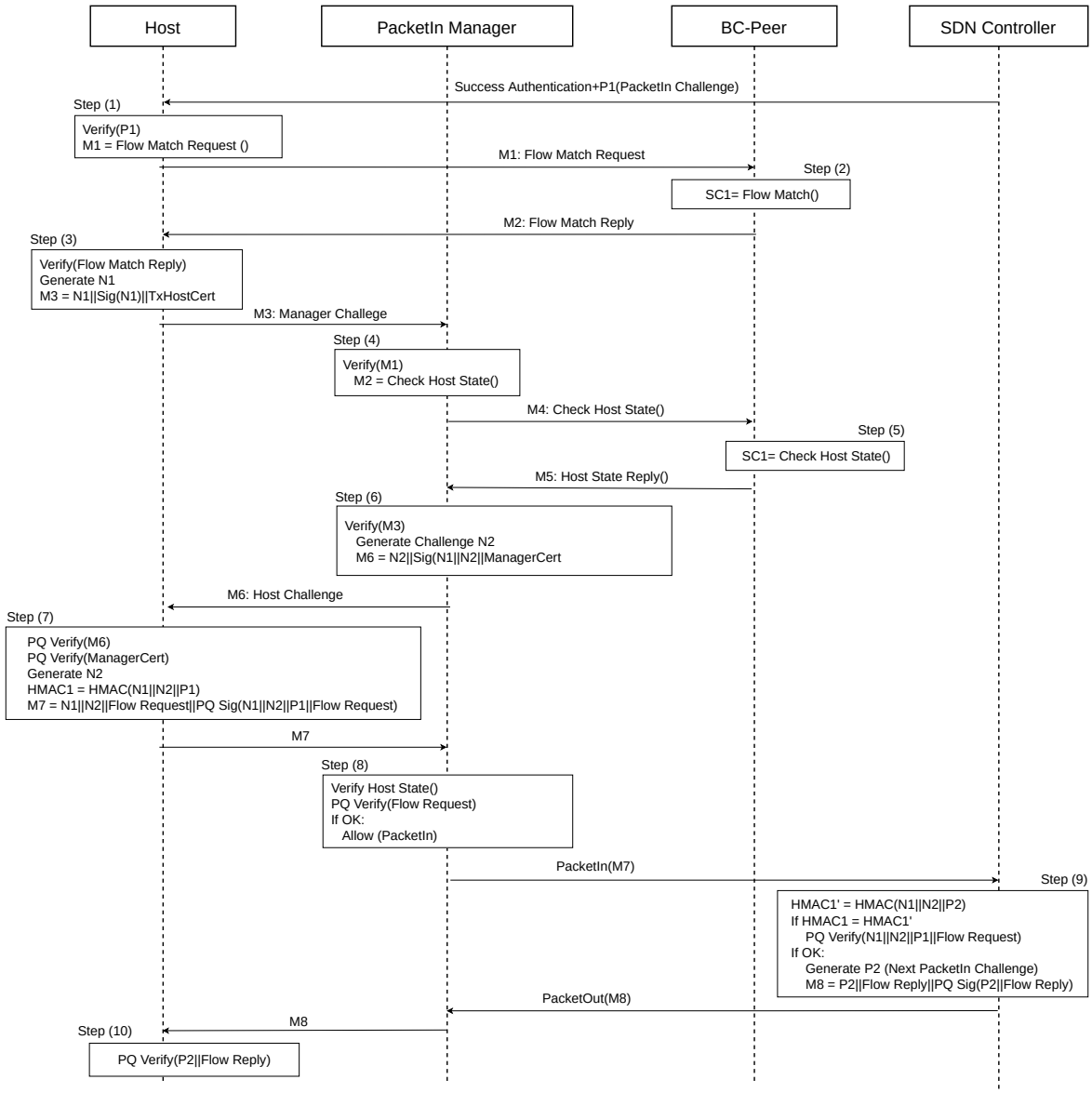


Figure 47: Detailed steps of our proposed BC-enabled Packet-In/Packet-Out authentication method

4.1.5 Mutual Host-Host Authentication

Before discussing our proposed host-host authentication protocol, first we introduce IP assignment, secure ARP and IRP in HostSec framework. Secure ARP and IRP protocols are the main building blocks to achieve a verifiable identity for layer 2 and layer 3 in the SDN network.

IP Assignment in HostSec For static networks, each IP address is assigned during the registration stage, where the network administrator, generates a new signed transaction that includes the MAC address, and the IP address of the corresponding host. The transaction is sent to the corresponding smart contract. Whereas for dynamic networks, the IP addresses are assigned using transactions signed by registered Dynamic Host Configuration Protocol (DHCP) server. The host, on the other hand, can verify replies received from BC smart contract.

In HostSec, each host has two types of records in the blockchain network. The first type includes a verifiable MAC/PublicKey pair (layer 2 identities). Whereas the second type includes a verifiable IP/MAC pair (layer 3 identities). Note that decoupling the identities allows our protocol to be consistent with network layers. Layer 2 identities ensure that a certain MAC address is associated with a specific public key, whereas layer 3 identities ensure that a certain IP address is associated with a specific MAC address.

Layer 2 and Layer 3 Identities

$$Layer\ 2\ Id = MAC||Pub_k||Sig(MAC||Pub_k)$$

$$Layer\ 3\ Id\ (Static) = IP||MAC||ExpTime||Tx_{Admin}$$

$$Layer3\ Id\ (Dynamic) = IP||MAC||ExpTime||Sig(MAC||IP||ExpTime)$$

BC-based Secure ARP in HostSec Lack of authentication mechanisms in traditional Address Resolution Protocol (ARP), opens the door for spoofing attacks among communication parties. After a successful authentication using HostSec framework for both the sender and receiver, each host quires the blockchain through BC-enabled

SDN switches to obtain a verifiable MAC address associated with a specific IP address. Each ARP response can be verified by the corresponding host. The reply is signed as follows

Exchanged ARP Messages

$$ARP\ Request = OpCode||IP$$

$$ARP\ Reply(Static) = OpCode||MAC||ExpTime||Tx_{Admin}$$

$$ARP\ Reply(Dynamic) = OpCode||MAC||ExpTime||Sig(MAC||IP||ExpTime)$$

Identity Resolution Protocol (IRP) in HostSec Note that ARP protocol allows only establishing IP/MAC tables/caches for SDN data plane. In order to provide authentication between SDN hosts, each host still needs to learn the layer 2 identity of the sender host in order to ensure that a particular packet is sent by the corresponding host. We refer to this process as Identity Resolution Protocol (IRP). After establishing the correct ARP tables, the host obtains verifiable layer 2 identities from our blockchain network.

Exchanged Identity Messages

$$ID\ Request = OpCode||MAC$$

$$ID\ Reply = OpCode||Pub_k||Tx_{Admin}$$

The host verifies the transaction that includes MAC/PublicKey pair signed by the corresponding certificate authority or domain administrator.

Host-Host Authentication We build our host-host authentication protocol on top of our secure ARP/IRP protocols. We use HostSec framework to provide protection for UDP and TCP protocols. Figure 48 shows the main steps in our host-host authentication protocol.

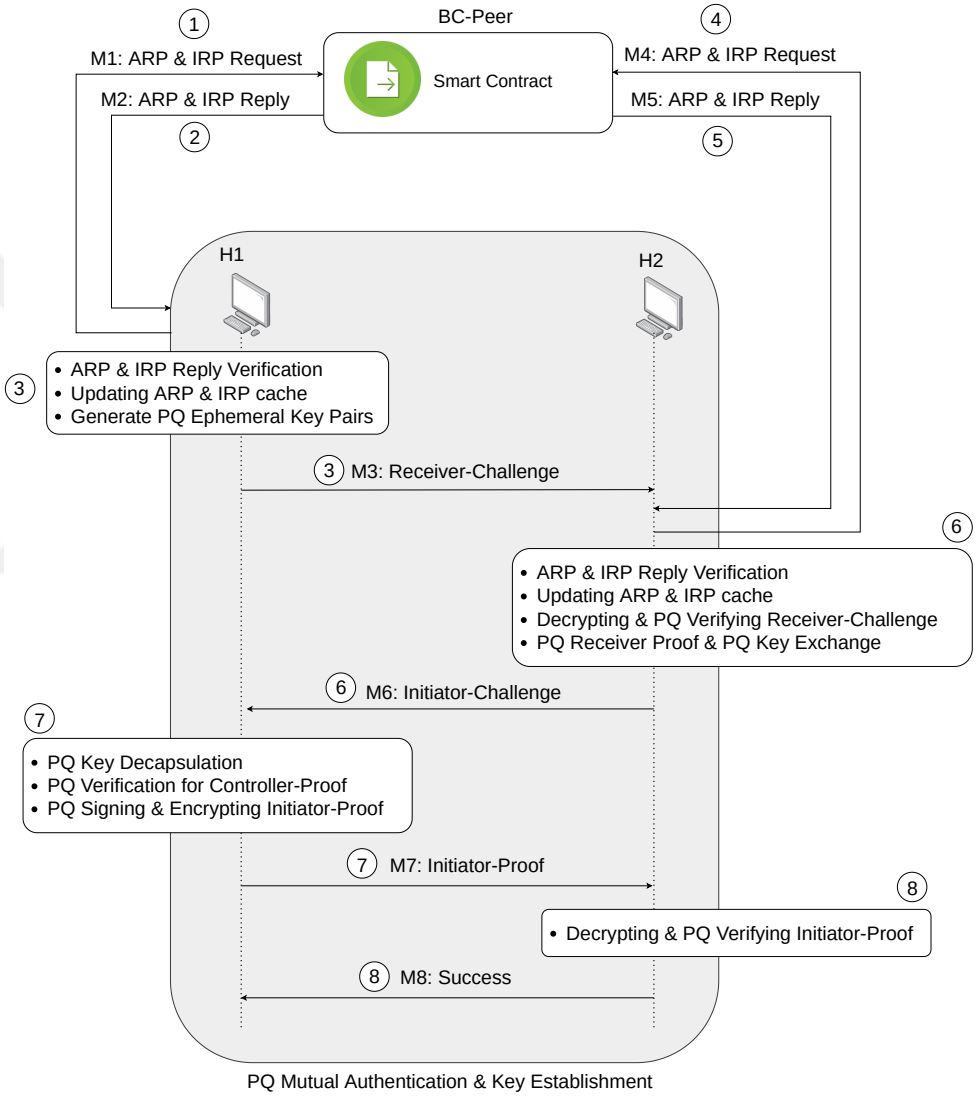


Figure 48: Overview of our proposed BC-enabled host-host authentication

Host-Host Authentication Steps

In step (1), the host sends new ARP and IRP requests to the smart contract.

In step (2), the contract retrieves the requested records and sends ARP and IRP reply messages, which also includes PQ signature over each record. Reply messages are sent to the corresponding host.

In step (3), the host verifies and updates its ARP and ID cache. Then, it will generate a new ephemeral key pair, a nonce value (N_1). Ephemeral public key along with nonce value are signed and submitted to the corresponding host.

In step (4), the receiver host sends ARP and IRP requests to the smart contract in order to verify layer 2 and layer 3 identities of the initiator host.

In step (5), the contract returns ARP and IPR reply messages to the initiator host.

In step (6), the host updates its ARP and ID cache. Then, it needs to verify the ephemeral key of the initiator host. Once verified, the host generates a nonce value (N_2) and shared secret encapsulated based on PQ KEM method. Both the nonce and encapsulated shared secret are signed and sent to the corresponding initiator host.

In step (7), the initiator host verifies the signature and decapsulates the shared secret. Then, signs $N_1||N_2$ and encrypts the signed message using the shared secret. Next, the encrypted message is sent to the receiver host.

Finally, in step (8), the receiver host decrypts the message and verifies the signature. Once verified, success message is sent to the initiator host.

A detailed description for the authentication steps is shown in Figure 49.

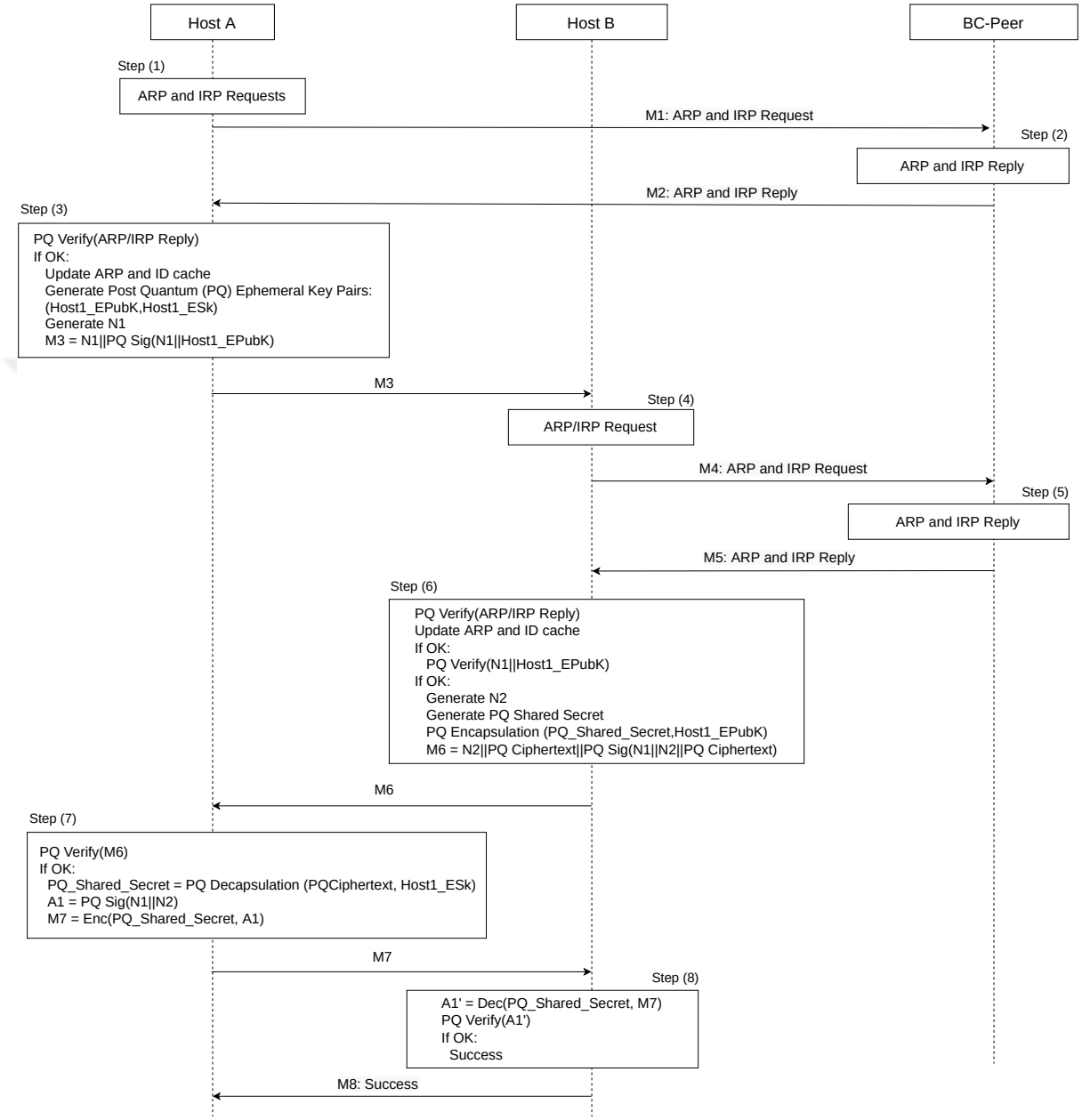


Figure 49: Detailed steps of our proposed BC-enabled host-host authentication method

4.2 Experimental Results

In this section, we provide detailed experiments that demonstrate the applicability and feasibility of our protocol along with comparisons with existing solutions. The experimental work is conducted on a physical machine with Xubuntu 20.04 OS. We use liboqs library [75] from the Open Quantum Safe project to enhance the security of host authentication against quantum adversaries. As an SDN controller, we use Ryu [62]. We use Hyperledger Fabric [63] for our BC part. All communications with BC nodes are secured through TLS protocol. In addition, BC nodes include a PQ verification module to verify PQ signatures based on the liboqs library. The details of our experimental setup are given in Table 3. Our consortium BC part includes 2 organizations and consists of 4 peers in total, 2 peers for each, in addition to the orderer peer which is responsible for determining the order of BC transactions [55]. We utilize Mininet to emulate our SDN network using single topology. We compare the latency of HostSec with AuthFlow [7]. We measure the latency of host-controller, host-host, and Packet-In/Packet-Out authentication approaches. Then, we report the obtained results.

Table 3: Experimental setup

#	Details
CPU	Intel i5-1135G7
RAM	16 GB
Operating system	Xubuntu 20.04
Emulator	Mininet 2.3.0
SDN controller	Ryu 4.34
SDN switch	OVS 2.15.1
Blockchain	Hyperledger Fabric 1.4
Number of BC organizations	2 organizations, 2 peers per organization
Block size	100 transaction per block
Batch timeout	1 second (1 block per second)

4.2.1 Latency of HostSec

First, we measure the average latency for host-controller and Packet-In authentication for our proposed system. We also compare HostSec with AuthFlow [7]. Figure 50, shows that HostSec has higher latency compared to AuthFlow [7]. AuthFlow [7] provides host authentication based on Remote Authentication Dial-In User Service (RADIUS) authentication server, which utilizes MS-CHAPv2 (Microsoft Challenge-Handshake Authentication Protocol). AuthFlow [7] does not provide any secure channel between the host and SDN controller. Therefore, it cannot detect/handle malicious SDN switches. HostSec has higher latency compared to AuthFlow [7] as the host needs to interact with BC smart contract. The advantage provided by the decentralized approach is that both the SDN host and the SDN controller can detect malicious SDN switches. SDN switches are also able to detect spoofed messages received from malicious SDN hosts.

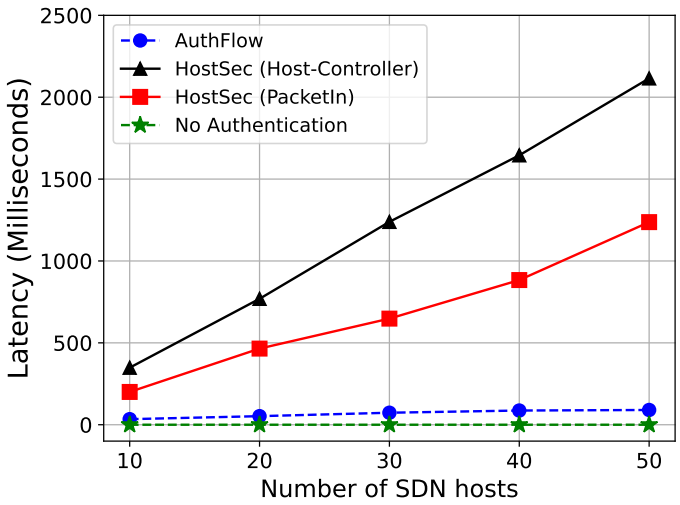


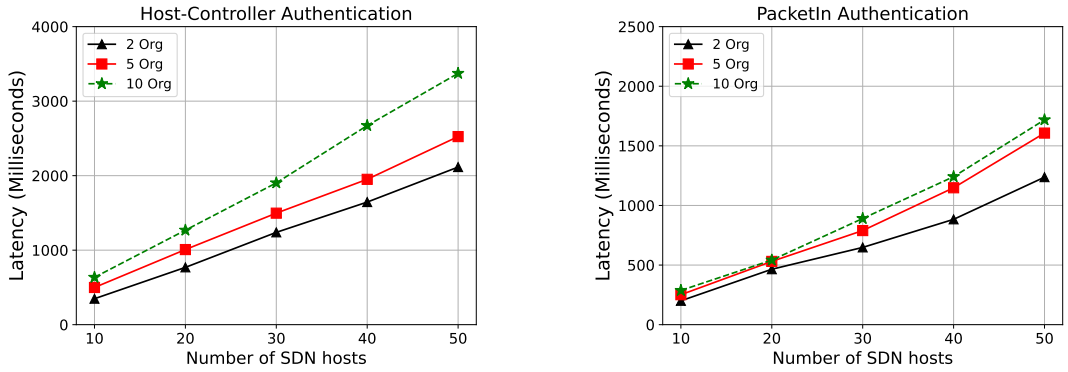
Figure 50: Average latency for HostSec

We also measure the latency for our proposed host-host authentication approach. We used HostSec framework to provide protection for UDP and TCP protocols. HostSec-enabled TCP has slightly higher latency (61.21 milliseconds) compared to

HostSec-enabled UDP (59.04 milliseconds). The advantage provided by the decentralized approach is that malicious events from host and switches can identified during and also after the authentication process.

4.2.2 The Impact of Increasing The Number of Organizations

We investigated the impact of increasing the number of organizations in our BC network, where we considered three cases. The first case consists of 2 organizations with two peers, representing our base model. In the second case, we included five organizations each of which includes two peers. In the third case, we included ten organizations with two peers for each organization. The obtained results are shown in Figure 51. For the host-controller authentication, in the first and second cases the average latency remains under 3s. However, in third case the latency increases above 3s when the number of SDN hosts is higher than 40 host. For the Packet-In authentication, in the first and second cases the average latency remains under 2s. For the Packet-In authentication, the average latency remains under 2s for all cases.



(a) Host-Controller Authentication (b) Packet-In Autentication

Figure 51: Average latency of HostSec with different number of organizations

We also measure the latency for our proposed host-host authentication approach with different number of organizations. As shown in Table 4, the latency slightly increases compared to the first case. For HostSec-enabled TCP, the latency increased,

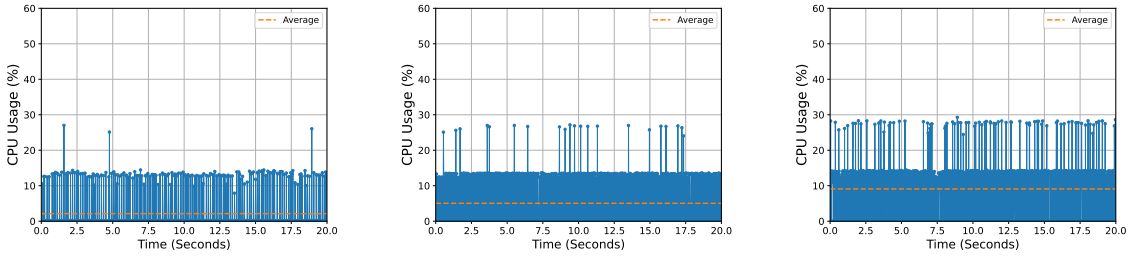
compared to the first case, by 14.24% and 17.85% for the second and third cases, respectively. For HostSec-enabled UDP, the latency increased, compared with the first case, by 11.63% and 16.37% for the second and third cases, respectively.

	2 Org (Millisecond)	5 Org (Millisecond)	10 Org (Millisecond)
HostSec-enabled TCP	61.21	69.93	72.14
HostSec-enabled UDP	59.04	65.91	68.71

Table 4: Average latency of Host-Host authentication with different number of organizations

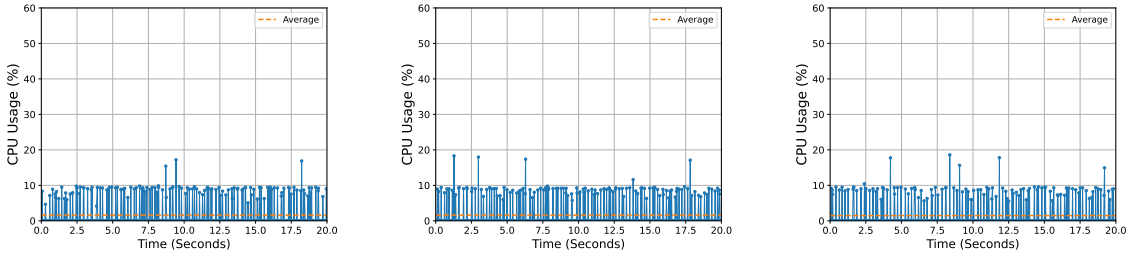
4.2.3 CPU Utilization

In this section, we study CPU utilization for the defense components in the SDN network for both AuthFlow [7] and HostSec framework. We report CPU utilization during spoofing attacks launched from SDN hosts. At each iteration, the attacker randomly chooses source MAC addresses of authenticated SDN hosts and sends packets



(a) Attack Rate = 100 (b) Attack Rate = 500 (c) Attack Rate = 1000

Figure 52: CPU utilization for the SDN controller with AuthFlow [7] during host-based spoofed Packet-In attack



(a) Attack Rate = 100 (b) Attack Rate = 500 (c) Attack Rate = 1000

Figure 53: CPU utilization for the Packet-In manager of HostSec during host-based spoofed Packet-In attack

with random destination MAC addresses in order to generate new Packet-In events towards the SDN controller. We compare our system with AuthFlow [7] which detects these attacks at the control plane based on MAC/Port pairs as long as the switch is not sending spoofed Packet-In messages. HostSec can detected these attacks at the data plane using its Packet-In manager, which decreases the load on the SDN controller. Unlike AuthFlow [7], our Packet-In manager can detect these attacks even when the switch is sending spoofed messages. Figure 52 shows CPU utilization for the SDN controller with AuthFlow [7] during this attack. The results reveal that CPU utilization increases according to the attack rate. Figure 53 shows CPU utilization for the Packet-In manager of HostSec during this attack. For low attack rate (i.e., 100 packets per second), the results show that Packet-In manager reduces the CPU utilization by 24.19%. Whereas for moderate attack rate (i.e., 500 packets per second), CPU utilization is reduced by 68.11%. Finally, for high attack rate (i.e., 1000 packets per second), CPU utilization is reduced by 83.72%. This results suggest that deploying defense mechanisms at the data plane can be useful for protecting the SDN controller and reducing the CPU utilization.

4.3 Summary

In this chapter, we introduced HostSec framework for enhancing the security of SDN networks. HostSec supports mutual host-controller, Packet-In/Packet-Out and host-host authentication based on a decentralized blockchain approach. We utilized lattice-based signatures and KEMs to provide protection against quantum adversaries. We also provided secure ARP and IRP protocols to protect layer 2 and layer 3 in the SDN network. HostSec ensured protection against single point of failure and malicious SDN components. In addition to distributed immutable state provided by utilizing blockchain technology. The results also revealed that HostSec was superior to AuthFlow [7] approach due to the ability to detect manipulation attacks and to

reduce the unnecessary load on the SDN controller. Overall, the experimental results presented a trade-off between improved security and lower latency.

CHAPTER V

SDN-API-SEC

In this chapter, we take advantage of the intrinsic characters of blockchain technology to protect critical SDN resources. We provide a flexible cross-domain access control method for the decision making components by redesigning the APIs of SDN through blockchain smart contracts. Distributed multi-domain SDN models were proposed to tackle the single point of failure problem and improve the performance of the SDN network [83]. Multi-domain SDN enables network applications to operate across different domains, which results in enhanced provisioning of required services in large scale SDN networks [83]. In multi-domain networks, each domain can be managed by a separate SDN controller. In this setting, a user U_i in domain A may request a service S_j provided by domain B [84, 18].

Cross-domain authorization can be used to handle this issue. In traditional solutions [85], domain B requires a centralized, trusted third party to authorize the user U_i since it does not have any direct relation with domain A [18]. The centralized solution causes a single point of failure and lacks transparency, which promotes Man In The Middle (MITM) attacks [17]. Once the trusted party is compromised, the attacker can control the whole network [18]. BC technology provides a decentralized, immutable, distributed ledger that does not require any third parties to be involved in the authorization process [86]. Consortium BCs are suitable for multi-domain SDN networks since each organization represents an SDN domain[3]. In SDN-API-Sec, we utilize consortium BC approach to achieve a decentralized cross-domain authorization mechanism that can be used for protecting SDN components from unauthorized access.

Our contributions are summarized below:

1. We propose, SDN-API-Sec, a flexible blockchain-based cross-domain approach that utilizes smart contracts to authorize SDN components and protect them from unauthorized entities using a consortium BC approach.

2. We use AVISPA [6] to provide formal security verification for our proposed authorization mechanism.

3. We implemented our system using a combination of Mininet emulator, Ryu SDN controller, Open vSwitch (OVS), and Hyperledger Fabric.

5.1 Proposed SDN-API-Sec

5.1.1 System Model in SDN-API-Sec

As shown in Figure 54, our system model represents a cross-domain network denoted as $D = [D_1, ..., D_\ell]$ managed by local SDN controllers where each domain D_i is

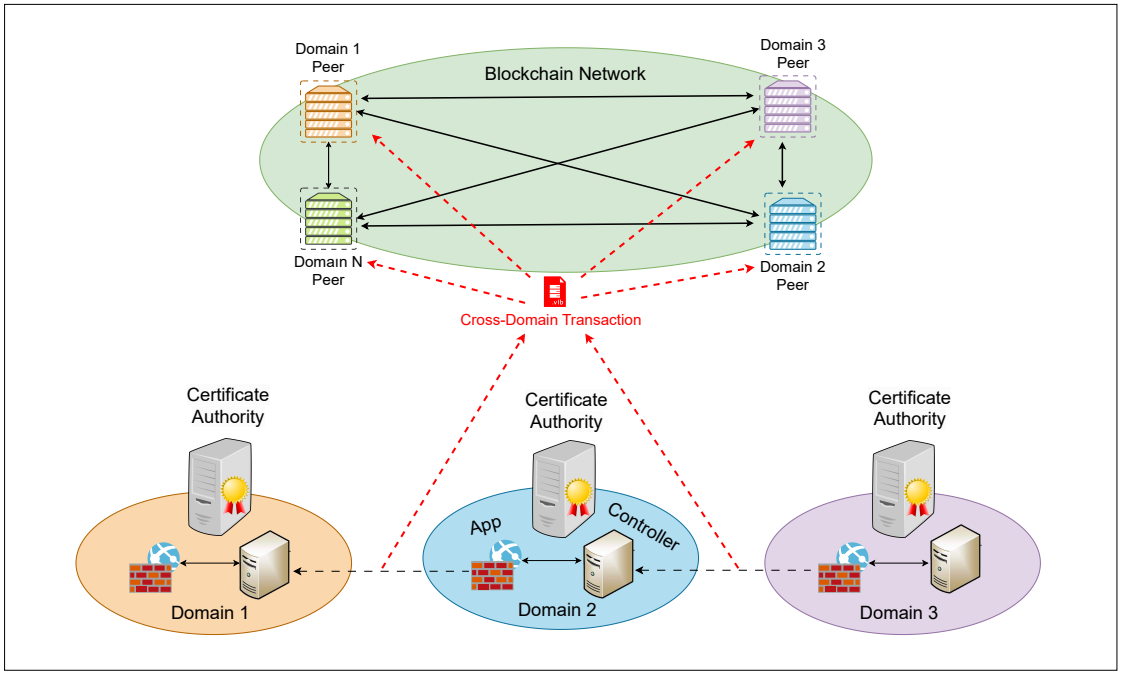


Figure 54: System model in SDN-API-Sec

managed by controller C_i. We consider a consortium BC in order to securely authorize the SDN components of over D. Each D_i has its own certificate authority CA_i, which is responsible for providing digital certificates, and delivering cryptographic keys for each SDN component. Each domain D_i includes a set of endorser peers, which are used to endorse the new transactions in the system.

5.1.2 The Main Components in SDN-API-Sec Framework

Inspired from [95, 96, 91], SDN-API-Sec includes components compatible with XACML[1].

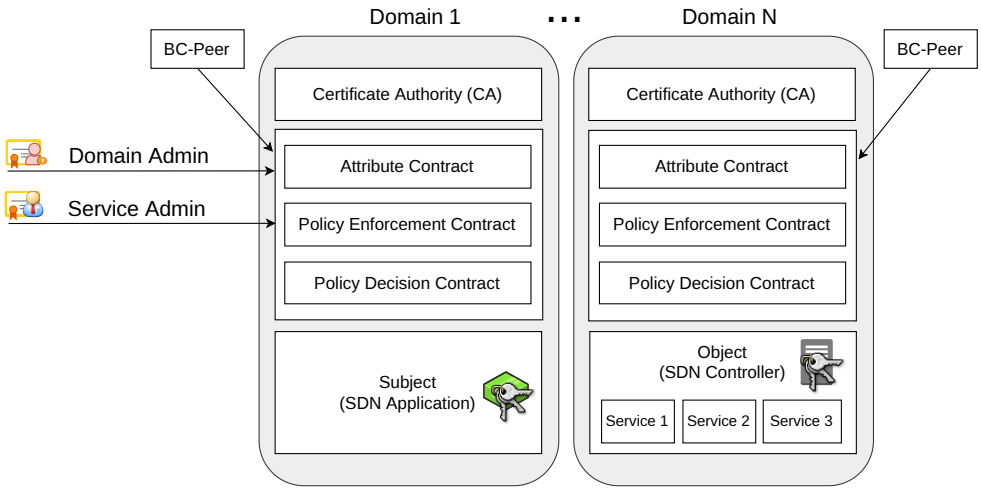


Figure 55: SDN-API-Sec components based on [95, 96, 91]

- BC-Peers: include the authorization smart contracts. BC-Peers are also responsible for verification and endorsement of transactions received from SDN components.
- Subject (e.g. SDN application): represents an entity requesting access to another entity in the SDN network.
- Object (e.g. SDN controller): represents an entity that provides an API that can be accessed by authorized entities.

[1]eXtensible Access Control Markup Language (XACML) [online]. Website https://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html [accessed 30 May 2023]

- Domain Admin: represents a (human) domain administrator, which adds the attributes of the subjects (e.g., SDN components) that belongs to its own domain. Domain administrator also adds different network services along with service administrators to the system.

- Network Service Admin: represents a (human) network service administrator, which adds access permissions and rules of its own service to the corresponding access contract. This allows a fine-grained access control for different network services provided by the same object.

- Attribute Contract: this contract includes attributes for each subject and object in the SDN network [95, 96, 91].

- Policy Enforcement Contract: this contract receives authorization requests from SDN components and then enforces authorization decisions received from the policy decision smart contract [95, 96, 91].

- Policy Decision Contract: this contract is used for making authorization decisions for subjects that belong to different SDN domains [95, 96, 91].

5.1.3 Authentication in SDN-API-Sec

Authentication is an essential step before starting the authorization process. In SDN-API-Sec, the authentication is based on our previous protocol CWT-DPA [3]. We also modified it slightly to send additional challenge to the corresponding subject. Note that our authorization protocol is also compatible with standard authentication protocols such as TLS.

5.1.4 Authorization in SDN-API-Sec

As shown in Figure 56 [95, 96, 91], the subject represents an SDN application, which aims to access a specific object (i.e. SDN controller) in the network.

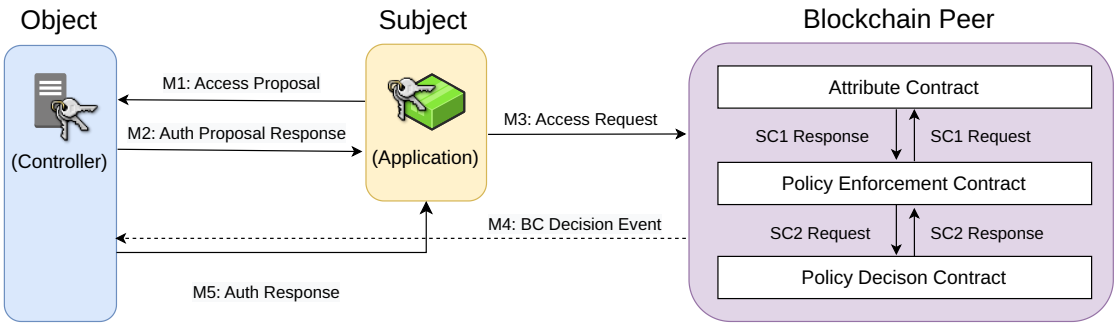


Figure 56: Overview of the authorization in SDN-API-Sec based on [95, 96, 91]

Authorization Steps in SDN-API-Sec

In step (1), the application sends an authorization proposal message M_1 to the SDN controller.

In step (2), the controller decrypts and verifies M_1 and then generates a new authorization proposal response M_2 that includes a new challenge and sends it to the SDN application.

In step (3), the application verifies M_2 and sends the required the access request message M_3 to the corresponding BC-peer.

In step (4), BC peer verifies the request and then calls the corresponding policy enforcement contract, which calls the attribute contract using SC_1 request. The attribute contract returns the attributes of the corresponding subject using SC_1 response. Then, the policy enforcement contract calls the policy decision contract using SC_2 request. The policy decision contract returns access decision using SC_2 response. Once the access is permitted, the policy enforcement contract generates a new BC event that includes the challenge sent by the corresponding SDN controller.

In step (5), the controller receives and verifies the BC decision event and then sends a new authorization response to the corresponding application.

In step (6), the application verifies the response received from the SDN controller [95, 96, 91].

A detailed description of our proposed authorization is shown in Figure 57.

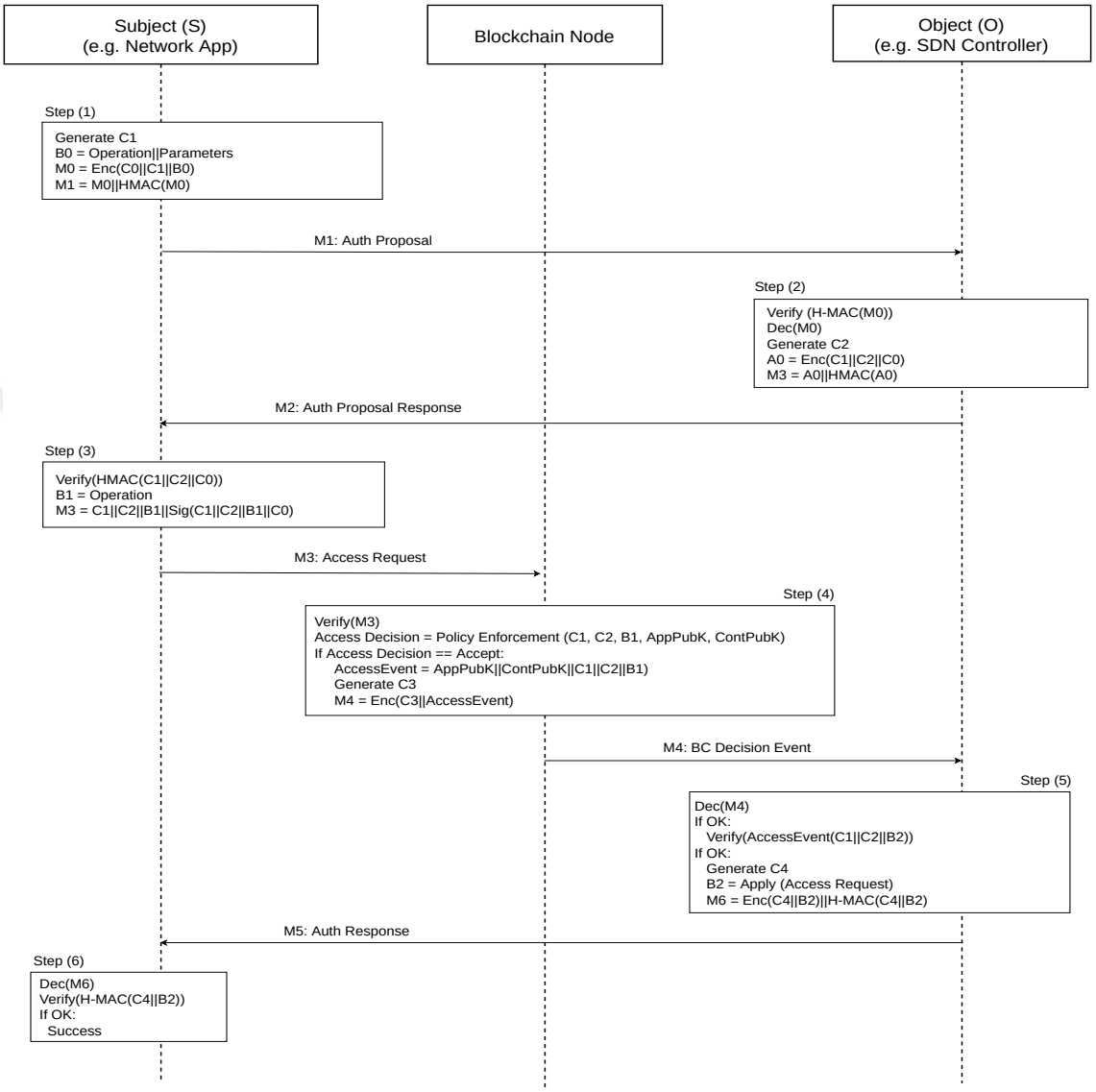


Figure 57: Detailed steps of authorization steps in SDN-API-Sec

5.1.5 Formal Security Verification Using AVISPA Tool

We utilize AVISPA [6] tool for validating the security properties of the proposed system. As shown in Figure 58, the results showed that SDN-API-Sec is safe under OFMC and CL-AtSe backends.

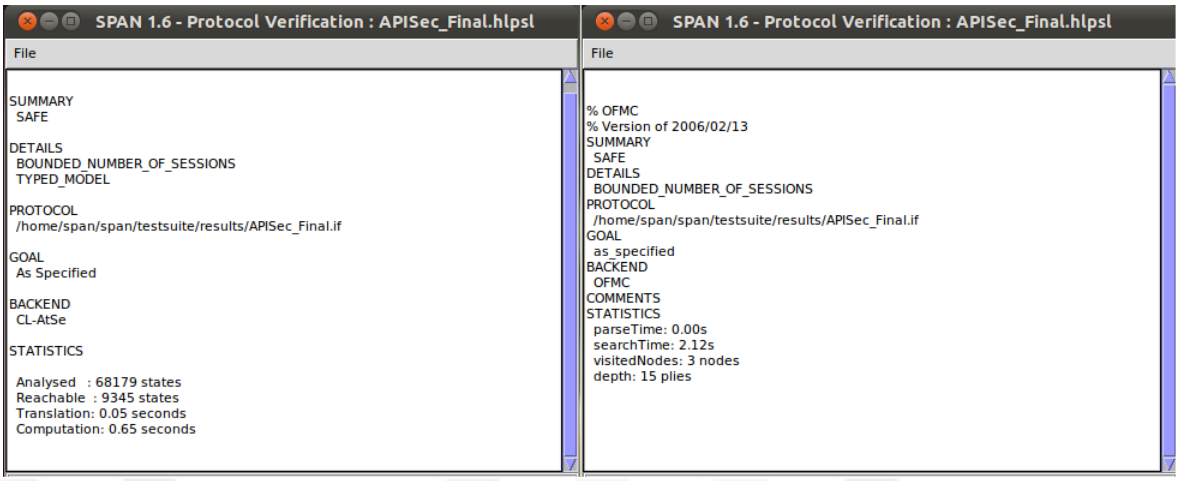


Figure 58: Verification results using AVISPA tool [6] under OFMC and CL-AtSe backends.

5.1.6 Permissions in SDN-API-Sec

Each permission in SDN-API-Sec consists of one or more access rules. Each access rule consists of subject attributes, which are used to evaluate the incoming requests. If all attributes, in a given request, are satisfied then the corresponding access rule applies to that request. The policy for permission is defined based on a set of access rules and a combining operator. The service administrator installs the access rules for its own domain. An example of access rules in SDN-API-Sec is shown in Listing 1. It is worth noting that access rules in SDN-API-Sec can be updated by the corresponding network service admin. The system does not require writing a new smart contract when the rules are updated.

Listing 5.1: Example of an access control policy in SDN-API-Sec.

```
// Combining Operator
COMRULE Deny-Unless-Permit
// Access Rules in SDN-API-Sec
OPUBKEY Key1, SERVICE Routing
ATTRIBUTE Domain,Type OPERATION Read
PERMISSION Permit WHERE Domain=1,Type=App
  ....
  ....
ATTRIBUTE Domain,Type OPERATION Write
PERMISSION Deny WHERE Domain=3,Type=Controller
```

The access rules are parsed using our parser which is written using Chevrotain[2] library for the lexing and parsing steps. The output of our parser is shown in Listening 2.

Listing 5.2: Example of a parsed rule in SDN-API-Sec.

```
//  Example of a parsed rule in SDN-API-Sec
//  OPUBKEY Key1, SERVICE Routing
//  ATTRIBUTE Domain,Type OPERATION Read
//  PERMISSION Permit WHERE Domain=1,Type=App
{
        "OPUBKEY": "Key1",
        "SERVICE": "Routing",
        "Attributes": [
                "Domain",
                "Type"
```

[2]Chevrotain Library [online]. Website https://chevrotain.io/docs/ [accessed 05 Feb 2023]

```
        ],
        "Operation": "Read",
        "Permission": "Permit",
        "MatchSet": [
                {
                        "attribute": "Domain",
                        "operator": "=",
                        "value": "1"
                },
                {
                        "attribute": "Type",
                        "operator": "=",
                        "value": "App"
                }
        ]
}
```


5.1.7 Detection of Conflict and Redundant Rules

Two rules R_i, R_j are inconsistent if all the attributes and values of rule R_i is shared with rule R_j and both have conflict decisions [87, 88]. When the rules are added to the BC, the smart contract ensures that no conflict exists between the rules submitted to the BC. In SDN-API-Sec, conflict detection is inspired from [88], which detects both explicit and implicit conflicts.

Explicit Conflicts

Explicit conflicts occur when the following conditions are met [88]:

1) All attributes from one rule is shared with another rule.

2) The values of the shared attributes intersect.

3) An overlap between affected permissions.

4) Decisions are distinct.

Implicit Conflicts

Implicit conflicts may occur for some access requests in which the corresponding attributes satisfy more than one rule, where their decisions are distinct [88]. To ensure that the system dos not include any implicit conflicts, there is a need to ensure the existence of non-overlapping attribute values [88].

Redundancy

Redundancy [87, 89] occurs when all the attributes and values of rule R_i is shared with rule R_j and both have the same decision [90]. When the rules are added to the policy decision contract, we check whether any conflict or redundant rules exist (see Function 3).

5.1.8 Rule Combining Operators in SDN-API-Sec

As we mentioned before, each access control policy in SDN-API-Sec contains a combining operator. We utilize Deny-unless-permit and Permit-unless-deny combining operators based on eXtensible Access Control Markup Language (XACML) [91, 92, 93]. The deny-unless-permit operator is used for the cases in which *Deny* has a higher priority over *Permit* [93]. The permit-unless-deny operator is used for the cases in which *Permit* has a higher priority over *Deny* [93].

5.1.9 Policy Decision Contract in SDN-API-Sec

The policy decision contract consists of the following functions: Domain administrator adds new service administrator using Function 1. Function 2, allows the service admin to add a new permission, related to a service offered by an object that belongs to its own domain, to the contract. The service admin adds access rules for a service offered by an object that belongs to its own domain using Function 3, where the rules are accepted only if they do not include any redundancy or conflicts. In Function 4, the contract takes access decision based on existing access rules. In Function 5, the contract evaluates the attributes for a given rule and subject.

Function 1: Add Service Admin Function

Input: $New\_Admin\_Pub_k$, $Object$, $Service$
if $caller.type == DomainAdmin$ **then**
  $ServiceAdmins[caller.Domain][Obj\_Pub_k][Service] = New\_Admin\_Pub_k$
else
  Reject;

Function 2: Add Permission Function

Input: $New\_Permission$, $Object$, $Service$
if $caller.type == DomainAdmin$ **then**
  $Permissions[caller.Domain][Obj\_Pub_k][Service] =$
   $Permissions[caller.Domain][Obj\_Pub_k][Service] \cup New\_Permission$
else
  Reject;

Function 3: Add Permission Access Rules Function with conflict detection [88] and redundancy detection [90]

Input: $Permission, Permission\_Access\_Rules(attributes, operators, values)$, CombiningOperator;
$BreakLoop = Conflict = Redundant = r3 = r4 = 0$;
if $caller.type == ServiceAdmin$ ***and*** $Permission \in Permissions[caller.Domain][Obj\_Pub_k][Service]$ **then**
  if $caller.Domain == New\_Access\_Rule.Subject.Domain$ **then**
    for $i = 0; i < (len(AccessRule)$ ***and*** $BreakLoop == 0)$; $i++$ **do**
      for $j = 0; j < len(AccessRule); j++$ **do**
        if $i == j$ **then continue**;
        $N\_Overlap = 0$; $N\_Distinct = 0$;
        $N\_Shared = len(SharedAttributes(R_i, R_j))$;
        if $N\_Shared == 0$ **then** $BreakLoop = 1$; **break** ▷ Implicit Conflict;
        foreach *Shared Attribute* $a_k \in (R_j \cap R_i)$ **do**
          if $(R_i[a_k].val \cap R_j[a_k].val) == \emptyset$ **then**
            $N\_Distinct++$; ▷ # of non-overlapping attribute values;
          else
            $N\_Overlap++$; ▷ # of overlapping attribute values;
      if $N\_Distinct == 0$ **then** $Conflict = 1$; **break** ▷ Implicit Conflict;
      if $N\_Overlap == 0$ **then continue** ▷ Conflict Free;
      if $N\_Shared == len(R_i) || N\_Shared == len(R_j)$ **then** $r3 = 1$;
      if $N\_Overlap == len(R_i) || N\_Overlap == len(R_j)$ **then** $r4 = 1$;
      if $decision(R_i) != decision(R_j)$ ***and*** $r_3 == true$ ***and*** $r_4 == true$ **then**
        $Conflict = 1$; **break** ▷ Explicit Conflict;
      else if $decision(R_i) == decision(R_j)$ ***and*** $r_3 == true$ ***and*** $r_4 == true$ **then** $Redundant = 1$; **break** ▷ Redundant Rule;
    if $!Conflict$ ***and*** $!Redundant$ **then**
      for *AccessRule* $R_i \in Permission\_Access\_Rules$ **do**
        $AccessRules[caller.Domain][Obj\_Pub_k][Service][Permission].append(R_i)$
      $Combine\_Operators[caller.Domain][Obj\_Pub_k][Service][Permission] = CombiningOperator$
    else
      Reject
  else
    Reject
else
  Reject

Function 4: Object Access Control Decision Function

```
Input: Object, Required_Operation
if Operation ∉ Operations[Subject] then
    return Reject
else
    CombiningOperator =
     Combining_Operators[Obj.Domain][Obj_Pub_k][Service][Required_Permission];

    foreach Rule
     R_i ∈ Access_Rules[Obj.Domain][Obj_Pub_k][Service][Required_Permission]
     do
        RP = RulePermission[R_i];
        foreach Attribute a_i ∈ R_i do
            r_1 = EvalAttribute(Subject[a_i], R_i[a].operator, R_i[a].val);
            if r_1 == false then
                continue
        if (RP == Deny and CombiningOperator == permit_unless_deny) then
            return Deny
        else if (RP == Permit and
         CombiningOperator == deny_unless_permit) then
            return Permit
    if (CombiningOperator == deny_unless_permit) then return Deny ;
    if (CombiningOperator == permit_unless_deny) then return Permit ;
```

Function 5: EvalAttribute

```
Input: Subject_Attribute.value, Operator, Rule_Attribute.value
if (Operator == "equal") then
return Subject_Attribute.value == Rule_Attribute.value ;
else if (Operator == "larger") then
    return Subject_Attribute.value > Rule_Attribute.value
else if (Operator == "smaller") then
    return Subject_Attribute.value < Rule_Attribute.value
else if (Condition) then ....;
else return Error ("Unknown Operator");
```

5.1.10 Policy Enforcement Contract in SDN-API-Sec

The policy enforcement contract consists of Function 6, in which the contract retrieves the attributes of the corresponding subject. Then, it calls the policy decision contract, which returns the access decision to the policy enforcement contract.

If the access is permitted and the application is not in blocked state, then the contract generates a new event that includes $AppPub_k, ContPub_k, Operation, Cont\_Challenge$.

Otherwise, the contract calculates the rejection rate of the corresponding application. If the rejection rate passes the threshold defined for the application level, then the application is blocked.

Function 6: Policy Enforcement

Input: $Cont\_Challenge$, $AppPub_k$, $ContPub_k$, $Operation$
$TotalRequests[AppPub_k]++$;
$AppAttributes = AttributeContract.GetAttributes(AppPub_k)$;
$AppLevel = AppAttributes.Level$;
$Res =$
$PolicyDecisionContract.AccessControl(AppAttributes, ContPub_k, Operation)$
if $Res == Permit\ and\ State[AppPub_k] != Blocked$ **then**
    $Generate\ New\ Event(AppPub_k, ContPub_k, Operation, Cont\_Challenge)$;
    return $PERMIT$;
else
    $TotalRejections[AppPub_k]++$
    $RejectionRate[AppPub_k] =$
    $TotalRequests[AppPub_k]/TotalRejections[AppPub_k]$;
    if $RejectionRate[AppPub_k] >= Threshold[AppLevel]$ **then**
        $State[AppPub_k] = Blocked$;
        return $BLOCKED$;
    return $DENY$;

5.2 Experimental Results

This section includes the experimental results that showcase the applicability of SDN-API-Sec along with comparisons with existing solutions. The experimental work is carried out on a physical machine with Xubuntu 20.04 OS. We used Hyperledger Fabric[3] for our BC. The details of our experimental setup are given in Table 5. Our consortium BC part includes (3, 5, 10) organizations, each of which consists of 2 peers. In addition to one orderer peer, which is responsible for determining the order of BC transactions [55]. We used Hyperledger Caliper[4] to measure the latency and throughput of SDN-API-Sec. We compared the results of our proposed system with B-DAC [2].

#	Details
CPU	Intel i5-1135G7
RAM	16 GB
Operating system	Xubuntu 20.04
Blockchain	Hyperledger Fabric 1.4
Number of BC organizations	(3, 5, 10) organizations, 2 peers per organization
Block size	100 transaction per block
Batch timeout (Block frequency)	1 second (1 block per second)

Table 5: Experimental setup

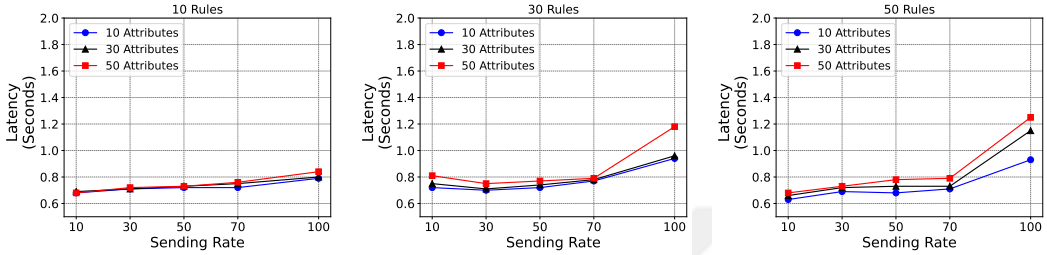
5.2.1 Latency and Throughput for 3 Organizations

Figure 59 shows the average latency for 3 organizations. The first case shows that the latency remains below 1 seconds for the different attributes. The second case shows that the latency increases above 1 second when the request rate is higher 70 with 50 attributes. The third case shows that the latency increases above 1 second when the request rate is higher 70 and the number of attributes is above 10. It is important to note that when the sending rate increases, the difference for higher number of

[3]Hyperledger Fabric [online]. Website https://hyperledger-fabric.readthedocs.io/en/release-1.4/ [accessed 31 Jan 2023]

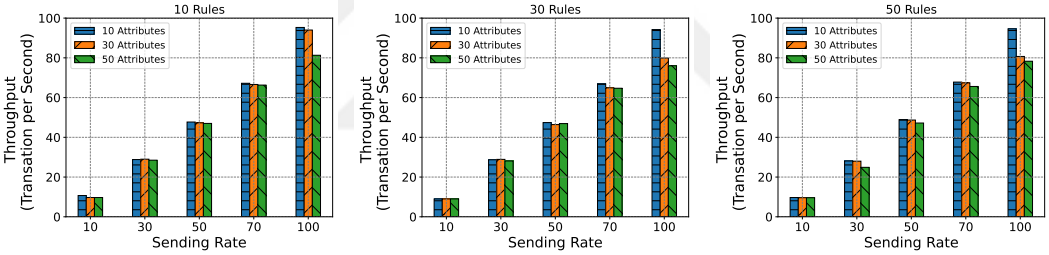
[4]Hyperledger Caliper [online]. Website https://hyperledger.github.io/caliper/ [accessed 31 Jan 2023]

attributes increases as well. Figure 60 shows the throughput for 3 organizations. The results show that the throughput decreases when the number of the attributes is increased. The results show that there is no significant difference for all cases until 100 sending rate.



(a) Case 1: 10 Access Rules(b) Case 2: 30 Access Rules(c) Case 3: 50 Access Rules

Figure 59: Latency for 3 organizations

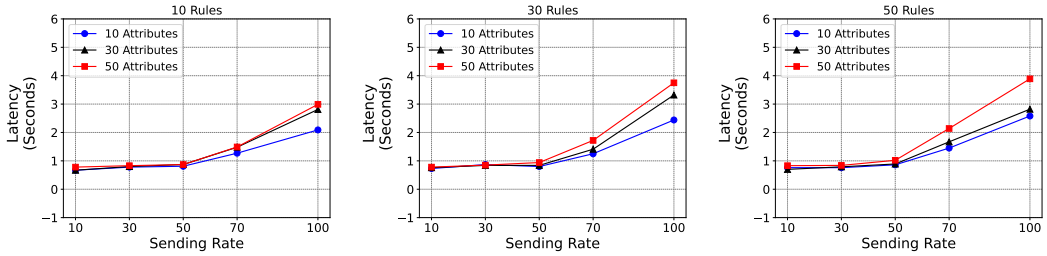


(a) Case 1: 10 Access Rules(b) Case 2: 30 Access Rules(c) Case 3: 50 Access Rules

Figure 60: Throughput for 3 organizations

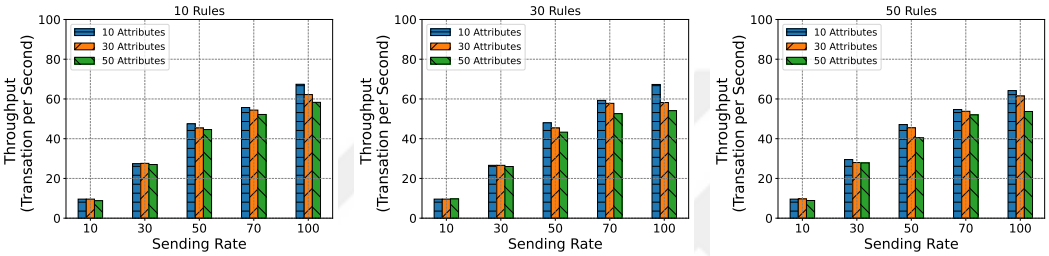
5.2.2 Latency and throughput for 5 Organizations

Figure 61 shows the average latency for 5 organizations. The first and second cases show that the average latency increases above 2 second when the request rate is higher than 70. The third case show that the average latency increases above 2 seconds when the request rate is higher than 50. Figure 62, show the throughput for 5 organizations. The second case shows that the throughput remains below 70 transaction per second for all cases.



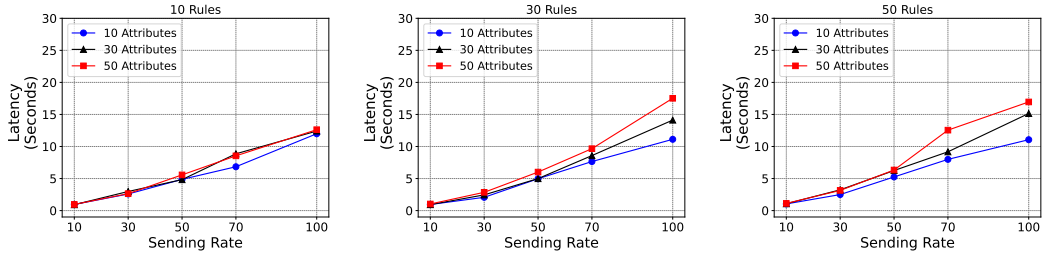
(a) Case 1: 10 Access Rules(b) Case 2: 30 Access Rules(c) Case 3: 50 Access Rules

Figure 61: Latency for 5 organizations



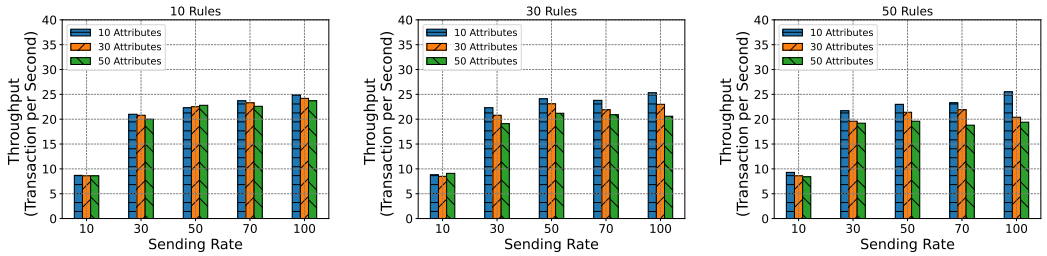
(a) Case 1: 10 Access Rules(b) Case 2: 30 Access Rules(c) Case 3: 50 Access Rules

Figure 62: Throughput for 5 organizations



(a) Case 1: 10 Access Rules(b) Case 2: 30 Access Rules(c) Case 3: 50 Access Rules

Figure 63: Latency for 10 organizations



(a) Case 1: 10 Access Rules(b) Case 2: 30 Access Rules(c) Case 3: 50 Access Rules

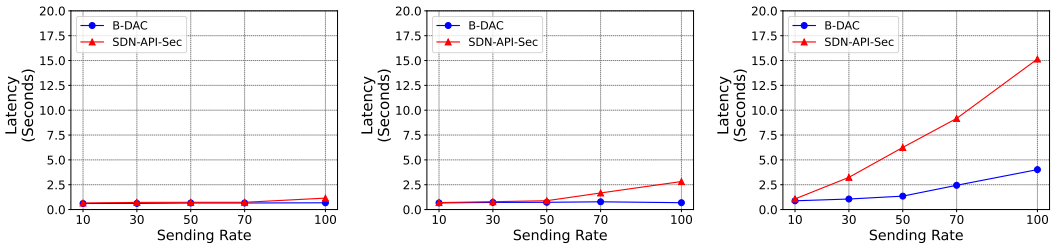
Figure 64: Throughput for 10 organizations

5.2.3 Latency and Throughput for 10 Organizations

Figure 63 shows the average latency for 10 organizations. The results show that the average latency increases above 2 second when the request rate is higher than 10 for all cases. Figure 64 shows the throughput for 10 organizations. The results show that the throughput remains below 30 transaction per second for all cases. This is expected as increasing the the number of BC organizations will also increase the latency and decrease the throughput of the BC system.

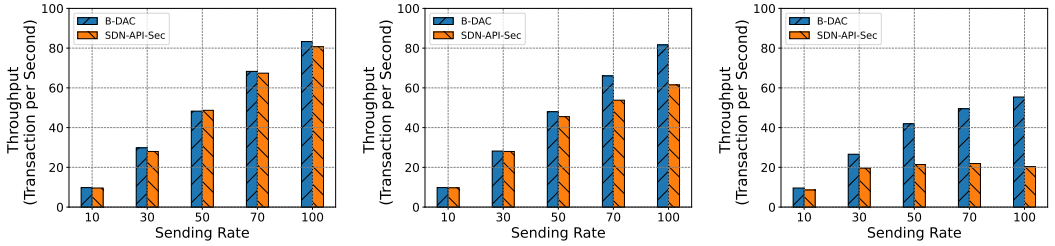
5.2.4 Comparing with B-DAC [2]

We also compare SDN-API-Sec with B-DAC [2] in terms of latency and throughput. We set the number of attributes to 30 and access rules to 50. As shown in Figure 65, the latency for B-DAC [2] remains under 1 seconds for the first and second cases. The third case showed that the latency for B-DAC [2] increases above 1 second when the request rate is higher than 10.



(a) Case 1: 3 Org (b) Case 2: 5 Org (c) Case 3: 10 Org

Figure 65: Comparing the latency of SDN-API-Sec with B-DAC [2]



(a) Case 1: 3 Org (b) Case 2: 5 Org (c) Case 3: 10 Org

Figure 66: Comparing the throughput of SDN-API-Sec with B-DAC [2]

For SDN-API-Sec, the first case showed that the latency increases above 1 second when the request rate is higher than 70. The second case of SDN-API-Sec showed that the latency increases above 1 second when the request rate is higher than 50. The third case showed that the latency increases above 2 seconds when is higher than 10. Figure 66 shows that B-DAC [2] has higher throughput compared with SDN-API-Sec. This is due to the fact that B-DAC [2] relies on a single smart contract, and utilizes role-based access control approach, which limits its flexibility and applicability in systems that require complex authorization rules.

Table 6 shows a comparison between B-DAC [2] and SDN-API-Sec in terms of the features included in each system. SDN-API-Sec provides more fine grained access control compared to B-DAC [2] due to reliance on the attribute-based approach. Moreover, SDN-API-Sec is designed for cross-domain environments where each domain is managed by a separate SDN controller, which is more appropriate for large scale SDNs.

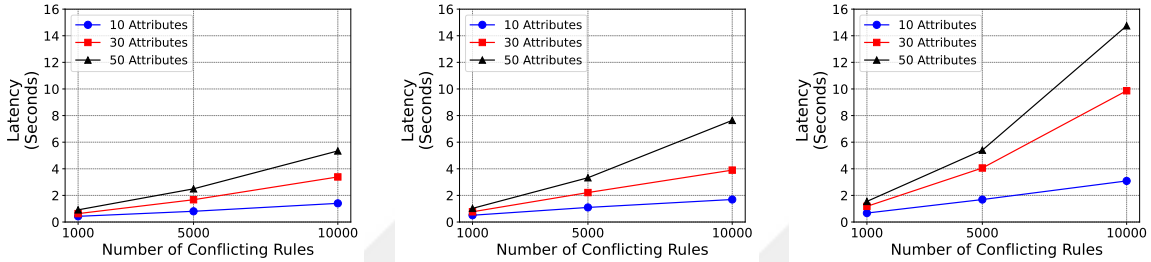
Furthermore, SDN-API-Sec provides a flexible approach since access rules can be updated. Whereas B-DAC [2] relies on a fixed smart contract which checks an authorization token, sender's role, and the requested permission. In SDN-API-Sec, we write fine-grained access rules expressed using as a set of attributes and combining algorithm. This allows the system to be utilized in more complex scenarios. Additionally, our proposed system can detect both conflicting and redundant access rules among different SDN domains to avoid any conflicts introduced by the service administrator.

	B-DAC [2]	SDN-API-Sec
Access control method	Role-based	Attribute-based
Cross-domain	✗	✓
Flexibility	✗	✓
Conflict detection	✗	✓
Redundancy detection	✗	✓

Table 6: Comparing between SDN-API-Sec and B-DAC [2].

5.2.5 Latency for detecting conflicts in SDN-API-Sec

Figure 67, shows the latency for detecting conflicting rules in SDN-API-Sec. The results show that the latency increases in proportion to number of rules and attributes included in each rule.



(a) Case 1: 3 Org (b) Case 2: 5 Org (c) Case 3: 10 Org

Figure 67: Latency of detecting conflicts in SDN-API-Sec

5.3 Summary

In this section, we proposed a blockchain-based cross domain access control approach for SDNs. Our system relies on attribute-based access control mechanism. The system avoids single point of failure and data manipulation by using blockchain technology. SDN-API-Sec provides a highly flexible mechanism since it allows changing the access rules without changing the corresponding smart contract. It also provides a fine-grained access control mechanism by utilizing attribute-based approach. In addition, the smart contracts used in SDN-API-Sec can detect conflict and redundant rules. We conduct a formal security analysis using the AVISPA [6] tool, which ensured the security goals of the proposed system. We compared our work with B-DAC [2], which is less flexible, compared to SDN-API-Sec, since it relies on role-based access control approach. In addition, B-DAC [2] does not consider cross-domain access requests. The results showed that B-DAC [2] has better performance in terms of latency and throughput. However, the benefits provided by SDN-API-Sec, such as a highly flexible, fine-grained cross-domain access control, outweigh the decreased

performance when the system is under high loads. Additionally, SDN-API-Sec does not include any trusted components, which allows the system to work in hostile environments.

CHAPTER VI

CONCLUSIONS

In this thesis, we focused on the integration between SDN and blockchain. We showed how blockchain can be used to achieve a decentralized authentication and authorization for SDNs. Our contributions to BC-enabled SDN are summarized as follows:

- We proposed DPSEC as a Proof-of-Concept (PoC) for blockchain-based data plane authentication in SDN. DPSEC considers both SDN switches and hosts. DPSEC can be applied for both intra- and cross-domain settings. The results showed that DPSEC has higher latency compared with traditional SDN. This is expected since each switch has to wait for the final commitment.

- We improved the performance of DPSEC by separating switch-related and host-related transactions. In CWT-DPA, the switch can send switch-related transactions without waiting for final commitment. Whereas for host-related transactions it has to waits only for a fraction of submitted transactions based on FWFC metric calculated for each switch during each epoch. In addition, we provided protection against quantum adversaries for SDN switches and controllers by utilizing lattice-based signatures and Key Encapsulation Methods (KEMs). The results showed that CWT-DPA significantly reduces the latency of BC-enabled SDN compared with DPSEC.

- For host security, we proposed a blockchain-based authentication framework that provides mutual host-controller, Packet-In/Packet-Out and host-host authentication methods. We also provide secure ARP and IRP to protect layer 2 and layer 3 of the SDN network. The results showed that HostSec can detect

both host-based and switch-based Packet-In attacks and also reduces the load on the SDN controller using our Packet-In manager component. In addition, the results demonstrated a trade-off between improved security and lower latency.

- We proposed a cross-domain access control in SDN using a consortium BC approach. Our proposed system provides a flexible access control method for SDN components by utilizing smart contract based APIs. Attribute-based approach is used to achieve a fine-grained access control in SDNs. The results showed that B-DAC [2] has better performance in terms of latency and throughput compared to our system. This is expected since B-DAC [2] relies on a single smart contract and utilizes role-based access control approach. Therefore, there is a trade-off between performance and flexibility among different BC-based access control systems.

No security solution is fully shielded against all potential attacks and our proposed work isn't either. In this thesis, we focused on enhancing the security of existing SDN paradigm by adding a level of decentralization based on blockchain technology. The system also relies on the security of BC, which can be improved to increase the overall resiliency of the proposed system. Moreover, malicious adversaries can target users in the system through social engineering attacks [94].

Future Work Our work enables a secure collaboration between different SDN domains based on BC-SDN model in which BC-enabled SDN network devices (i.e, SDN switches) and applications (i.e, router, firewall) can interact with verified hosts. As a future direction, this work can be employed to enhance the security of cross-domain applications for large-scale SDN networks. For example, it is possible to create secure collaborative intrusion detection system on top of our proposed work.

It is also possible to explore the potential of blockchain smart contracts to create decentralized network applications, which can utilize our proposed system to ensure that each component is authenticated and authorized. This approach provides more secure and reliable solution compared to centralized network application.

There is also a need to investigate the integration between blockchain and network protocols utilized in SDNs. The security of existing network protocols utilized in SDN can be improved using blockchain technology. This approach provides a level of decentralization to SDN-tailored network protocols.

Furthermore, enhancing the privacy of BC-SDN models is an important topic that needs to be investigated. Therefore, there is a need for BC-enabled privacy preserving secure authentication and authorization mechanisms for SDNs.

REFERENCES

[1] M. Latah and K. Kalkan, "Dpsec: A blockchain-based data plane authentication protocol for sdns," in *Second International Conference on Blockchain Computing and Applications (BCCA)*, pp. 22–29, IEEE, 2020.

[2] P. T. Duy, H. Do Hoang, A. G.-T. Nguyen, V.-H. Pham, *et al.*, "B-dac: A decentralized access control framework on northbound interface for securing sdn using blockchain," *Journal of Information Security and Applications*, vol. 64, p. 103080, 2022.

[3] M. Latah and K. Kalkan, "Cwt-dpa: Component-wise waiting time for bc-enabled data plane authentication," *Computer Networks*, vol. 219, p. 109423, 2022.

[4] M. Latah and K. Kalkan, "When sdn and blockchain shake hands," *Communications of the ACM*, vol. 65, no. 9, pp. 68–78, 2022.

[5] K. L. Huang, S. S. Kanhere, and W. Hu, "On the need for a reputation system in mobile phone based sensing," *Ad Hoc Networks*, vol. 12, pp. 130–149, 2014.

[6] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. H. Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, *et al.*, "The avispa tool for the automated validation of internet security protocols and applications," in *International Conference on Computer Aided Verification*, pp. 281–285, Springer, 2005.

[7] D. M. F. Mattos and O. C. M. B. Duarte, "Authflow: authentication and access control mechanism for software defined networking," *Annals of Telecommunications*, vol. 71, no. 11, pp. 607–615, 2016.

[8] 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.

[9] J. C. C. Chica, J. C. Imbachi, and J. F. B. Vega, "Security in sdn: A comprehensive survey," *Journal of Network and Computer Applications*, vol. 159, p. 102595, 2020.

[10] D. B. Rawat and S. R. Reddy, "Software defined networking architecture, security and energy efficiency: A survey," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 325–346, 2016.

[11] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[12] M. Soliman, B. Nandy, I. Lambadaris, and P. Ashwood-Smith, "Exploring source routed forwarding in sdn-based wans," in *2014 IEEE International Conference on Communications (ICC)*, pp. 3070–3075, IEEE, 2014.

[13] W. Lin, Y. Niu, X. Zhang, L. Wei, and C. Zhang, "Using path label routing in wide area software-defined networks with openflow," in *International Conference on Networking and Network Applications (NaNA)*, pp. 149–154, IEEE, 2016.

[14] J. Lopez, R. Oppliger, and G. Pernul, "Authentication and authorization infrastructures (aais): a comparative survey," *Computers & Security*, vol. 23, no. 7, pp. 578–590, 2004.

[15] M. Trnka, T. Cerny, and N. Stickney, "Survey of authentication and authorization for the internet of things," *Security and Communication Networks*, vol. 2018, 2018.

[16] T. Sans, F. Cuppens, and N. Cuppens-Boulahia, "A flexible and distributed architecture to enforce dynamic access control," in *Security and Privacy in Dynamic Environments: Proceedings of the IFIP TC-11 21st International Information Security Conference (SEC), 22–24 May 2006, Karlstad, Sweden 21*, pp. 183–195, Springer, 2006.

[17] A. Rashid, A. Masood, H. Abbas, and Y. Zhang, "Blockchain-based public key infrastructure: a transparent digital certification mechanism for secure communication," *IEEE Network*, vol. 35, no. 5, pp. 220–225, 2021.

[18] X. Jia, N. Hu, S. Su, S. Yin, Y. Zhao, X. Cheng, and C. Zhang, "IRBA: An identity-based cross-domain authentication scheme for the internet of things," *Electronics*, vol. 9, no. 4, p. 634, 2020.

[19] A. S. Rajasekaran, M. Azees, and F. Al-Turjman, "A comprehensive survey on blockchain technology," *Sustainable Energy Technologies and Assessments*, vol. 52, p. 102039, 2022.

[20] Y. Chen and C. Bellavitis, "Blockchain disruption and decentralized finance: The rise of decentralized business models," *Journal of Business Venturing Insights*, vol. 13, p. e00151, 2020.

[21] W. Li, W. Meng, Z. Liu, and M.-H. Au, "Towards blockchain-based software-defined networking: Security challenges and solutions," *IEICE Transactions on Information and Systems*, vol. 103, no. 2, pp. 196–203, 2020.

[22] P. K. Sharma, S. Singh, Y.-S. Jeong, and J. H. Park, "Distblocknet: A distributed blockchains-based secure sdn architecture for iot networks," *IEEE Communications Magazine*, vol. 55, no. 9, pp. 78–85, 2017.

[23] Z. A. El Houda, A. S. Hafid, and L. Khoukhi, "Cochain-sc: An intra-and inter-domain ddos mitigation scheme based on blockchain using sdn and smart contract," *IEEE Access*, vol. 7, pp. 98893–98907, 2019.

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

[25] K. Kataoka, S. Gangwar, and P. Podili, "Trust list: Internet-wide and distributed iot traffic management using blockchain and sdn," in *IEEE 4th World Forum on Internet of Things (WF-IoT)*, pp. 296–301, IEEE, 2018.

[26] M. Pourvahab and G. Ekbatanifard, "An efficient forensics architecture in software-defined networking-iot using blockchain technology," *IEEE Access*, vol. 7, pp. 99573–99588, 2019.

[27] Q. Qiao, X. Li, Y. Wang, B. Luo, Y. Ren, and J. Ma, "Credible routing scheme of sdn-based cloud using blockchain," in *International Conference of Pioneering Computer Scientists, Engineers and Educators*, pp. 189–206, Springer, 2019.

[28] C. Qiu, F. R. Yu, H. Yao, C. Jiang, F. Xu, and C. Zhao, "Blockchain-based software-defined industrial internet of things: A dueling deep q-learning approach," *IEEE Internet of Things Journal*, 2018.

[29] S. Rathore, B. W. Kwon, and J. H. Park, "Blockseciotnet: Blockchain-based decentralized security architecture for iot network," *Journal of Network and Computer Applications*, vol. 143, pp. 167–177, 2019.

[30] Z. Shao, X. Zhu, A. M. Chikuvanyanga, and H. Zhu, "Blockchain-based sdn security guaranteeing algorithm and analysis model," in *International Conference on Wireless and Satellite Systems*, pp. 348–362, Springer, 2019.

[31] H. Yang, Y. Liang, Q. Yao, S. Guo, A. Yu, and J. Zhang, "Blockchain-based secure distributed control for software defined optical networking," *China Communications*, vol. 16, no. 6, pp. 42–54, 2019.

[32] M. Steichen, S. Hommes, and R. State, "Chainguard—a firewall for blockchain applications using sdn with openflow," in *Principles, Systems and Applications of IP Telecommunications (IPTComm)*, pp. 1–8, IEEE, 2017.

[33] Z. A. El Houda, L. Khoukhi, and A. Hafid, "Chainsecure-a scalable and proactive solution for protecting blockchain applications using sdn," in *IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, 2018.

[34] A. Yazdinejad, R. M. Parizi, A. Dehghantanha, and K.-K. R. Choo, "P4-to-blockchain: A secure blockchain-enabled packet parser for software defined networking," *Computers & Security*, p. 101629, 2019.

[35] A. Yazdinejad, R. M. Parizi, A. Dehghantanha, and K. R. Choo, "Blockchain-enabled authentication handover with efficient privacy protection in sdn-based 5g networks," *IEEE Transactions on Network Science and Engineering*, pp. 1–1, 2019.

[36] N. Zhao, H. Wu, and X. Zhao, "Consortium blockchain-based secure software defined vehicular network," *Mobile Networks and Applications*, pp. 1–14, 2019.

[37] A. Yazdinejad, R. M. Parizi, A. Dehghantanha, Q. Zhang, and K.-K. R. Choo, "An energy-efficient sdn controller architecture for iot networks with blockchain-based security," *IEEE Transactions on Services Computing*, 2020.

[38] 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.

[39] T. Alharbi, "Deployment of blockchain technology in software defined networks: A survey," *IEEE Access*, vol. 8, pp. 9146–9156, 2020.

[40] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.

[41] "OpenFlow Switch Specification Version 1.5.1." https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf, Online Accessed 2023-05-30.

[42] V. Pashkov, A. Shalimov, and R. Smeliansky, "Controller failover for sdn enterprise networks," in *International Science and Technology Conference (Modern Networking Technologies)(MoNeTeC)*, pp. 1–6, IEEE, 2014.

[43] K. Kalkan, G. Gur, and F. Alagoz, "Defense mechanisms against ddos attacks in sdn environment," *IEEE Communications Magazine*, vol. 55, no. 9, pp. 175–179, 2017.

[44] 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*, pp. 171–172, 2013.

[45] F. Hauser, M. Schmidt, and M. Menth, "Establishing a session database for sdn using 802.1 x and multiple authentication resources," in *IEEE International Conference on Communications (ICC)*, pp. 1–7, IEEE, 2017.

[46] B. Yigit, G. Gur, B. Tellenbach, and F. Alagoz, "Secured communication channels in software-defined networks," *IEEE Communications Magazine*, vol. 57, no. 10, pp. 63–69, 2019.

[47] M. Liyanage, M. Ylianttila, and A. Gurtov, "Securing the control channel of software-defined mobile networks," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, pp. 1–6, IEEE, 2014.

[48] A. Ranjbar, M. Komu, P. Salmela, and T. Aura, "An sdn-based approach to enhance the end-to-end security: Ssl/tls case study," in *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*, pp. 281–288, IEEE, 2016.

[49] J. Lam, S.-G. Lee, H.-J. Lee, and Y. E. Oktian, "Securing sdn southbound and data plane communication with ibc," *Mobile Information Systems*, vol. 2016, 2016.

[50] N. Gray, T. Zinner, and P. Tran-Gia, "Enhancing sdn security by device fingerprinting," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 879–880, IEEE, 2017.

[51] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," tech. rep., Manubot, 2019.

[52] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[53] A. Bosu, A. Iqbal, R. Shahriyar, and P. Chakraborty, "Understanding the motivations, challenges and needs of blockchain software developers: A survey," *Empirical Software Engineering*, vol. 24, no. 4, pp. 2636–2673, 2019.

[54] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," *International Journal of Web and Grid Services*, vol. 14, no. 4, pp. 352–375, 2018.

[55] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, pp. 1–15, 2018.

[56] H.-N. Dai, Z. Zheng, and Y. Zhang, "Blockchain for internet of things: A survey," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8076–8094, 2019.

[57] M. Castro, B. Liskov, *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, pp. 173–186, 1999.

[58] S. Algarni, F. Eassa, K. Almarhabi, A. Algarni, and A. Albeshri, "Bcnbi: A blockchain-based security framework for northbound interface in software-defined networking," *Electronics*, vol. 11, no. 7, p. 996, 2022.

[59] I. Al Ridhawi, M. Aloqaily, A. Boukerche, and Y. Jaraweh, "A blockchain-based decentralized composition solution for iot services," in *ICC IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2020.

[60] E. Yuan and J. Tong, "Attributed based access control (abac) for web services," in *IEEE International Conference on Web Services (ICWS'05)*, IEEE, 2005.

[61] "LINC-Switch." https://github.com/FlowForwarding/LINC-Switch. [Online accessed May 17, 2020].

[62] "RYU." https://ryu.readthedocs.io/en/latest/. [Online Accessed 03-May-2022].

[63] "Hyperledger Fabric." https://www.hyperledger.org/use/fabric/. [Online Accessed 03-May-2022].

[64] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ECDSA)," *International Journal of Information Security*, vol. 1, pp. 36–63, 2001.

[65] E. Rescorla, "The transport layer security (TLS) protocol version 1.3," tech. rep., 2018.

[66] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, Z. Zhang, *et al.*, "Falcon: Fast-Fourier lattice-based compact signatures over NTRU," tech. rep., National Institute of Standards and Technology, 2017.

[67] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-dilithium: A lattice-based digital signature scheme," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 238–268, 2018.

[68] J. Hoffstein, J. Pipher, and J. H. Silverman, "NTRU: A ring-based public key cryptosystem," in *Algorithmic Number Theory: Third International Symposiun*, pp. 267–288, Springer, 1998.

[69] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-kyber: a cca-secure module-lattice-based kem," in *IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 353–367, 2018.

[70] "NIST." https://nvd.nist.gov/vuln/detail/CVE-2018-1000155. [Online Accessed May 25, 2020].

[71] P. M. Mohan, T. Truong-Huu, and M. Gurusamy, "Towards resilient in-band control path routing with malicious switch detection in sdn," in *10th International Conference on Communication Systems & Networks (COMSNETS)*, pp. 9–16, IEEE, 2018.

[72] Q. Zhou, J. Yu, and D. Li, "A dynamic and lightweight framework to secure source addresses in the sdn-based networks," *Computer Networks*, vol. 193, p. 108075, 2021.

[73] A. Ahmad, M. Saad, J. Kim, D. Nyang, and D. Mohaisen, "Performance evaluation of consensus protocols in blockchain-based audit systems," in *2021 International Conference on Information Networking (ICOIN)*, pp. 654–656, IEEE, 2021.

[74] G. D. Putra, V. Dedeoglu, S. S. Kanhere, R. Jurdak, and A. Ignjatovic, "Trust-based blockchain authorization for iot," *IEEE Transactions on Network and Service Management*, 2021.

[75] "Open Quantum Safe Library." https://openquantumsafe.org/liboqs/. [Online Accessed 03-May-2022].

[76] N. Lu, Y. Zhang, W. Shi, S. Kumari, and K.-K. R. Choo, "A secure and scalable data integrity auditing scheme based on hyperledger fabric," *Computers & Security*, vol. 92, p. 101741, 2020.

[77] NIST, "Submission requirements and evaluation criteria for the post-quantum cryptography standardization process." http://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf. [Online accessed 03-May-2022].

[78] "Iperf: The TCP/UDP bandwidth measurement tool." http://dast.nlanr.net/Projects/Iperf/. [Online Accessed 03-May-2022].

[79] D. Kreutz, F. M. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proceedings of the second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, pp. 55–60, 2013.

[80] "Open vSwitch." https://www.openvswitch.org/. [Online accessed 03-May-2022].

[81] 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.

[82] G. Alagic, G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, *et al.*, *Status report on the first round of the NIST post-quantum cryptography standardization process*. US Department of Commerce, National Institute of Standards and Technology, 2019.

[83] F. X. Wibowo, M. A. Gregory, K. Ahmed, and K. M. Gomez, "Multi-domain software defined networking: research status and challenges," *Journal of Network and Computer Applications*, vol. 87, pp. 32–45, 2017.

[84] H. Zhang, X. Chen, X. Lan, H. Jin, and Q. Cao, "Btcas: A blockchain-based thoroughly cross-domain authentication scheme," *Journal of Information Security and Applications*, vol. 55, p. 102538, 2020.

[85] 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*, pp. 31–40, 2011.

[86] Y. Li, Y. Yu, C. Lou, N. Guizani, and L. Wang, "Decentralized public key infrastructures atop blockchain," *IEEE Network*, vol. 34, no. 6, pp. 133–139, 2020.

[87] R. A. Shaikh, K. Adi, and L. Logrippo, "A data classification method for inconsistency and incompleteness detection in access control policy sets," *International Journal of Information Security*, vol. 16, pp. 91–113, 2017.

[88] G. Liu, W. Pei, Y. Tian, C. Liu, and S. Li, "A novel conflict detection method for abac security policies," *Journal of Industrial Information Integration*, vol. 22, p. 100200, 2021.

[89] S. Kern, T. Baumer, S. Groll, L. Fuchs, and G. Pernul, "Optimization of access control policies," *Journal of Information Security and Applications*, vol. 70, p. 103301, 2022.

[90] K. Vijayalakshmi and V. Jayalakshmi, "Identifying considerable anomalies and conflicts in abac security policies," in *5th International Conference on Intelligent Computing and Control Systems (ICICCS)*, pp. 1273–1280, IEEE, 2021.

[91] "Extensible Access Control Markup Language (XACML) Version 3.0." http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html, Online Accessed 2023-05-30.

[92] N. Li, Q. Wang, P. Rao, D. Lin, E. Bertino, and J. Lobo, "A formal language for specifying policy combining algorithms in access control," *CERIAS.(Cited on pages 88 and 112.), Tech. Rep.*, pp. 2008–9, 2008.

[93] C. D. P. K. Ramli, H. R. Nielson, and F. Nielson, "The logic of xacml," *Science of Computer Programming*, vol. 83, pp. 80–105, 2014.

[94] N. Ivanov, J. Lou, T. Chen, J. Li, and Q. Yan, "Targeting the weakest link: Social engineering attacks in ethereum smart contracts," in *Proceedings of the ACM Asia Conference on Computer and Communications Security*, pp. 787–801, 2021.

[95] M. T. De Oliveira, L. H. A. Reis, Y. Verginadis, D. M. F. Mattos, and S. D. Olabarriaga, "Smartaccess: Attribute-based access control system for medical records based on smart contracts," *IEEE Access*, vol. 10, pp. 117836–117854, 2022.

[96] D. D. F. Maesa, P. Mori, and L. Ricci, "A blockchain based approach for the definition of auditable access control systems," *Computers & Security*, vol. 84, pp. 93–119, 2019.

APPENDIX A

COPYRIGHT PERMISSION

For the article [4], published in CACM: "Authors can include partial or complete papers of their own (and no fee is expected) in a dissertation as long as citations and DOI pointers to the Versions of Record in the ACM Digital Library are included". https://authors.acm.org/author-resources/author-rights

For the article [1], in reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of Özyeğin University's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to

http://www.ieee.org/publications$_s$*tandards/publications/rights/rights*$_l$*ink.html*

to learn how to obtain a License from RightsLink.

© 2020 IEEE. Reprinted, with permission, from M. Latah and K. Kalkan, "DPSEC: A blockchain-based data plane authentication protocol for sdns," in 2020 Second International Conference on Blockchain Computing and Applications (BCCA), pp. 22–29, IEEE, 2020.

For the article [3], "Author rights in Elsevier's proprietary journals: Include in a thesis or dissertation (provided this is not published commercially)".

https://www.elsevier.com/about/policies/copyright

VITA

Majd Latah received his BSc degree in Computer Engineering from İttihad University in 2011 and his MSc degree in Computer Engineering from Ege University in 2018. He is currently pursuing his Ph.D. at the Department of Computer Science at Özyeğin University under the supervision of Asst. Prof. Kübra Kalkan. His research interests include network security, SDN, and blockchain.