NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

SOFTWARE DEFINED NETWORKS: DIALECTING SECURITY

by

Nektaria Patrozou

March 2022

Thesis Advisor: Britta Hale
Co-Advisor: Geoffrey G. Xie

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2022	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE SOFTWARE DEFINED NETWORKS: DIALECTING SECURITY	**5. FUNDING NUMBERS**
6. AUTHOR(S) Nektaria Patrozou	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

OpenFlow is the standard used in Software Defined Networks. It handles the communication between the network devices. However, there are some weaknesses linked to OpenFlow. With the use of TLS as a security solution, it inherits the vulnerabilities of TLS in downgrade attacks. Furthermore, TLS is optional. To enhance the security in OpenFlow, previous research work provided a solution that comes with the notion of protocol dialects. Protocol dialects are variations of an existing implementation of an open-source protocol, such as OpenFlow. They are implemented either by adding proxies or directly modifying the protocol to the core. The protocol dialect we analyze in this research follows the first approach by manipulating the protocol in such a way that the actual devices continue to function as before, but additional security measures are put in place with the use of proxies. Desired additional functionality, additional security measures, and changes in fields of the actual protocol are performed within the proxies. The devices "think" that they are communicating with each other exactly as before, but in reality a proxy is standing in front of each device, and the actual communication takes place with the proxies' mediation. In this research, we aim to show the enhanced security of the dialected OpenFlow protocol. We follow the computational analysis model to conduct a security proof for the dialect, and we also analyze some difficulties in conducting such a proof.

14. SUBJECT TERMS SDN, dialecting security, software defined networks, protocols	**15. NUMBER OF PAGES** 79
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

SOFTWARE DEFINED NETWORKS: DIALECTING SECURITY

Nektaria Patrozou
Lohagos, Hellenic Army
BS, Hellenic Army Academy (Evelpidon), 2008

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2022

Approved by: Britta Hale
Advisor

Geoffrey G. Xie
Co-Advisor

Gurminder Singh
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

OpenFlow is the standard used in Software Defined Networks. It handles the communication between the network devices. However, there are some weaknesses linked to OpenFlow. With the use of TLS as a security solution, it inherits the vulnerabilities of TLS in downgrade attacks. Furthermore, TLS is optional. To enhance the security in OpenFlow, previous research work provided a solution that comes with the notion of protocol dialects. Protocol dialects are variations of an existing implementation of an open-source protocol, such as OpenFlow. They are implemented either by adding proxies or directly modifying the protocol to the core. The protocol dialect we analyze in this research follows the first approach by manipulating the protocol in such a way that the actual devices continue to function as before, but additional security measures are put in place with the use of proxies. Desired additional functionality, additional security measures, and changes in fields of the actual protocol are performed within the proxies. The devices "think" that they are communicating with each other exactly as before, but in reality a proxy is standing in front of each device, and the actual communication takes place with the proxies' mediation. In this research, we aim to show the enhanced security of the dialected OpenFlow protocol. We follow the computational analysis model to conduct a security proof for the dialect, and we also analyze some difficulties in conducting such a proof.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

API	Application Programming Interface
AS	Autonomous Systems
DiD	Defense in Depth
DoS	Denial of Service
IT	Information Technology
IXP	Internet Exchange Points
MITM	Man In the Middle
NAC	Network Access Control
NAT	Network Address Translation
NOS	Network Operating System
OF	OpenFlow
ONF	Open Networking Foundation
OTWG	Optical Transport Working Group
SDN	Software Defined Networks
SNMP	Simple Network Management Protocol
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TSA	Time Stamp Authority
TLS	Transport Layer Security

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

I would like to express my deep gratitude to my two advisors, Dr. Britta Hale and Dr. Geoffrey Xie. It has been an honor to have worked with them. Their guidance, mentorship, and patience have helped with the successful completion of this thesis.

A special thank you to Dr. Britta Hale for always finding time despite her hectic schedule to help or guide me through the difficulties that I encountered along the way. Her integrity and professionalism are admirable and motivational.

I would also like to thank my siblings and friends who have supported, encouraged and helped me through every step of the way in this challenging journey of the past two years.

I would like to thank the people that helped me build the foundation of this educational journey, my parents. Their efforts and sacrifices in order to provide the best education for me and my siblings have been the rudimentary key to my success in this program. All in all, their continuous love and support truly helped me conquer all the hurdles in order to achieve my goals. It was them that lifted me up during moments of discouragement and struggles throughout this program.

And last but not least, in remembrance of my lovely aunt Vaso, for her unconditional love, support, and encouragement. You have taught me the meaning of unconditional love. You will always be a part of my journey. And thank you for the love and care you have given me.

THIS PAGE INTENTIONALLY LEFT BLANK

Dedication

In the memory of Panagiota Stratou-Katsimpra, Vera Katsimpra and Vaso Talatzi. The three people who inspired most in life with their patience, resilience, and hard work.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Prelude

In this chapter we introduce the motivation behind our research and describe the main components of this work.

Software Defined Networking (SDN) technology aims to make a network programmable and manageable from a central decision point called controller. Thus, the idea behind SDN is centralized control and management of network devices using software. The controller is logically centralized because its functionality may be carried out by a collection of controller instances that are distributed. In SDN, network devices that process and forward data packets are referred to as switches. An SDN switch relies on the controller to provide instruction on how to process incoming traffic on a per application flow basis. This emerging technology has the potential to change the networks industry. However, for this to be achieved, research must be performed in the management and evaluation of all the security aspects to implement reliable networks.

There are great financial benefits for organizations from using this technology. When routers and switches are supported by SDN, the organizations do not have to rely on an external company with special engineers to undertake the implementation of the network infrastructures.

1.1 Motivation

The OpenFlow Protocol (OF) is a standard for facilitating communication between an SDN controller and each switch under its control [1]. Typically, a controller supports many switches at the same time.

As the only option in Software Defined Networks, the OpenFlow Protocol has some weaknesses. Specifically, OpenFlow uses Transport Layer Security (TLS) as a security solution to authenticate the participating devices and encrypt the control messages. However, according to recent research [2], TLS is vulnerable to downgrade attacks. Additionally, TLS is optional in the current OF standard.

A methodology to strengthen vulnerable protocols comes with the notion of protocol dialects. A protocol dialect is a modification of the existing implementation of an open-source protocol. Dialecting a protocol can be performed with the use of proxies or with the modification of the actual protocol implementation at the binary level. With protocol dialects, we are able to add security measures or remove unused features while maintaining the core functionality of the protocol. In this work, we analyze a protocol dialect of OpenFlow that was developed by Michael Sjoholmsierchio in a recent NPS thesis [2].

Proxies are put in place in front of each device and the additional security is provided only between the proxies. The actual devices, "switch" and "controller", think that they are communicating with each other exactly as before, but the communication is taking place from device-proxy to proxy-device on the other side of the communication channel. Proxies communicate in accordance to the modified version of the protocol and they translate any modified field back to its original, before providing it to the controller or switch. A visual representation of the topology with proxies is shown in Figure 1.2. The functionality taking place in each proxy is shown in Figure 1.1.

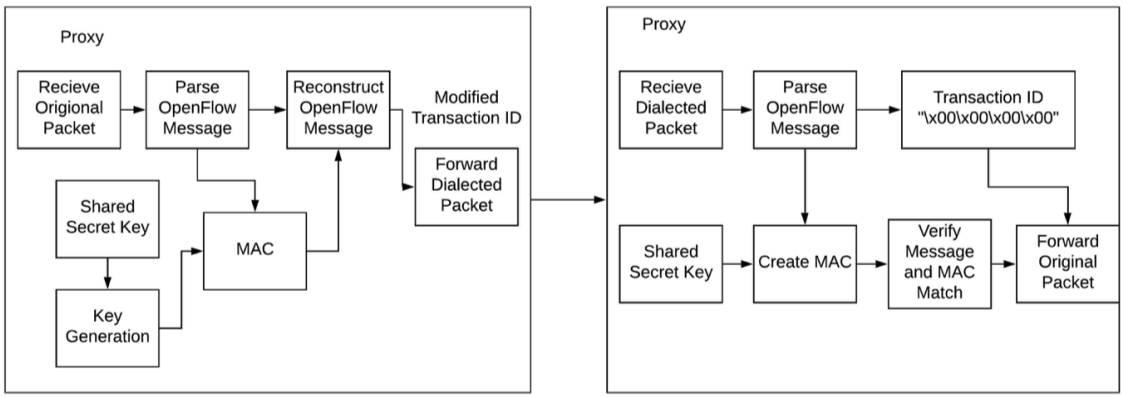

Figure 1.1. Dialect Proxy Software. Source: [3].

1.2 Problem Statement

There are few prior analyses of protocol dialects regarding their security properties, while some of the dialects explicitly aim to strengthen vulnerable protocols. In our work, we analyze a protocol dialect of OpenFlow that was implemented in [3] with the use of proxies.

That work aimed on a more secure OpenFlow (OF) protocol in the Software Defined Networks (SDN) environment, while maintaining an acceptable performance.

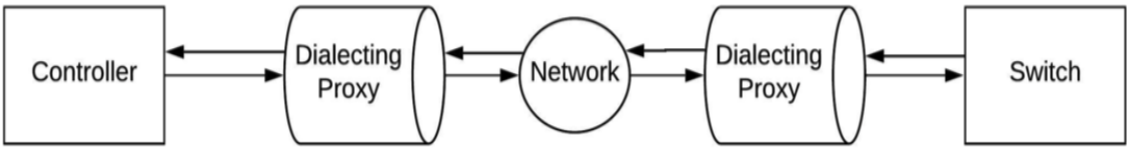

Figure 1.2. Dialecting Proxies Implementation Method. Source: [3].

1.3 Research Questions

This research focuses on mathematical analysis and provable security of Sjoholmsierchio's OpenFlow dialects, which was designed and tested in previous research. We provide a formal proof of the resulted dialected protocol. We try to answer questions such as what key establishment security considerations are in SDN dialecting and how we can mathematically analyze these options. Our goal is to enhance security in the existing protocols used in SDN while making as minimum changes as possible trying to achieve Defense in Depth (DiD).

1.4 Thesis Organization

In Chapter 2 we provide background information about SDN, their limitations and features. Furthermore, we introduce OpenFlow, the protocol used in SDN. Then, we talk about secure communication and how it is provided, while we analyze security goals and introduce security primitives. Afterwards, we review related work that led to the conduction of this research. Chapter 3 contains information about SDN security and also the core preliminaries of our analysis. Chapter 4 contains the protocol description and explanation of a deconfliction algorithm used in our protocol, since it's a protocol that utilizes timestamps. Chapter 5 contains the security model and the security proof. Finally, Chapter 6 contains the conclusion of our analysis as well as ideas for completion and possible future work.

In this chapter we showed what led to the construction of this research, and next we will analyze concepts in more depth.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Background

In the previous chapter we introduced the motivation and main components of our research. In this chapter we explain basic concepts and secure communication of SDN technology. We also refer to previous work that led to this research.

2.1 Software Defined Networking

In traditional networks, there was not an option for a centralized management of the entire network. There was a lack of separation between the control and data planes, which are integrated in the same network devices [4]. Though critical in the early days of networks, it resulted to the main reason why the traditional network architectures are rigid, multifaceted, and hard to manage.

According to Open Networking Foundation (ONF) [1], Software Defined Networking technology is an arising architecture, in which there is a distinction between the routing and the forwarding processes. Furthermore, it can be manipulated by programming. This relocation of control, which used to be closely linked with devices like routers or switches, allows the abstract virtual view of the whole network as a logical entity.

Kreutz et al. [4] distinguish four main components in the design of SDN:

1. The control and data planes are decoupled. Control functionality is removed from network devices that become simple (packet) forwarding elements.
2. Forwarding decisions are flow-based, instead of destination-based. A flow is broadly defined by a set of packet field values acting as a match (filter) criterion and a set of actions (instructions). In the SDN/OpenFlow context, a flow is a sequence of packets between a source and a destination. All packets of a flow receive identical service policies at the forwarding devices. The flow abstraction allows unifying the behavior of different types of network devices, including routers, switches, firewalls, and mid-

dleboxes[1]. Flow programming enables unprecedented flexibility, limited only to the capabilities of the implemented flow tables.

3. Control logic is moved to an external entity, the so called SDN controller or Network Operating System (NOS). The NOS is a software platform that runs on commodity server technology and provides the essential resources and abstractions to facilitate the programming of forwarding devices based on a logically centralized, abstract network view. Its purpose is therefore similar to that of a traditional operating system.
4. The network is programmable through software applications running on top of the NOS that interacts with the underlying data plane devices. This is a fundamental characteristic of SDN, considered as its main value proposition. [4]

2.1.1 Limitations Addressed by SDN

The coverage of today's market demands is almost impossible with just the traditional network architectures [1]. In an effort to cope with low budgets, Information Technology (IT) departments are using management tools on the device's level and non-automatic procedures. However, traditional network architectures stumble on coping with today's user's or business's requirements. The ONF [1] lists the limitations that today's network designers meet. Below we provide summaries of these listings:

- **Complexity that leads to stasis:** Complexity is one of the first and foremost limitations of networks today, with one protocol corresponding to one problem and without the notion of abstraction. It is this complexity that leads to static networks, and those static networks are in contrast with today's dynamic server environments. Furthermore, the number of users has increased, with these users requiring connections from various physical locations.
- **Inconsistent policies:** The effort of putting isolated protocols together trying to achieve a network-wide policy leads to another limitation. It can lead to the need for configuration of a tremendous amount of devices in the network. Additionally, any reconfiguration needed can take a lot of time, hours or even days in some cases. That is

[1]Middle-box: Network device that converts, inspects, and filters or handles in another way the traffic, for reasons other than packet forwarding.

an extremely rough challenge, the IT departments have to cope with, making it almost impossible not to have omissions, leading to security breaches and vulnerabilities in the network.

- **Inability to scale:** The network grows along with the demands on the data center. Thus, with the addition of new devices, configuration and management as is, cannot hold up to this growing complexity.
- **Vendor dependence:** Unfortunately, there is still no standardization, when it comes to network equipment. Enterprises are trying to develop and grow by searching and applying new capabilities according to user or business demands, but they have the limitation of depending on the vendor they purchased products from. Addition of equipment from a different vendor may not be able to work with existing equipment. This is where the Software-Defined Networking architecture comes in contrast, by developing new standards.

Furthermore with the rapid development in technology, computer networks grew bigger, leading to more complex and difficult management, maintenance and control [1]. There was clearly a need to reexamine the traditional network architectures. The staticity of existing networks does not coincide with the dynamic nature of the computing environments in today's enterprises.

Data centers saw changes in traffic patterns, moving from the traditional way of communication between a client and a server, to machine-to-machine and with users connecting to the network from all kinds of devices, from different locations and time [1]. Additionally, enterprises implemented utilities including private cloud or public cloud resulting to additional traffic across the network.

IT's are struggling to incorporate the users' needs while trying to maintain compliance mandate measures protecting data and intellectual property [1]. Another struggle for IT's comes with cloud services, the providing of which must not stop security maintenance and auditing requirements, along with changes in the organization system, consolidations etc.

Lastly, the struggle of bandwidth [1]. With today's big datasets there is a requirement of parallel processing of a big number of servers, all of which directly communicate with each other. This leads to a bigger demand for network capacity. To scale the network to such a before unthinkable size, while maintaining full connectivity is an extremely hard task.

On the other hand, SDN provides ease of adaptation to dynamic inner-organization changes, reduce of control, maintenance and management complexity, and ease of compliance with the high-bandwidth demands [1]. SDN centralizes and simplifies the control and management of networks. It allows networks to keep up with the rapid development in technology.

Some of the benefits that can be achieved through SDN technology, according to ONF [1], are summarized below:

Centralized control of different environments: SDN software provides the control of network devices from different vendors. We can configure, maintain, and update devices wherever in the network, from our office.

Reducing complexity: Automation of managing tasks that before acquired manual treatment. That reduces operational overhead, delays or network instability caused by any kind of human error, and increases the enterprise's agility.

Higher rate of innovation: By making the network programmable, SDN provides escalation to innovation, since it makes it easier to meet user needs and requirements at the exact moment they show up.

Increase of reliability and security: In SDN architecture network devices do not need to be configured individually every time that there is a new add-on or removal or just relocation of a device. That reduces network failures that root from configuration or policy inconsistencies. Additionally it provides a visual representation and thus easier control of the network, with less operational expenses, more configuration capabilities, less errors, and ease of policy enforcement.

Better user experience: SDN architecture dynamically adapts to user needs. The centralized control and information about the state of devices provided to higher-level applications, easily meets any user's needs. An example comes with network resolution. In traditional networks, resolution is a setting which users must select not knowing with certainty if the network is able to support it. That results in an unpleasant user experience, causing delays or interruptions . In SDN,though resolution is automatically adjusted.

2.1.2 SDN Controllers

We can treat SDN controllers as network operating systems [5]. They form a software platform which constitutes the base of all the network control applications. Usually, a set of modules form an SDN controller, and provide different services like routing, security, policy management, and more, in the direction of meeting the enterprises objectives.

We find a number of SDN controllers that were developed with different objectives in mind. According to [5], the most widespread ones are summarized below:

- NOX Controller: This controller was developed by Nicira Networks early on along with the OpenFlow protocol. As one of the firsts, it is considered a safe and stable option, it is widely used and is preferred in educational environments. There are two versions of this controller with the first being the NOX-classic, implemented in C++ and Python supporting the network control application by using both languages. Using two languages though led to inconsistencies concerning features and interfaces. It might be due to these issues that this first version is no longer supported, and instead we moved to the second version which is called simply NOX or "new NOX". This version was implemented in C++, supporting the network control application also developed in C++. Even if considered harder, especially for users that are not familiar with C++, "new NOX" supports both 1.0 and 1.3 versions of OpenFlow and has better performance and programmability.
- POX Controller: For users that are not familiar with C++, POX is the Python version of the NOX controller. With Python's simplicity and flexibility, POX has been the top choice for SDN projects, in things like debugging SDN applications or implementing network virtualization. The POX controller is also supported by the official NOX community. On the downside, POX supports only 1.0 version of OpenFlow protocol but still provides better performance compared with the NOX-Classic. However, Python is not a compiling language, therefore it cannot be used for low-level functionality down to the core of the OpenFlow Protocol, in contradiction with the NOX.
- Ryu Framework: Ryu simplifies network management and applications control. It is based on Python and supports a variety of protocols, such as NETCONF[2], OF-config[3], and OpenFlow, to include version 1.0 and versions 1.2–1.4. Ryu is implemented using Python.
- Floodlight Controller: Except from C++ and Python, there are Java-based versions of OpenFlow Controller, one of them being the Floodlight Controller. This is an open-

[2]"NETCONF is a protocol defined by the IETF to install, manipulate, and delete the configuration of network devices. NETCONF operations are realized on top of a Remote Procedure Call (RPC) layer using an XML encoding and provide a basic set of operations to edit and query configuration on a network device" [6].

[3]"The OF-CONFIG protocol enables configuration of essential artifacts of an OpenFlow logical Switch so that an OpenFlow controller can communicate and control the OpenFlow Logical switch via the OpenFlow protocol" [7]

source option with a great industry support, which comes from Big Switch Networks. Maybe the more complex from the options above, due to a large set of features, it enables easy integration with external business applications.

- OpenDaylight Controller: Another Java-based SDN controller, OpenDaylight was created as a Linux Foundation collaborative project and it focuses on innovation in the SDN environment. Besides the open source community, OpenDaylight project is supported by companies like Cisco, IBM, Brocade and VMware. It supports both 1.0 and 1.3 OpenFlow versions, as well as other protocols like OVSDB[4] We can remotely configure most network devices using legacy protocols, such as Simple Network Management Protocol (SNMP) and NETCONF. On the drawbacks, OpenDaylight is complex because of its architecture and a big number of services provided, making it having a steep learning curve.

	NOX	**POX**	**Ryu**	**FloodLight**	**ODL**
Language	C++	Python	Python	Java	Java
Performance	High	Low	Low	High	High
Distributed	No	No	Yes	No	Yes
OpenFlow	1.0	1.0	1.0, 1.2-1.4	1.0, 1.3	1.0, 1.3
Multi-tenant Cloud	No	No	Yes	Yes	Yes
Learning Difficulty	Moderate	Easy	Moderate	Steep	Steep

Figure 2.1. Main Characteristics of SDN Controllers. Adapted from [5].

SDN Applications

There is wide variety of network environments where we encounter SDN applications. The enabling of customized control with programmable networks, the simplification of development and deployment of new network services and protocols make it a preferable choice that is already used [9]. Below view different e nvironments i n w hich Software Defined Networking is implemented, summarized from [9].

[4]OVSDB: Open vSwitch Database, is a management protocol in a software-defined networking environment. OVSDB was created by the Nicira team that was later acquired by VMware [8].

- Enterprise Networks. In enterprises we usually encounter large networks, while security and performance requirements are quite demanding. Of course requirements differ from one enterprise to another according to the goals, characteristics, number of employees etc. As an example, networks in Universities, can be challenging in terms of security, with many devices connecting with these temporary connections not being controlled by the University, or the resource allocation. Additionally, there is the need in these environments, to provide support for research related projects, experiments and protocols. With SDN, the difficulty and complexity is limited by the ability "to programmatically enforce and adjust policies" [9], or network performance, or make it simple to monitor network activity.

  Some examples of SDN implementation include "Network Address Translation (NAT), firewalls, load balancers, and Network Access Control (NAC)" [9]. In more complex implementations, SDN can be used for centralized management and control. Another issue in large networks aligns with consistent network updates. Instability in networks roots mainly from configuration changes and may cause security flaws, outages and performance disruptions. "With a set of high-level abstractions, network administrators are able to update the entire network, guaranteeing that every packet traversing the network is processed by exactly one consistent global network configuration. To support these abstractions, several OpenFlow-based update mechanisms were developed" [9]. Lastly, OpenFlow itself, was initially designed taking issues found in enterprise networks into consideration.
- Data Centers. Data centers have evolved and continue to evolve rapidly continuously attempting to meet higher and higher standards. The need for careful traffic management and enforcement of policies escalates with the size of the infrastructures. Especially in businesses where even just a small delay can lead to financial damage. Despite the challenges of large scale networks, and the complexity which large scale brings, data centers have always been adapting in higher demands and requirements. Therefore, they run well below capacity while being ready to support higher workloads.

  An important issue in large scale data centers, is energy consumption with its large scale cost. There has been research, that has been focused on improving servers by better managing hardware or/and software, but still the amount of energy consumed is great. Heller et al. [10] research, proposed a power manager (ElasticTree) throughout

the network, which with the help of SDN, tries to find a sub-network in the network, that uses minimum power for real-time traffic conditions. Additionally, it turns off switces that are not used at the time. Their results showed energy savings up to 62%. However, there are high performance networks in which SDN solutions are not appropriate. There is a need to balance the simplified traffic management and visibility coming from SDN technology, with scalability and performance overhead.

OpenFlow might be considered excessive, since it combines central control and complete visibility, even when only "significant" flows need to be managed. That may add delays, especially when switches are overwhelmed with flow table entries. There are ways to manage that issue but still it might affect the controller on effectively managing traffic and gathering statistics. There have been suggestions for design modifications that try to keep flows in the data plane while maintaining visibility, which could, at some level, balance these issues out.

- Optical Networks. Software-defined and OpenFlow networks, with the management of data traffic as flows, are able to support multiple network technologies. Thus, they are able to provide centralized control for optical networks.

  According to the ONF [1] and the Optical Transport Working Group (OTWG) that was created in 2013, SDN and the OpenFlow standard provides benefits to optical transport networks, such as management flexibility, improvement of network control, third-party integration, control systems, and new services.
- Home and Small Business. Home or small businesses networks are, maybe, the most widely used networks and they have become highly complex as well. Low cost network devices allow flexibility and spread of the network but at the same time require a more careful network management and stricter security objectives. Networks that are not secured properly constitute attack targets for malware, while any outages rooting from network configuration issues can cause financial or other types of losses in small businesses. It is not practical or even feasible to have a network administrator to each home or office. SDN technology enables visibility and gives users a view of their network layout. It also offers a single point of control. Furthermore, there is the option of assigning the network management to a third party, through remotely programmable switches, distributed network monitoring algorithms, and conclusion exporting, for the detection of possible security issues.

- Internet Exchange Points (IXP). IXP stations use BGP[5] for routing between sectors, which has certain limitations since it can forward traffic based on destination's IP prefix only. SDN deployment on IXP may provide liberation from these limitations, load balancing etc.
- Backbone Networks. An SDN architecture in backbone networks[6] can provide high programmability and availability. A notable example is Google. Generally, by following a centralized management approach, there are beneficial results like better network utilization, easier network testing etc.

2.1.3 OpenFlow Protocol

Networking technology has developed and improved in a large scale through innovative transformations and mechanisms, boosting speeds, increasing reliability and security of networks [13]. Network devices developed on the physical layer, providing high capacity connections, improving computing power and various applications, offering tools for easy inspection of functions. However, the network structure hasn't seen so many changes from its early days.

In the existing structure, various network devices from different vendors executing different software, or in fact "firmware,"[7] are handling tasks that compound the network functionality in its whole, like routing, packet transportation or network access decisions [13]. This doesn't leave enough room for for innovative research ideas, like new routing algorithms which can be tested in large scale environments. Additionally, every effort for new experimental ideas for constructing the network may result in network failure at certain points, a fact that has led to static and non-flexible structure of the network and hasn't been attracting important innovations towards that direction.

The OF Protocol solves this problem by providing the ability to network administrators, of

[5]Border Gateway Protocol is a protocol which is designed to distribute routing information between Autonomous Systemss (ASs) on the Internet [11].

[6]"A backbone or core network is a part of a computer network which interconnects networks, providing a path for the exchange of information between different LANs or subnetworks. A backbone can tie together diverse networks in the same building, in different buildings in a campus environment, or over wide areas" [12].

[7]"Firmware is a specific class of computer software that provides the low-level control for a device's specific hardware. Firmware, such as the BIOS of a personal computer, may contain basic functions of a device, and may provide hardware abstraction services to higher-level software such as operating systems" [14].

practising or applying various capabilities through the software [15]. The OF specification can be found at the ONF organization. The ONF's administrative board consists of the following well known companies: Google, Microsoft, Deutsche Telekom, Verizon, Yahoo and NTT. Also, companies such as HP, IBM, and CISCO, provide hardware that uses OF.

Definition

As the protocol used in SDN, OpenFlow is the first standard that allowed interaction between the data and control plane [15]. The flow tables used in OpenFlow are analogous to the lookup tables used by routers and switches in traditional networks. Through these flow tables, we can implement firewalls or NAT or collect statistical data. Additionally, they provide action rules which are created or modified from a centralized controller. The network administrator is able to program flow control through the controller, and to determine specific route from the source to the destination using the flow based on the packet forwarding processing. It decreases the power consumption and the network management cost by eliminating the procedure of packets processing of the router, since now packet routes are determined from the centralized controller.

OpenFlow Architecture

Three basic components form the OpenFlow network architecture [13]:

1. OpenFlow switches that make the data plane.
2. OpenFlow controllers that make the control plane.
3. A secure control channel that connects the switches with the control plane.

The communication takes place between switches and hosts or switches and switches using the data path, and between controller and switches using the control path, as shown in Figure 2.2.

OpenFlow uses Secure Sockets Layer (SSL) and TLS protocols to secure connections between the controller and the switch [13]. Mutual authentication between the controller and the switch is performed by exchanging certificates signed by each side's private key. However, the controller may be subject to Denial of Service (DoS)[8] and/or Man In the

[8]"A Denial-of-Service (DoS) attack occurs when legitimate users are unable to access information systems, devices, or other network resources due to the actions of a malicious cyber threat actor. A denial-of-service

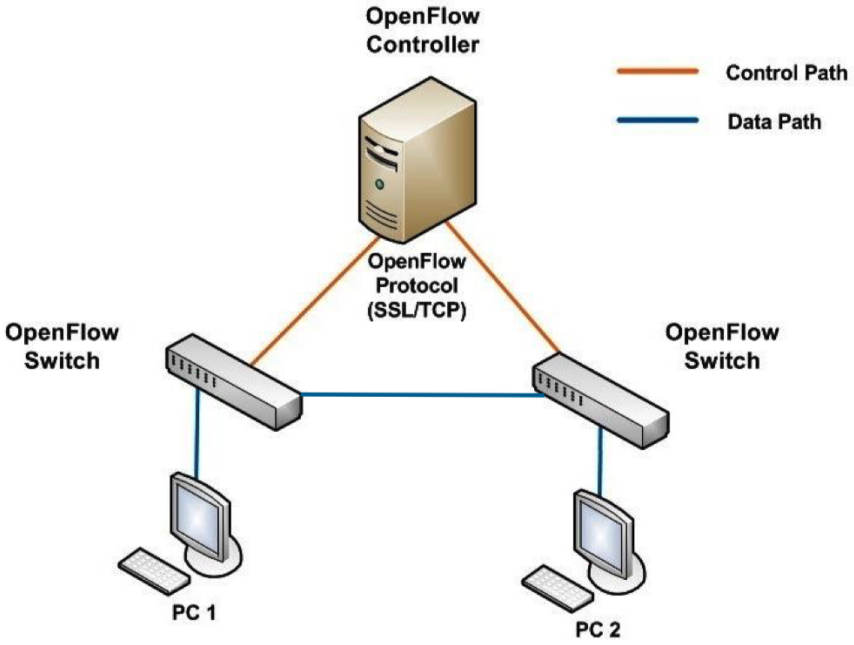

Figure 2.2. OpenFlow Network Architecture. Source: [13].

Middle (MITM)[9]. Therefore, security practises implementation is necessary to prevent such attacks.

OpenFlow Controller

The controller is the "intelligent" part of the network. It maintains network protocols and policies, and delivers instructions to the various network devices. It adds or removes flow entries over the secure channel in switch flow tables, using the OpenFlow protocol. The connection between the switch and the controller takes place with the switch initiating a TLS or Transmission Control Protocol (TCP) connection when it knows the controller's IP address.

There is also the possibility of establishing connections between multiple controllers and a switch [15]. This option improves reliability since the switch can continue working in OpenFlow mode if a connection with one of the controllers fails. The controllers hand-off

condition is accomplished by flooding the targeted host or network with traffic until the target cannot respond or simply crashes, preventing access for legitimate users" [16].

[9]"A form of active wiretapping attack in which the attacker intercepts and selectively modifies communicated data to masquerade as one or more of the entities involved in a communication association attack" [17].

load to each other and this way they are able to maintain a balanced workload between them and to recover from any failure. After all, the goal of this "multiple-controllers" functionality is to contribute to the synchronization of this "handoff" functionality of the controllers.

OpenFlow Switch

The OpenFlow switch is the main device for forwarding packets according to one or more flow tables and a group table [18]. A procedure that is similar to the traditional forwarding tables with the difference that in the traditional ones the tables are not managed or maintained by the switch.

OpenFlow switches can be classified into two different categories, according to [18]:

- OpenFlow-only switches: They support functionality that is only based on the OpenFlow protocol.
- Hybrid switches: They support OpenFlow operation but also normal Ethernet switching operation.

In more detail, the OpenFlow switch consists of one or more flow tables, one group table—which perform lookups and forwarding of packets—a meter table, one or more secure OpenFlow channels, and ports as shown in Figure 2.3.

OpenFlow messages

OpenFlow protocol was created to provide a standardized way of communication between an OF switch and an OF controller [18]. There are three different message types in OpenFlow: controller-to-switch, asynchronous, and symmetric. Each of these types is further distinguished in various sub-types. In the first type, controller-to-switch, the controller is the initiator of the message and this kind of messages allow the controller to manage, control or perform switch status inspection. In the asychronous messages, the switch is the initiator and this type is used for the switch to inform the controller for network events or changes in its status. In symmetric either one is the initiator and this type does not require a request. Below, we describe the message types used by OF, summarized from [18]:

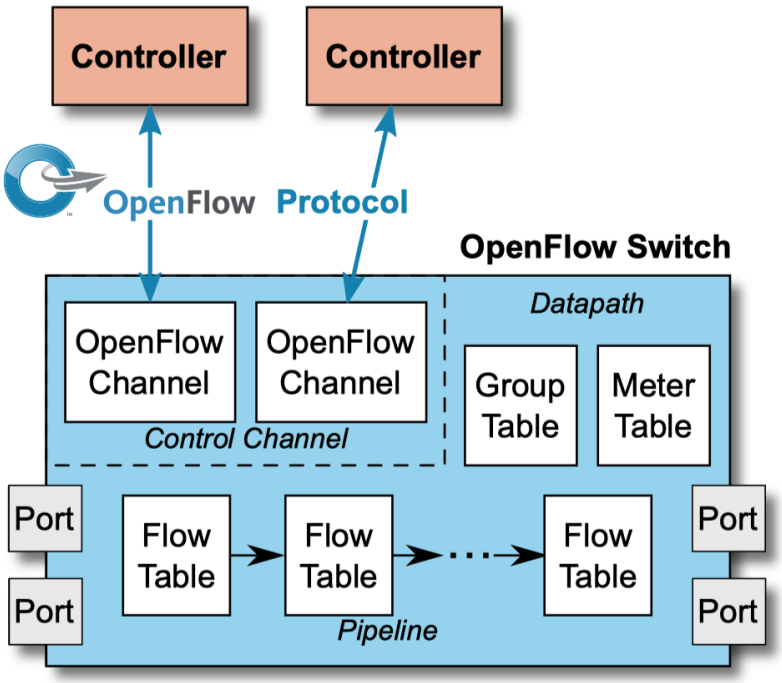

Figure 2.3. Main Components of an OpenFlow Switch. Source: [18].

- Controller-to-Switch
  - Features: By a features request, the controller may request can get basic features knowledge of a switch. The switch responds by sending its identity and basic capabilities. This sub-type is usually used while establishing the OF channel.
  - Configuration: With the use of queries, the controller is able to configure the switch.
  - Modify-State: With this sub-type the controller is able to perform modifications on the state of the switches.
  - Read-State: In this sub-tybe the controller is not able to perform any kind of modifications but is able to receive information about capabilities, statistics or switch configuration.
  - Packet-out: With this sub-tybe, the controller sends packets using a specific port on the switch. The corresponding messages are Packet-in, which the controller receives and/or forwards.

- Barrier: With this sub-type the controller checks message dependencies is notified for completed operations.
- Role-Request: In this sub-type the controller sets its OF channel role and/or its ID.
- Asynchronous-Configuration: In this sub-type the controller can configure filters on the messages it can receive. This is also usually used while establishing the OF channel.

• Asynchronous
  - Packet-in: We mentioned the packet-out messages above and this is the corresponding sub-type. The controller can control these packets. For all packets that are forwarded to the switch and are not matched with any of the flow entries, there is an event created and sent as a packet-in event. If the event's goal is to configure a packet, and if the switch has enough memory to temporarily save (buffer) that packet, then the message that is sent, contains only the required from the controller information, which is about the packet header and a buffer-ID. For switches that do not support the internal temporary save, or do not have enough memory, the message that is sent to the controller contains the full packet.
  - Flow-Removed: This sub-type is about providing information to the controller about a change in the flow table. Specifically if a flow entry was removed, usually after the controller requested for it.
  - Port-status: Switches send port-status messages to the controller, informing of changes on ports.
  - Role-status: Providing information to the controller if there is a change on its role.
  - Controller-status: A switch informs controllers when the status of an OpenFlow channel changes. This can help in redirection if controllers have a failure of communication to each other.
  - Flow-monitor: This is also a sub-type for changes in the flow table. In comparision, with the "flow-removed" sub-type. this one informs and keeps track of all kind of changes in the flow table and not only when it comes to a flow entry's removal.

- Symmetric
  - Hello: Upon connection establishment, the switch and the controller exchange Hello messages.
  - Echo: These messages are sent to check liveness on the other side of the connection. With an echo message we expect an echo reply. They are also used to check latencies or bandwidth.
  - Error: Error messages are used to check for errors in the connection. The switch uses these messages to inform for a controller's request that failed.
  - Experimenter: This is a sub-type of messages that enable embedded functionality. They are also meant for additional features in future OpenFlow revisions.

2.2 Secure Communication

We have already mentioned that SDN uses the Openflow standard which relies on TLS. Unfortunately, TLS is vulnerable and questions arise for the security of SDNs. A security analysis of OF is given in [19], where the authors evaluate various attacks such as denial of service and information disclosure, by assuming an adversary that has access to the data plane.

To be able to talk about secure communication in the dialected protocol, we introduce security primitives and definitions, such as entity authentication and key agreement.

A basic definition, for example, is Entity Authentication:

Definition 2.2.1 (Entity Authentication [20])**.** *Entity authentication is the process in which one party participating in a protocol P is assured of the identity of a second party involved in P, and that the second has participated also, either in the present or immediately prior to the current time that the evidence is acquired.*

We will present more on secure communication and definitions in Chapter 3.

2.3 Related Work

Previous research [3], [2] was performed, in which the authors analyzed the OpenFlow protocol's weaknesses, and proposed a dialect approach using proxies in an effort to strengthen

security in SDNs. A protocol dialect option for OpenFlow was designed, conducted and tested in the Mininet simulation environment using proxies. The analysis of that protocol takes place in two stages, what the authors call derivatives or dialect sub-solutions, as shown in Figure 2.4. On the first stage TLS is not yet established. The second stage starts after the TLS handshake. In D_1 an authentication code is merged into the 32-bit transaction-id field of the original OpenFlow Hello message. In D_2 the packet is not modified and a wrapper is put as the additional security to protect every OpenFlow message. The authentication process in D_1 is performed before the TLS is enabled while in D_2 it takes place after the TLS handshake is established.

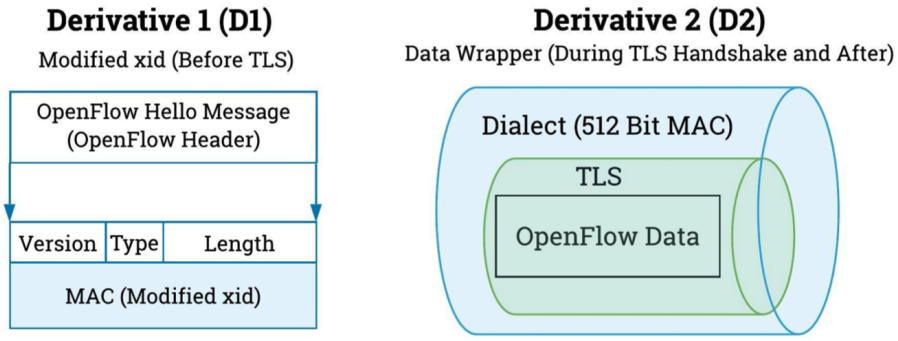

Figure 2.4. Illustration of Dialecting of Existing OpenFlow Messages to support the new Derivatives. Source: [2]. D_2 outer(blue) indicates the derivative while the inner (green) shading indicates the TLS wrapper.

In this chapter we presented the background of the main components of our work and where this research rooted from. On the next chapter we start getting in a more specific view of the core of this research, the security in SDNs.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Security and Preliminaries

In the previous chapter we presented the backgroung of our work, while on this chapter we get more specific and introduce security and preliminaries of our analysis.

3.1 SDN Security

Although SDN provides new network implementations, security has become a main concern in this technology [21]. Research has shown that various attacks are possible against SDN through various elements of these networks. Since SDN depends on software, vulnerable coding points impact SDN security. Furthermore, SDN provides numerous abilities for security check implementation through the controller applications. More functional means for network security checks result from this kind of software solutions.

3.1.1 SDN Threat Model

Every single one of SDN architecture's elements can contain possible vulnerabilities, resulting to exploitation of an attacker and harming of the network [21]. To understand the SDN's threat model, i.e possible threats on SDN networks, each of the threats can be categorized in 3 elements [21]:

1. Threat source: A source that causes the vulnerability.
2. Vulnerability source: An SDN element on which the vulnerability issue is created.
3. Threat action: An activity with which the threat is performed.

Threat sources can further be categorized [21]:

- A non-SDN element: A system that is not a part of SDN architecture.
- A misleading SDN element: An unauthorized SDN system inside an SDN network which performs unauthorized activities.
- A malicious SDN application: An application which has been exposed or a user who perfoms malicious activities using this application.

- A malicious controller: A compromised controller or a user that performs malicious activities on the controller.
- A malicious network component: A compromised network element or a user, who performs malicious activities using that element.
- A malicious management console: A management console that has been exposed or a user who performs malicious activities using that console.

Each of the SDN architecture elements listed below can be a possible vulnerability source [21]:

- An application
- A controller
- A network component
- A management console
- Northbound interface
- East/westbound interface
- Southbound interface
- Management interface

3.1.2 Possible attacks

Various attacks are possible against SDN or the OpenFlow protocol [21]. These attacks can aim on unauthorized access, unauthorized information disclosure, unauthorized modifications, or service disruption. Various attack scenarios are listed below, summarized from [21]:

Unauthorized access with the use of Brute-Forcing or Password-Guessing attacks. An intruder penetrating a non-SDN element can achieve unauthorized access in an SDN element. For example, if the intruder access a management console through random or systematic use of Password-Guessing, then that console could authorize the intruder with resources to deploy attacks against the controller and the network which the controller manages.

Unauthorized access with the use of remote applications exploitation attacks. Taking advantage of a software vulnerability in an SDN element, an intruder has the opportunity

to gain unauthorized access to that element. For example, with access to the management console, an intruder can take advantage of a vulnerability of buffer overflow in an SDN server and gain access to SDN applications.

Unauthorized information disclosure with the use of RAM scraping attacks. The intruder gains unauthorized access to an aimed system first and the sweeps its physical memory, hoping to extract sensitive information.

Unauthorized information disclosure with the use of Application Programming Interface (API) exploitation attacks. The controller plane provides APIs to applications for information searching, monitoring and response in network modifications. Applications that are created using APIs may be vulnerable towards an intruder that performs various information disclosure attacks.

Unauthorized destruction using API exploitation attacks. The attacker takes advantage of a vulnerability in the northbound interface land is able to delete network flows to block traffic from reaching its destination.

Unauthorized information disclosure with the use of Sniffing attacks. Sniffing attacks, is about stealing data by recording network traffic using a sniffer (an application that aims in receiving network packets). An attacker can performing a sniffing attack to exploit non or weakly encrypted communication and gain access to information about the network, which can result to the network collapse or destruction.

Unauthorized modification using identity destruction attacks. An attacker can bypass the identity of a legit controller and attempt to interact with a network component, so as to create instant flow entries in the network's flow table.

Service interruption using remote application access attacks. This attack could lead in the decrease in availability of an SDN element. For example, the attacker can take advantage of an authentication vulnerability in the controller's software or any network element and make it unavailable.

Unauthorized information disclosure using attacks against the channel. An attacker can perform an attack through the channel to see if there are any flow rules by detecting a difference in the time that is necessary for a connection establishment.

Service interruption using flooding attacks. An attacker could use an exposed network element aiming to flood a controller with network messages and exhaust its resources.

Unauthorized modification using data forgery attacks. A network element that has been breached could forge network data and affect the controller's aspect of the network topology.

3.2 Preliminaries

In this Section we introduce and show the basis for what is about to follow. How we choose to use certain security primitives, how these primitives bind with our protocol analysis and why they are necessary. We also recall the required definitions for our result on the OpenFlow protocol.

Message Authentication Codes One important property in our analysis comes with the use of a MAC. In addition to providing authentication, a MAC provides unforgeability. In the definition below we refer to a probabilistic polynomial time algorithm, or else a probabilistic polynomial time Turing machine (PPT), that is equipped with a spare randomness tape.

At this point, it is worth mentioning two security properties, that are closely connected to MAC's. The security property of Strong Unforgeability under Chosen Message Attack (SUF-CMA) and the Existential Unforgeability under Chosen Message Attack (EUF-CMA). The difference between the two is that in the former, an attacker cannot forge a tag on a message he did not ask the MAC for from a challenger oracle, and in the latter he cannot generate another tag for a message he already asked the MAC for.

Definition 3.2.1 (Message Authentication Code (MAC) [22])**.** *A strongly-unforgeable message authentication code is a probabilistic polynomial-time algorithm* $MAC_{(\cdot)}(\cdot)$*. Let* $m = \{0,1\}^*, mKey = \{0,1\}^k$ *for some number* k*, and* $t = \{0,1\}^{tLen}$ *for some number* $tLen$*.*

A message authentication code is defined by a pair of algorithms ($MAC_{(\cdot)}(\cdot)$, $MAC.ver_{(\cdot)}(\cdot,\cdot)$)*. To compute the MAC,* $MAC_{(\cdot)}(\cdot)$ *takes a key* $K \in mKey$ *and a message* m *and computes:*

$$t = MAC_K(m) \ .$$

The authenticated message is the pair (m,t)*;* t *is called the tag on* m*.*

To verify a purported message-tag pair (m, t), any entity with key K computes

$$MAC.ver_K(m, t) \ ,$$

which returns either 0 (message unauthentic) or 1 (message authentic). It is required for all $K \in mKey$ and $m \in \{0,1\}^*$, $MAC.ver_K(m, MAC_K(m)) = 1$. We abuse notation and also write $t = MAC_K(m)$ as $t = MAC(K, m)$, and $MAC.ver_K(m, t)$ as $MAC.ver(K, m, t)$.

An adversary $\mathcal{A}$ is a probabilistic polynomial-time algorithm which has access to an oracle that computes MACs under a randomly chosen key K'. The output of $\mathcal{A}$ is a pair (m, t) such that (m, t) was not previously output by the MAC oracle.

A MAC is SUF-CMA-secure if, for every adversary $\mathcal{A}$ of the MAC, the function $\epsilon(k)$ defined by

$$\epsilon(k) = P[K' \leftarrow \{0,1\}^k; (m, t) \leftarrow \mathcal{A} : 1 = MAC.ver_{K'}(m, t)]$$

is negligible. Note that $\mathcal{A}$ wins even if it outputs a different tag on a previously queried message.

In our protocol, we are limited by the 32-bit transaction-id field in the OpenFlow Header, as it is shown in Figure 3.1. That field is used by the dialect, to incorporate the use of a MAC function. Since, 32-bits provide a relatively small field for the use of a MAC, we balance that issue by introducing time intervals and the killing "off" keys after a certain time interval. Furthermore, we restrict adversarial access to a MAC oracle to one message, using the following notion of an One-Time MAC function.

We define a one-time MAC and corresponding security experiment similarly to a one-time signature from [23]. We also abuse notation and use sEUF-1-CMA to denote strong security against *existential forgeries under adaptive chosen message attacks* similarly to [23].

Definition 3.2.2 (One Time MAC)**.** *A one-time MAC scheme, OTMAC, consists of three probabilitstic, polynomial time algorithms (OTMAC.Kgen, OTMAC.MAC, OTMAC.Vfy).*

- *$OTMAC.KGen(1^\lambda) \rightarrow k_{OT}$: takes as input a security parameter λ and outputs a key k_{OT}.*

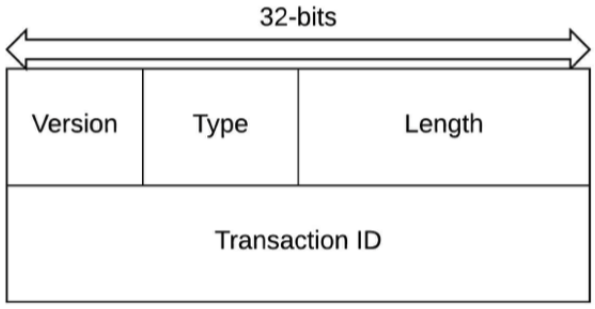

Figure 3.1. OpenFlow Header Layout. Source: [3].

- $OTMAC.MAC(k_{OT}, m, l) \rightarrow t$: *takes as input a key* k_{OT} *and a message string* $m \in \{0,1\}^*$*, and outputs a tag* t *of length* l.
- $OTMAC.Vfy(k_{OT}, (m,t)) \rightarrow$: *takes as input a key* k_{OT} *and a message-tag pair* (m,t). *If* t *is a valid tag for* m *under* k_{OT}*, then the algorithm outputs 1 (accept), else it outputs 0 (reject).*

We say that an OTMAC algorithm is correct *if, for all* $K \in mKey$ *and* $m \in \{0,1\}^*$, $|OTMAC.MAC(k_{OT}, m, l)| = l$ *and* $OTMAC.Vfy(k_{OT}, (m, OTMAC.MAC(k_{OT}, m, l)) = 1$.

Consider the following security experiment $\text{Exp}^{sEUF-1-CMA}_{\mathcal{A},OTMAC,l}(\lambda)$ with an adversary $\mathcal{A}$ against the one time strong security against *existential forgeries under adaptive chosen message attacks* (sEUF-1-CMA) security of the MAC, play with a challenger.

1. The challenger computes $k_{OT} \leftarrow OTMAC.KGen(1^\lambda)$ and runs $\mathcal{A}$ with input k_{OT}.
2. $\mathcal{A}$ may query one arbitrary string m to the challenger. The challenger replies with $t \leftarrow OTMAC.MAC(k_{OT}, m, l)$.
3. $\mathcal{A}$ eventually outputs a message string m^* and a tag t^*. We denote the event that $OTMAC.Vfy(k_{OT}, (m^*, t^*)) = 1$ and $(x^*, t^*) \neq (x, t)$ by
$$\text{Exp}^{sEUF-1-CMA}_{\mathcal{A},OTMAC,l}(\lambda) = 1 \ .$$

The above one-time experiment restricts the number of verification queries that an adversary may make. We allow further parameterization of the above experiment by a time duration

restriction, t_d, for the maximum allowed actual time duration between initialization in Step 1 and completion in Step 3. This is denoted $\text{Exp}^{sEUF-1-CMA}_{\mathcal{A},OTMAC,l,t_d}(\lambda)$.

Definition 3.2.3 (Security of One-Time MACs)**.** *We define the advantage of an adversary $\mathcal{A}$ in the game* $\text{Exp}^{sEUF-1-CMA}_{\mathcal{A},OTMAC,l}(\lambda)$ *as*

$$\text{Adv}^{sEUF-1-CMA}_{\mathcal{A},OTMAC,l}(\lambda) := Pr[\text{Exp}^{sEUF-1-CMA}_{\mathcal{A},OTMAC,l}(\lambda) = 1]$$

and say that OTMAC is sEUF-1-CMA-secure if

$$\text{Adv}^{sEUF-1-CMA}_{\mathcal{A},OTMAC,l}(\lambda) = \text{negl}$$

for a neglible function negl *in λ.*

The time-duration variant parameterization follows similarly to the above.

If $\text{Adv}^{sEUF-1-CMA}_{\mathcal{A},OTMAC,l}(\lambda)$ is a negligible function in λ for all probabilistic polynomial-time adversaries $\mathcal{A}$, then we can say that a one-time MAC scheme is also strongly secure against existential forgeries under chosen message attacks (sEUF-1-CMA).

For the rest of the paper we will be using the OTMAC notation instead of a simple MAC when it comes to D_1, corresponding to the definition of the One-Time MAC above. That is a deviation from [3]. We consider it necessary though, since it enhances security of the protocol. The reason is that given the shortness of the tag in D_1 – only 32 bits – the adversary would be able to successfully guess the tag in a relatively small number of guesses. By using an one-time MAC in the protocol, we restrict the number of MAC queries that are possible under a given key (the application only allows one attempt per timestamp). This is chosen to improve security against brute-forcing attacks.

For D_2 we will simply using the MAC notation, since we do not have the restriction of the 32-bit field anymore and thus there is no need for an one-time scheme.

Pseudo-Random Functions. Another security function used in our protocol is a Key Derivation Function (KDF). A KDF is a cryptographic hash function which generates secret keys using a pseudo-random function [20]. KDF takes the initial keying material, assuming randomness and derives from it strong secret keys.

Hash Functions. Hash functions are used to convert long sequences of bits into smaller ones with a fixed number of bits. They are used in many basic cryptographic schemes. They are designed to be one-way functions. That is, given the output it is hard to determine any information about the input.

Hash function are the main building component of Key Derivation Functions (KDF). In particular, a KDF is supposed to use the one-way and collision resistance properties of cryptographic hash functions [24].

Definition 3.2.4 (Pre-image Resistance [25])**.** *Given $y \in \{0,1\}^{\lambda}$, it should be hard to find a value x such that $H(x) = y$.*

Definition 3.2.5 (2nd Pre-image Resistance [26])**.** *For a hash function, given x, it should be hard to find a value $x' \neq x$ such that $H(x) = H(x')$.*

Definition 3.2.6 (Collision Resistance [27])**.** *A hash function H is collision resistant if for all adversaries $\mathcal{A}$ that run in time t, if it holds that:*

$$Pr[\mathcal{A}(H) = (x, x') : (x \neq x') \wedge H(x) = H(x')] \leq \epsilon_H \ .$$

3.3 Variables and Notation

In this Section we introduce all the variables and notation we will encounter throughout the paper. We also make some early observations about timestamps and their use in our protocol analysis.

3.3.1 Notation

- $S \in \mathcal{S}$: denotes a specific Switch from the set of all switches
- C: denotes the Controller
- K_{SC}: Key derived for message exchange of device S to device C (resp. K_{CS})
- I: the set of identities, $I = \{C\} \cup \mathcal{S}$
- π_i^{S}: an oracle instance i at switch S (resp. π_j^{C})
- D_1 (resp. D_2): Dialect 1 (resp. Dialect 2)
- $K_{SC}^{\mathrm{D1}} \in \mathcal{K}_1$: Pre-established temporary shared secret key for D_1 for messages from S to C (resp. K_{CS}^{D1})

- $K_{SC}^{\mathrm{D2}} \in \mathcal{K}_2$: D_2 session key generated with K_{SC} as input for messages from S to C (resp. K_{CS}^{D2})
- $K_{SC}^{\mathrm{D1}'}$: a new time interval D_1' key generated with K_{SC} as input (resp. K_{CS})
- sid: a session identifier. We calculate sid = D_1 OTMAC Switch $\|$ D_1 OTMAC Controller. [10]
- SN_S: a sequence number for party S (resp. C)
- TS_{S}: A timestamp at party S (resp. C)
- $l = 32bits$: the OTMAC length parameter

3.3.2 Clock Variables

Each party has a clock variable, initially set to 0. All instances of a protocol running at a party can access ($read$) that variable. The adversary has access and control of the clocks, with the ability to *increment* any clock forward.

Timestamps start counting in milliseconds from the moment that the first OpenFlow "Hello" message is sent. Timestamps' format looks like $[HH : MM : SS]$, where:

- $HH = hours$
- $MM = minutes$
- $SS = seconds$

With the use of timestamps we are able to calculate the time window (w). The time-window is defined as the acceptable duration in which each party accepts messages, and is calculated with the use of timestamps. We will see how that works in detail in Section 4.2. For now it is enough to know that the time-window is considered to be valid if $0 \leq w \leq 2sec$.

In this chapter we talked about security in SDNs and saw the preliminaries of our analysis. In the next chapter we will see our dialected protocol's description in detail, as well as the explanation of the deconfliction algorithm we use in our work, a concept we needed since our protocol utilizes timestamps.

[10] With OTMAC here we refer to the OTMAC tag, i.e $t = OTMAC_{K_{\mathrm{SC}}}(TS_{\mathrm{S}}, msg)$, not the hardware MAC address.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
A Protocol Utilizing Timestamps

In the previous chapter we presented the preliminaries of our work as well as variables and notation used throughout our work. Here we see the protocol description in detail and also explain the deconfliction algorithm, an algorithm necessary for the use of timestamps in our protocol.

4.1 Protocol Description

The design choices in the original dialected OF protocol, include a Hash Key Derivation Function (HKDF) based on a hash based message authentication code (HMAC). For the needs of our analysis and proof, referring to it as MAC or HMAC, does not causes any contradictions, since proving security for a MAC indicates properties needed from the underlying HMAC too.

Protocols Phases

Our protocol is considered a two party mutual authentication protocol, between a switch and a controller in the SDN environment. Below, we summarize each phase of the protocol, provided in [3], and shown in Figure 4.1.

- **"Phase 0":** Initial (D_1 in Fig. 4.1) Key Derivation.
  Before implementation of D_1 and D_2, "a pre-shared symmetric key K, generated at random, is established and delivered out-of-band (OOB)" [3]. As shown in (0a,0b) in Fig. 4.1, we have:

  $$0a.\ K_{SC} = KDF(K, TS_S, S, C)$$
  $$0b.\ K_{CS} = KDF(K, TS_C, C, S)$$

  The sending party computes the OTMAC with the key and the receiving party verifies it. The same happens on the receiver's side.

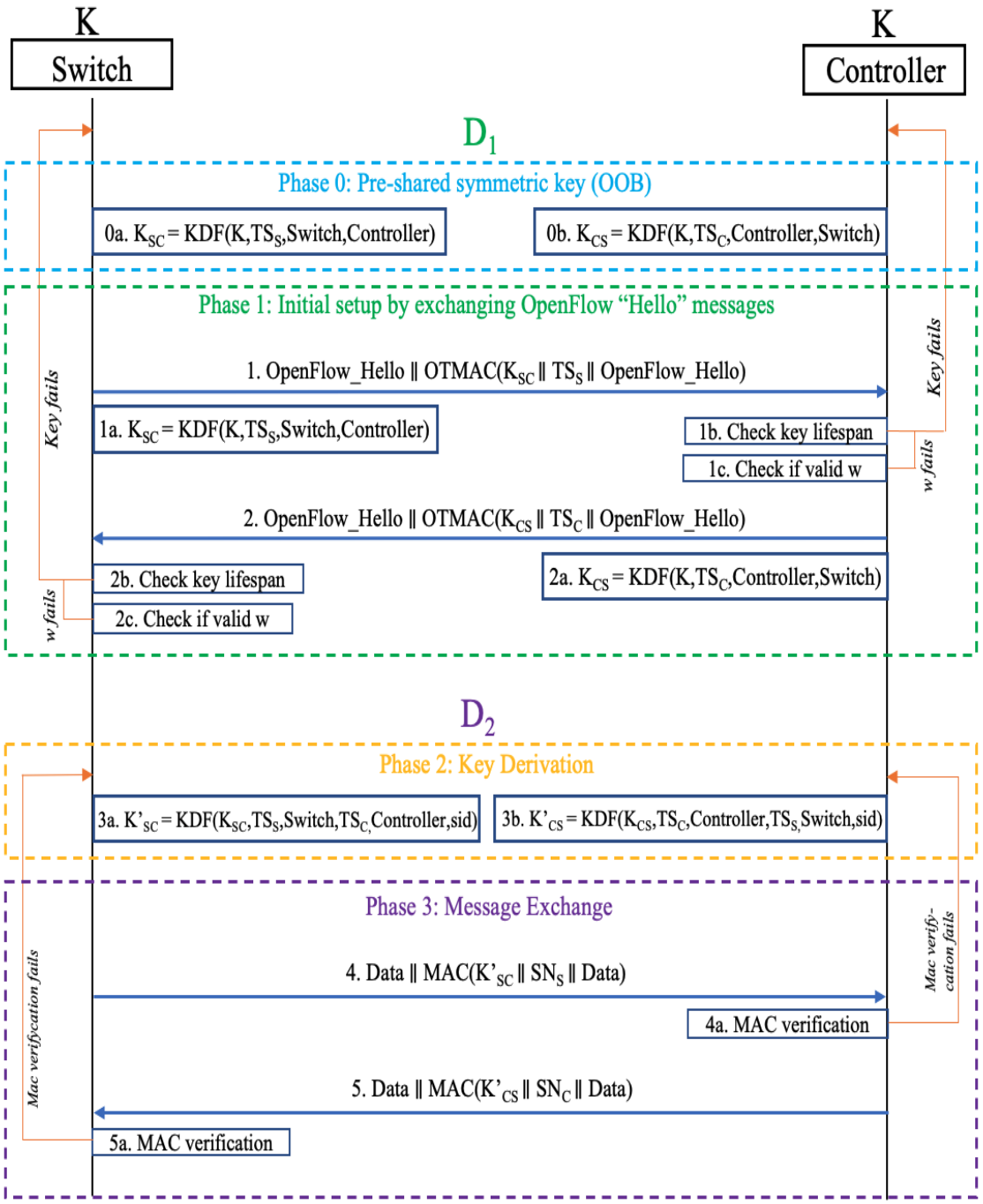

Figure 4.1. Dialecting Protocol Flow Diagram. sid is defined as the concatenated OTMAC tags from Steps 1 and 2.

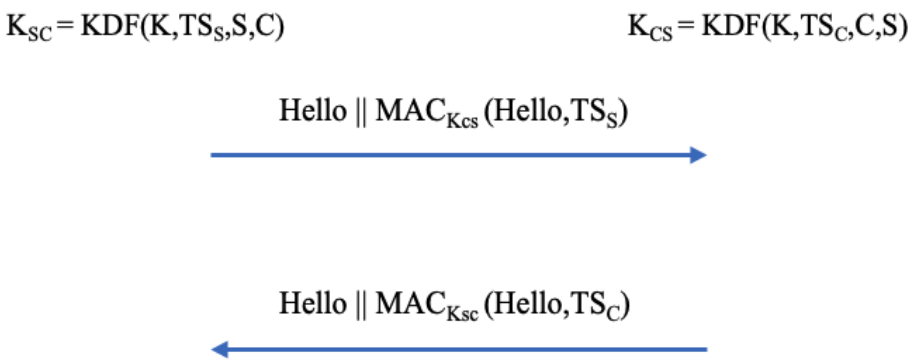

Figure 4.2. Simplified View of Dialected OpenFlow Hello exchange messages in "Phase 0".

- **"Phase 1":** Initial setup by exchanging OpenFlow "Hello" Messages (D_1).
  The initiator party (the switch) sends an Openflow Hello message (Line 1 in Figure 4.1), appending it with a OTMAC tag of length $l = 32$, using the already established pre-shared symmetric key (1a in Figure 4.1), a timestamp generated at his end (requesting it from local clock authority, see Section 4.2) and an OpenFlow Hello message. The receiving party (the controller) (in fact the proxy standing in front of the controller) will check the validity of both the key and the timestamp (1b, 1c in Figure 4.1). We have already mentioned the valid lifespan of a key being 1 second and the valid time window (w) of the sender's timestamp being 2 seconds (see Section 4.2 for more discussion on time selections).
  If a message fails the authentication check then the receiving proxy will discontinue the set up process. The initiator will have to start over. We assume that the K_{ij} keys are not reused for any other pair in any other connection.
  Otherwise, the receiving party sends another OpenFlow Hello message (Line 2 in Figure 4.1) as shown in Figure 4.1. If that message gets accepted, the communication is considered established.
  Freshness is achieved by implicit timestamps using the rejection loop described in Section 4.2.
  In D_2 we do not require the key lifespan because, unlike the OTMAC of length $l = 32$, in D2 we use a full length MAC tag. We still need unique keys for each communication.

- **"Phase 2":** D_2 Key Derivation (D_2 in Fig. 4.1). While D_1 gets completed, the two communicating parties' proxies create uni-directional keys for D_2. Keys are derived as follows and are shown in (3a, 3b) in Figure 4.1, respectively:

$$\text{3a. } K'_{\text{SC}} = KDF(K_{\text{SC}}, TS_{\text{S}}, S, TS_{\text{C}}, C, sid)$$
$$\text{3b. } K'_{\text{CS}} = KDF(K_{\text{CS}}, TS_{\text{C}}, C, TS_{\text{S}}, S, sid)$$

A visual representation of the procedure is shown in Figure 4.3.

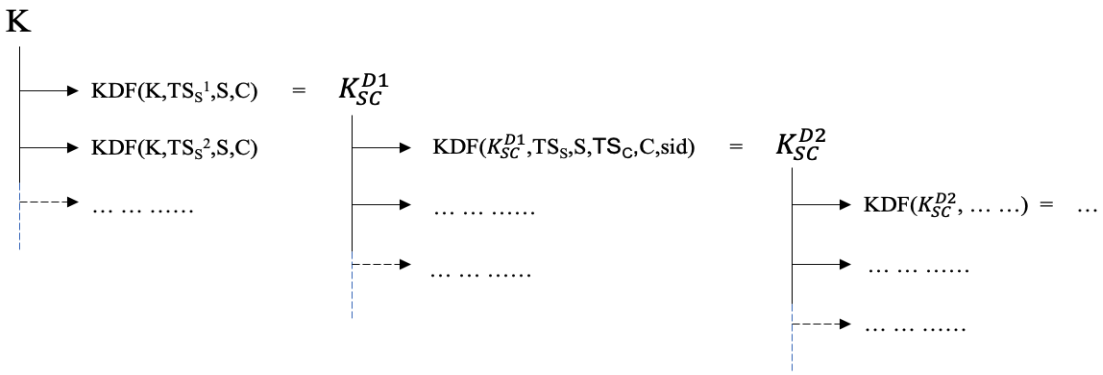

Figure 4.3. Key Schedule Key Derivation.

Uni-directional keys: Each key derivation has identifiers for the sending and the receiving party. That way, the sender of a message knows not to receive that message. That provides the protocol with protection against reflection attacks [28].
The keys are being derived every two minutes in time-intervals, as shown in Figure 4.4.

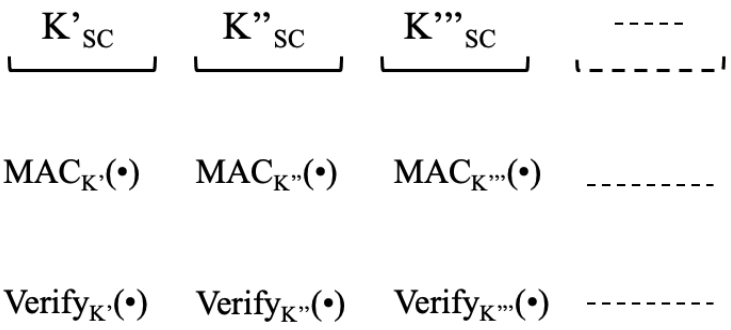

Figure 4.4. Keys derived for different Time Intervals.

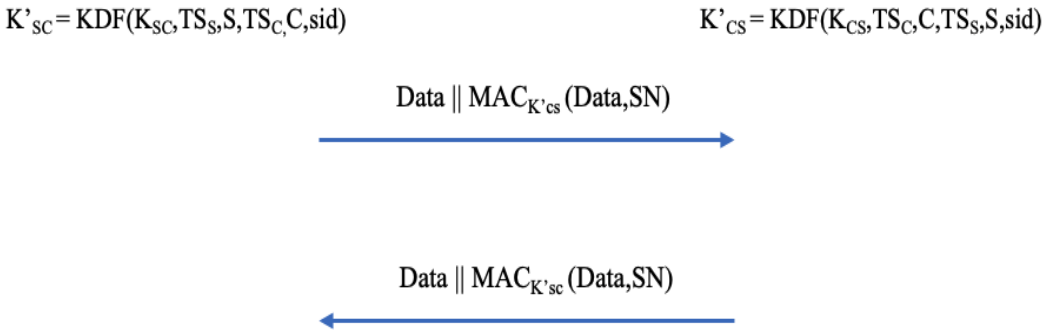

Figure 4.5. Simplified View of Dialected OpenFlow Data Exchange Messages in "Phase 3".

- **"Phase 3":** D_2 Message Exchange. D_2 appends a 512-bit MAC tag to encrypted data packets before sending (Line 4 in Fig. 4.1). The receiving party accepts the message only if it successfully passes the MAC check (4b in Fig. 4.1). D_2 does not modify fields or data in the packets, therefore it can be added to different protocols. The longer field provides a longer key lifetime, among other benefits. Furthermore, sequence numbers in the MAC calculation increment with each new message, something that provides a good way of protection against replay attacks.

  The sending party (switch) sends its data in the clear appended with a MAC calculated on the secret key, a sequence number, and the data (Line 4 in Fig. 4.1). A sequence number increments with each new message in a session. We can think sequence

numbers as nonces with the difference that they are not random and thus they can be predicted [26]. They are used to detect replayed or out of order messages.
The receiving party (controller) will perform the HMAC check (4a in Fig. 4.1). If the message passes the check, it will accept. Then it will send back its own data appended with a MAC calculated on its secret key, a sequence number and the data (Line 5 in Fig. 4.1). That procedure will go on for any number of messages.

In case of key failures or a non-valid time-window in D_1, the whole communication will have to start over with the exchange of the OpenFlow Hello messages (failures are shown as orange lines at the sides of the diagram in Fig. 4.1, pointing at the start, indicating that the whole procedure has to start over). In case of MAC verification failure in D_2 the communication will revert to the last used key, i.e last accepted MAC with confirmed receipt. That is also shown as orange lines at the sides of the diagram in Fig. 4.1.

4.2 Timestamps

4.2.1 Introducing the notion of freshness

With the use of timestamps, the protocol ensures freshness. We have already given a definition for freshness in Section 3.2. The party sending the message includes the current time with the use of a timestamp. That timestamp is checked by the receiving party which compares it with the local time. If the time window (w) is within the limits of what we have selected as a valid (acceptable) value, then the message is considered fresh. However, there are some issues arising from the use of timestamps which we will analyze later in this Section. First, we discuss other mechanisms that can ensure freshness and then we analyze what we call the *deconfliction* algorithm, which we use in our protocol. We also discuss other algorithm choices for agreement on timestamps that exist.

4.2.2 Mechanisms that ensure freshness

Freshness, as a term, indicates a value that has not been seen before [20]. Thus, either something that is freshly created, or that is used for the first time. The reason that we need to ensure freshness in values used in our protocol, is quite obvious. We want to ensure that values are not repeated or re-created. Therefore, we ensure that our protocol is secure against replay attacks.

There are generally two ways to ensure freshness in a protocol [20]. One method is through selection/receipt of new keys (e.g. ephemeral keys) or through selection/receipt of short values, such as timestamps, nonces (random challenges), and sequence counters. The user, either chooses the value (i.e a new session key), or depends on receiving something ac-

companied with a value that is ensured fresh (i.e a timestamp). We discuss these options below:

Keys. Use of ephemeral keys may contribute to freshness. A fresh ephemeral key can be chosen for each party in the protocol run. Subsequent use of those keys by both parties in creation of the session key then ensures freshness of the protocol run.

Timestamps. The sending party adds a timestamp of the current time, to the message it sends [20]. The receiving party checks that timestamp, when it receives the message, by comparing it to the local time. If the timestamp's value falls within an acceptable time window, then the message received is considered fresh. Timestamps are used in this research and are discussed further in Section 4.2. In the same Section we also discuss issues arising from the use of timestamps, i.e clock de-synchronization.

Nonces. One party generates a nonce (unique number that is used only once) and sends it to the other party [20]. That nonce may be later returned in the message sent after being modified with a cryptographic function. The receiver checks that the nonce has not been used before, and therefore knows that the message is fresh. A disadvantage in the use of a nonce is that it can add to the number of messages being exchanged making the protocol slower and less efficient [29]. It may also require memory caching to ensure that the nonce has not been previously used.

Counters. Both parties keep a synchronized sequence counter from which they select the current value to send with the message [20]. Then the counter gets incremented. A disadvantage here is that both parties need to maintain state information, which again, can affect the efficiency especially when we talk about a Controller which may maintain communication with many switches at a time.

4.2.3 Deconfliction Algorithm

We mention our valid time window which is $0 \leq w \leq 2sec$. To use this time window (w), we analyze algorithm choices for the agreement of controller and switch on timestamps. Time plays an important role in our protocol dialect implementation. Furthermore timestamps potentially give a simple way to kill "off" the keys to increase security, if used correctly.

Deconfliction Algorithm. Assume a case of increasing local time based on a set increment interval. In this approach each of our parties has a local timer. When party S wants to send a message to C it requests a timestamp TS_{S} from the local timer and puts the timestamp in the appropriate field in the protocol message. When the message is received by party C, it generates its own timestamp TS_{C} from its own local timer. Then the values TS_{S} and TS_{C} are compared. So, the message is further validated by subtracting $TS_{\mathrm{C}} - TS_{\mathrm{S}}$ to get a range value. This is the value we need to check for validity. We use the absolute value of $TS_{\mathrm{C}} - TS_{\mathrm{S}}$, because we want our valid time window interval to equally be distributed with an upper and lower range of 2 sec (for 4sec altogether). If $\lceil | \, TS_{\mathrm{C}} - TS_{\mathrm{S}} \, | \rceil \leq 2sec$ the message is accepted; and rejected otherwise. Furthermore, in this work we say that TS_{C} and TS_{S} are *uniquely deconflicted* if $\lceil | \, TS_{\mathrm{C}} - TS_{\mathrm{S}} \, | \rceil \leq 2sec$.

Implicit Timestamps In the protocol described in Figure 4.1, the timestamps TS_{C} and TS_{S} are used to calculate keys and are not sent explicitly in the protocol flows. That is, they are not used as input in the protocol data packet fields but in the key derivation and tag derivation for the OTMAC instead [2]. Thus, the deconfliction algorithm use in the protocol must account for *implicit* timestamps. Without a concrete TS_{C} being received, S cannot compute the ceiling function directly. Instead, S uses its own timestamp TS_{S} as a guess at TS_{C} in the calculation of the OTMAC verification.

Since the timestamp input into the OTMAC must be exact (given the avalanche property of the MAC), there is not a margin for error and the OTMAC will not verify correctly if $TS_{\mathrm{C}} \neq TS_{\mathrm{S}}$. We note that the incrementation interval is therefore important. S may calculate all potential timestamps TS such that $\lceil | TS - TS_{\mathrm{S}} \, | \rceil \leq 2sec$, and use each permitted TS as a test verification value for the OTMAC; however, if the number of timestamps $|\{TS\}|$ is too large, then this method is infeasible.

That is yet another reason why protocols using timestamps are difficult to verify. There must be no significant delays in the protocol flows. Furthermore, we must make an assumption that the Controller and Switch are able to use a sufficiently similar clock time to avoid the protocol 'rejecting' unnecessarily [2]. With the use of implicit timestamps $D1$ becomes more similar to the OpenFlow standard.

Non-synchronized clocks. In the case that the clocks of the two parties are not synchronized, our algorithm would provide false negatives meaning it would reject messages that should

not be rejected. Let's consider the case where party S sends a message with timestamp 15:15:24. At the exact same time party's C clock shows 15:15:26. When the message is received by party C it compares with local time which is now 15:15:27. As we can see, the message will be rejected having a time window of 3 sec $\nleq$ 2 sec, even if its time-window is actually 1 sec. Furthermore, the algorithm can provide false positives. Let's consider another case where party S sends a message with a timestamp of 15:15:15. At the exact same time party's C local timer shows 15:15:12. The moment that party C receives, its local timer shows 15:15:16 while $S's$ local timer is now 15:15:19. When C is comparing with 15:15:15 timestamp it results in a time-window of 1 sec, therefore C accepts the message. The actual time-window though is 3 $\nleq$ 2 sec and the message should have been rejected. As [H. Massias et al.] conclude in their paper "Timestamps: Main issues on their use and implementation" [30] time is a value that must be published in an unmodifiable media, and the place and frequency contribute to the level of trust and granularity that timestemps provide. This dependency on the published value is what restricts the scalability of the timestamping system.

Buldas and Saarepera [31] provided a formal proof on the security of time-stamping schemes. In their work they are referring to underlying hash functions and hash-based schemes assuming the property of collision resistance. They also suggested a new security condition namely "chain resistance". To explain chain-resistance in summary, the authors write:

> Loosely speaking, if x can be time-stamped based on $y = f(x)$, then x can be efficiently computed based on y, and hence the time stamp is "legitimate". This condition implies chain-resistance if we define $f(x) \equiv 1^k$ [31].

Thus, with this security condition the authors suggest an function using a parameter k that makes it harder for the adversary to compute a "legitimate" timestamp and might provide secure time-stamping schemes.

Rompay, Preneel and Vandewalle [32] give an overview of the timestamping problem and discuss several techniques for timestamp distribution and verification, using the notion of Time Stamp Authority (TSA).

In general, the common idea in most papers includes a trusted centralized service that provides communication parties with a value, that it digitally signs, corresponding to the current time, by using a precise clock [31]. Or as we find it in [32], the TSA, which accepts requests from communicating parties, appends them with the current date and time and digitally signs the result to produce a timestamp.

Issues. Some issues considering the use of timestamps are:

- The receiver must be able to detect replay messages within a time-window by keeping a local state [29].
- Messages with invalid timestamps are discarded but a decision must be made about messages that are repeats within the acceptance window. A solution here might be to store all previously received messages to prevent accepting duplicates. So in a way to use timestamps as sequence numbers.
- If a party's clock is ahead of other clocks its messages can get postdated.
- Schemes may be vulnerable to a suppress-replay attack where a message is suppressed and then replayed [33]. Even if a faulty clock gets re-synchronized this vulnerability remains.

We could treat timestamps as random nonces to simplify the analysis but there would be a trade-off on security [29].

We described our dialected protocol in this chapter and we saw how the deconfliction algorithm used in our protocol works. We also described some issues that we consider from the use of timestamps. Coming next on Chapter 5 is the analysis and the security proof, the core concept of our work.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Security Analysis

After describing the protocol and the deconfliction algorithm in the previous chapter, in this chapter we provide the core of our analysis, the security model and the security proof.

5.1 Security Model

In this Section we explain assumptions and definitions that are required for our computational analysis on the OpenFlow dialect. Our protocol's main goals are Mutual Authentication and Key Agreement (AKA) for a symmetric key protocol. The Authenticated Key Exchange (AKE) model [27] was chosen as a basis for the model we use here.

The security modelling and analysis found in [29] was also beneficial for our analysis, since our protocol relies on timestamps.

5.1.1 Session Variables

We refer to the protocol participants as principals, with the identity set $\mathcal{I} = \mathcal{S} \cup \{C\}$ which consists of the set of switches ($\mathcal{S}$) and the controller (C). During the execution of the protocol we may have running instances of each principal. Those will be what we call session oracles, denoted by π such that instance i of principal S is π_i^S. These instances are like running processes controlled by the respective principal and that is how we can treat them. For the protocol execution we have an instance talking first. That is the initiator of the conversation producing the first message. The instance that replies is the responder. The process can go on for a fixed number of flows until both of the participants terminate which might mean that they have 'accepted' if the protocol runs according to expectations.

We associate with each session π_i^S the following variables:

- $id \in \mathcal{I}$: the identity of the session (i.e., id = S_n for $\pi_i^{S_n}$).
- $pid \in \mathcal{I} \cup \{\perp\}$: the identity of the partner session.

- $role \in \{initiator, responder\}$: the sessions role in the protocol. We require that $role = initiator$ if and only if $id \in \mathcal{S}$, and respectively $role = responder$ if and only if $id = C$.
- $trans \in \{0,1\}^* \cup \{\perp\}$: A transcript of sent and received messages at the session, in chronological order.
- $\texttt{time} \in N$: the time tick which the session occurs in based on a global clock.
- $K^{\texttt{time}} = K_{\text{SC}}||K_{\text{CS}} \in \mathcal{K} \times \mathcal{K}$: A variable storing a session key for time tick time. $\text{K}^{\texttt{time}}$ is initialized to $\perp||\perp$.
- $\Lambda^{\texttt{time}} \in \{accept, reject\}$: A variable indicating acceptance for the current session key. $\Lambda = accept$ if and only if $\text{K}^{\texttt{time}} \neq \emptyset$.

5.1.2 Queries

We can define the adversary $\mathcal{A}$ as a probabilistic polynomial-time (PPT) Turing machine. $\mathcal{A}$ communicates with the session oracles via queries. We suppose that the adversary has control over all communication between the interacting parties (session instances). He can read, send, modify, and delete messages or even create new instances [26]. Additionally he can place the following queries (adapted from [23], [27]):

- $\mathsf{Send}(\pi_i^U, m)$: This oracle sends a message m to session oracle π_i^U, for $U \in \mathcal{I}$. The oracle will respond according to the protocol and its internal state.
  When an attacker sends the first Send query to an oracle π_i^U, the oracle looks to see if m contains a unique initialization symbol $\top$. If true and $U \in \mathcal{S}$, it sets $id = U$, $role = initiator$, $pid = C$, and answers with the first protocol message. Otherwise and if $U = C$, it sets $id = C$, $role = responder$, and responds according to the protocol, setting pid according to the received message. Else, it responds with $\perp$.
- $\mathsf{Reveal}(\pi_i^U, \texttt{time})$: This oracle returns the contents of the session key $K^{\texttt{time}}$ session i of party $U \in \mathcal{I}$ at time interval time. If the adversary $\mathcal{A}$ issues a Reveal query in interval time or in any prior time interval, we say that the protocol at π_i^U is time-revealed. Note that timestamp time and U is consistent across all sessions i. We interpret the *time interval* associated with a timestamp time to include any timestamp $\texttt{time}_l$ such that time and $\texttt{time}_l$ can be uniquely deconflicted (e.g. by a deconfliction algorithms such as proposed in Section 4.2.3).

- Test (π_i^{S}): In the case of π_i^{S} having state $\Lambda \neq accept$, the oracle returns a failure symbol $\perp$. Else, it tosses a fair coin b, selects a independent key $K_0 \leftarrow_{\$} \mathcal{K}$, sets $K_1 = K$ to the 'real' key computed by π_i^{S}, and returns K_b. This query can only be asked once during the game.
- Tick(U): This query increments the clock at party U by one time tick interval.

The adversary has complete control of the time, under the restriction of using the Tick query [29]. This is intended to model the real-world abilities of an adversary that may not rewind the actual clock but may cause it to speed ahead or stall.

The Bellare and Rogaway [26] model, as well as many other models following it, used the notion of matching conversations. While the BR approach is worth mentioning, we are using the definition found in [27] which uses the idea of "transcripts" and provides a more generalized approach to matching conversations. Following that approach, we denote with $\pi_i^{\mathrm{S}}.trans$ the sequence of sent and received messages by π_i^{S} "in chronological order", inclusive of any additional data, where each message is prefixed by a timestamp at π_i^{S}, `time` in which the message was sent (resp. received). Additionaly, $\pi_i^{\mathrm{S}}.trans$ is the transcript of π_i^{S}. For two transcripts $\pi_i^{\mathrm{S}}.trans$ and $\pi_j^{\mathrm{C}}.trans$, we say that $\pi_i^{\mathrm{S}}.trans$ is a *prefix* of $\pi_j^{\mathrm{C}}.trans$, if $\pi_i^{\mathrm{S}}.trans$ holds one or more messages, and the timestamp-message pairs in $\pi_i^{\mathrm{S}}.trans$ "are identical to and in the same order as the first" | $\pi_i^{\mathrm{S}}.trans$ | timestamp-message pairs of $\pi_j^{\mathrm{C}}.trans$. For this we define "identical timestamp" as any timestamp within the same time interval (e.g. for $\mathtt{time}_{S,i}$ at π_i^{S} and $\mathtt{time}_{C,j}$ at π_j^{C}, $\mathtt{time}_{S,i}$ and $\mathtt{time}_{C,j}$ are in the same time interval if they uniquely deconflict such as by a deconfliction algorithm such as discussed in Section 4.2.3).

Definition 5.1.1 (Matching Conversations [27])**.** *We say that π_i^{S} has a matching conversation to π_j^{C}, if*

1. *$\pi_j^{\mathrm{C}}.trans$ is a prefix of $\pi_i^{\mathrm{S}}.trans$ and π_i^{S} has sent the last message(s), or*
2. *$\pi_j^{\mathrm{C}}.trans = \pi_i^{\mathrm{S}}.trans$ and π_j^{C} has sent the last message(s).*

While matching conversations aim to identify partners and to make sure that these partners agree on the messages sent and received in a protocol run they were part of, we also need different roles when it comes to communication between participants in a conversation, e.g. a switch and a controller in our protocol. Sessions can be called partners after agreement on who takes the initiator and who takes the responder role in the run [20].

Jager et. al [27], employ a variant definition of matching conversations (see Definition 5.1.1), in their AKE experiment definition, where the AKA experiment is a game between an adversary $\mathcal{A}$ and a challenger oracle. We use an adaptation of their definitions, below, accounting for timestamps:

Definition 5.1.2 (Correctness [27])**.** *Assume a "benign" adversary $\mathcal{A}$, which picks two arbitrary oracles π_i^{S} and π_j^{C} and performs a sequence of* Send *queries by faithfully forwarding all messages between π_i^{S} and π_j^{C} for delivery within a uniquely deconflicting time interval. Let K_i^{S} denote the key computed by π_i^{S} and let K_j^{C} denote the key computed by π_j^{C}. We say that an AKA protocol is* correct*, if for this benign adversary and any two oracles π_i^{S} and π_j^{C} it always holds that:*

1. *both oracles have $\Lambda =$ accept, and*
2. $K_i^{\mathrm{S}} = K_j^{\mathrm{C}} \in \mathcal{K}$.

We make the natural extension of this for our protocol as $K_i^{\mathrm{S}} = K_j^{\mathrm{C}} = K^{\mathtt{time}} = K_{\mathrm{SC}}||K_{\mathrm{CS}} \in \mathcal{K} \times \mathcal{K}$.

Definition 5.1.3 (Authenticated Key Agreement Protocol Security. Adapted from [27])**.** *We say that an adversary $\mathcal{A}$ (t, ϵ)-breaks an AKA protocol Π, if $\mathcal{A}$ runs in PPT t, and at least one of the following two conditions holds:*

1. *When $\mathcal{A}$ terminates, then with probability at least ϵ there exists an oracle π_i^{S} such that*
   - *π_i^{S} 'accepts' when in time period $\mathtt{time}_S$ with intended partner $pid = U$,*
   - *U is not $\mathtt{time}_U$-revealed for any session j and timestamp $\mathtt{time}_U$ such that $\mathtt{time}_U$ and $\mathtt{time}_S$ are in the same time interval, and*
   - *there is no unique oracle π_j^{U} such that π_i^{S} has a matching conversation to π_j^{U}.*

   If an oracle π_i^{S} accepts in the above sense, then we say that π_i^{S} accepts maliciously.

2. *When $\mathcal{A}$ issues a* Test*-query to any oracle π_i^{S} and*
   - *neither π_i^{S} nor π_j^{U} are* time*-revealed such that π_i^{S} has a matching conversation to π_j^{U} (if such an oracle exists),*

   then the probability that $\mathcal{A}$ outputs b' which equals the bit b sampled by the Test*-query satisfies*

$$| \, Pr[b = b'] - 1/2 \, | \geq \epsilon \, . \tag{5.1}$$

   If an adversary $\mathcal{A}$ outputs b' such that $b' = b$ and the above conditions are met, then we say that $\mathcal{A}$ answers the Test*-challenge correctly.*

We say that an AKA protocol Π is (t, ϵ)-secure, if it is correct and there exists no adversary that (t, ϵ)-breaks it. We define the adversarial advantage of $\mathcal{A}$, $\mathrm{Adv}^{\mathrm{AKA}}_{\Pi,\mathcal{A}}(\lambda)$ as the probability that $\mathcal{A}$ (t, ϵ)-breaks the protocol Π.

This model aims to capture authenticated key agreement for the environment. This is called an experiment environment, where we run the protocol in the presence of an adversary $\mathcal{A}$ that is trying to break it. Using this experiment, we look to prove security of the protocol, i.e. that the adversary is unable to succeed under the presented conditions. In a way, we execute the adversary by giving it input, letting it run and receiving the output. We can also interact with the adversary while it runs. If the adversary successfully completes the task, then it wins.

In a similar way we conduct our proof (in Section 5.2) as a series of games, starting with bounding simple abilities for the adversary and building up to more complex ones, ending up by showing that a strong adversary is unable to break the protocol under the experiment. Some of the abilities that the adversary has, between two communicating parties, include the fact that it can view outgoing messages and create instances or messages and deliver them out of order and to any recipient (according to the model description above). Thus we consider that the communication between parties is entirely controlled by the $\mathcal{A}$.

5.2 Security Proof

In this Section we prove security of the dialected version of the OpenFlow protocol. Our protocol assumes that a session in the initiator role will always be a Switch, which leaves the Controller to always being in the responder role.

We have already mentioned our protocol's main goal in Section 3.2, which is Authenticated Key Agreement. Our proof of security focuses on the model as shown in Section 5.1.

Specifically, we focus on showing security of Dialect 1 as an symmetric AKA protocol with additional data, establishing a session key based on the pre-shared OOB key. Based on the key indistinguishability of the eventually established session key $K' = K'_{\text{SC}}||K'_{\text{CS}}$, the authentication security of Dialect 2 follows by the SUF-CMA security of the OTMAC.

The keys are being derived every two seconds in time-intervals, as shown in Figure 4.4. $\mathcal{A}$ can select any time window and try to verify the key, by incrementing the clock using the Tick query. Because of the imposed protocol restrictions, $\mathcal{A}$ can only try verification once per time interval (this also applies for honest parties). The key changes per interval, so if $\mathcal{A}$ fails, it can try another interval but the experiment will be under a different key (leading to a one-time MAC experiment).

Theorem 5.2.1. *Suppose KDF is a secure PRF and OTMAC is sEUF-1-CMA-secure for a specified OTMAC length l and time interval t. Then Dialect 1 as defined in Figure 4.1 is a secure Authenticated Key Agreement protocol running with time interval t.*

Proof. Let $\mathcal{A}$ be a probabilistic polynomial time adversary oracle (PPT) against the AKA, per Definition 5.1.3.

The proof follows as a sequence of games to test the adversary's abilities, run by a challenger. The first game is simply the security experiment described above. Then several intermediate games follow which modify the game one step at a time and with alignment for the adversary's ability to computationally distinguish it from the game before. We denote Adv^i as the adversarial advantage in Game i.

Game 0: This game is identical to the AKA security experiment described above. Thus, the advantage of the adversary on this game is:

$$\text{Adv}^0 = \text{Adv}^{\text{AKA}}_{\Pi,\mathcal{A}}(\lambda) \ .$$

Game 1: The challengers guesses the identity of the switch from $\mathcal{S}$, where $|\mathcal{S}| = n_p$ and the sessions at S and C from $[n_s]$, and aborts if the guess is wrong.

We bound this step by $n_s^2 \cdot n_p$:

$$\mathrm{Adv}^0 \leq n_s^2 n_p \cdot \mathrm{Adv}^1 \ .$$

Game 2: This game is the same as Game 1, except the challenger guesses the timestamp at S and at C for which the adversary will try to win. If the adversary has q_T queries to the Tick oracle, this can be bounded by

$$\mathrm{Adv}^1 \leq q_T^2 \cdot \mathrm{Adv}^2 \ .$$

Game 3: This game is the same as Game 2, except we replace the keys generated in Step 0 of Phase 0 with randomly sampled keys for the target sessions (K^*_{CS} and K^*_{SC}) for the OTMAC oracles. Game 2 and Game 3 are indistinguishable based on the pseudorandomness of the KDF, for both K_{SC} and K_{CS}.

We bound this step as:

$$\mathrm{Adv}^2 \leq \mathrm{Adv}^3 + 2 \cdot \mathrm{Adv}^{\mathrm{PRNG}}_{KDF,\mathcal{A}_1}(\lambda) \ .$$

Game 4: The goal of the adversary is to make one of the parties accept without a matching conversation. $\mathcal{A}$ can make Reveal queries at will; however if $\mathcal{A}$ ever calls Reveal on either the target session or its partner (if one exists), then the experiment aborts. Thus, it can use the Reveal query on any switch or controller but not the switch or controller session that are in a matching conversation with current session id.

Case 1. The first case considers a controller that accepts maliciously. At some point, C receives a flow of the form `Hello`, $OTMAC_{K^*_{\mathrm{SC}}}(TS_{\mathrm{S}} \| \texttt{Hello})$ that verifies correctly, but was not sent by a valid session at the switch (note that the given switch and the controller already share a symmetric from which the OTMAC key is derived along with a specific "Switch"/"Controller" ordering, so the direction and identity of the sender is unique). Also, the target session and time interval at C and any session at S in the same time interval, are

not `time`-revealed. Therefore the adversary does not have access to the respective keys nor any prior key and forging another timestamp within the time interval is bounded by ability to forge at all.

Since the key K^*_{SC} is specific to the time internal and any second message received within a time interval is silently dropped, $\mathcal{A}$ is restricted to a one-time game. Thus to win against matching conversations, $\mathcal{A}$ must win by forging the OTMAC, which can be done by either forging the message or the tag, i.e. breaking sEUF-1-CMA security. Additionally, we account for adjusted length on the OTMAC function, $l = 32$, and time duration parameter, $t_d = 4sec$. Therefore we have,

$$\text{Adv}^3 \leq \text{Adv}^4 + \text{Adv}^{sEUF-1-CMA}_{\mathcal{A}_2,OTMAC,l=32,t_d=4sec}(\lambda) \ .$$

Case 2. In this case the first oracle to accept maliciously has role "initiator". The step follows similarly to Case 1. Role distinction is implied in the computation of the OTMAC key.

Thus, we have that

$$\text{Adv}^3 \leq \text{Adv}^4 + 2 \cdot \text{Adv}^{sEUF-1-CMA}_{\mathcal{A}_2,OTMAC,l=32,t_d=4sec}(\lambda) \ .$$

(Indistinguishability of the keys.) If the adversary successfully distinguishes the session key from a random value (b from b') it wins. For session key for Dialect 1 is $K^{D1} = K^{D1^*}_{SC} \| K^{D1^*}_{CS}$, where $K^{D1^*}_{SC}$ (resp. $K^{D1^*}_{CS}$) is the output of the KDF in Step 3a and 3b of Figure 4.1. Thus distinguishing K^{D1} from random implies distinguishing either Thus,

$$\text{Adv}^4 \leq 2 \cdot \text{Adv}^{\text{PRNG}}_{KDF,\mathcal{A}_3}(\lambda) \ .$$

□

In Sjoholmsierchio et. al [2] implementation, the time-window, i.e., time duration parameter, that was used was really small -only one second- and not four seconds like we chose. Their choice though was made mainly for simplicity reasons and is something that can be easily changed (one place in the programming code). Changing the time-window value and

experimenting with the implementation can give us various insights according to different results we would receive. We could look at things like performance or compare numbers of false negatives, i.e., messages that should pass validation but were rejected instead.

No matter the variant of a MAC algorithm, it usually inherits efficiency and security from its parents' primitives [34]. The HMAC algorithm in our protocol uses the cryptographic hash function BLAKE2b, based on the selection made in prior work [3]. This has been shown to be efficient, thus making the HMAC relatively fast.

The factor of q_T^2 appears as a notable reduction in tightness of the proof. However, unlike a normal sEUF-CMA MAC security where the adversary may have an equivalent number of queries against the MAC under a given key, we instead restrict the adversary to one query per MAC under a given key. Consequently a polynomial number of queries against the MAC is not surprising but rather distributed across many different instantiations of the algorithm (i.e., run under different keys).

The proof demonstrates areas for optimization in the dialect design. For example, the use of two KDF computations for key derivation adds a factor. If one KDF was used and split into sending/receiving keys, this could be optimized.

The use of the Reveal queries points to reveal of the full state. If pre-shared keys / per time interval keys were stored more securely than session keys, then separating modeling of compromise of the session key and other keys could provide additional security insights.

In this chapter we presented the security model and the security proof. In the next chapter we write our conclusions and some ideas for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 6:
Conclusion and Future Work

In this chapter we present conclusions of this thesis and propose some ideas for future work.

6.1 Conclusion

In this research we have shown that the dialected Openflow protocol proposed and designed in [3], [2] provides a viable option for improving security for some of the existing issues in the original Openflow. In our analysis, we assume that D1 and D2 are performed sequentially with D2 following the implementation of D1. Another assumption that we make is that keys are used only once for a certain Controller/Switch pair.

Based on key derivation, a new shared OTMAC key is derived every 2 seconds. $\mathcal{A}$ must forge a message-tag pair within the given time window. The one-time MAC is making that a hard challenge for $\mathcal{A}$. Not only does it have a small probability because of new keys being derived in small time-windows but also it can only try on each time-window only once.

The design of the dialect seems supported by our proof which works by providing abilities to a strong $adversary$ on each step and reducing the chances for breaking the protocol. This is due to mutual authentication taking place and key establishment before any actual exchange of data take place. The use of key derivation functions and one-time MACs provide additional security with or without TLS being enabled.

Our goal is to analyze security in the dialected Openflow at a high level, thus we do not take any issues on implementation into account. A more in-depth analysis can consider issues like the time aliveness of the keys before they are “killed off" or a closer assessment of the time intervals for OTMAC verification functions. Another important outcome of future analysis can be a different dialect solution but as we have shown, there are limitations because of the nature of the protocol. The 32-bit header field in Openflow limits our dialect option if the protocol packet appearance must be (or is desired to be) preserved.

Even if there are still some open questions in the security analysis of the Openflow protocol, we consider this dialect solution and our results as a strong indicator for the viability of the dialected Openflow protocol.

6.2 Future Work

The dialect can be modified and tried with other choices than the ones we made. For example different time interval for the "killing off" of keys, or different time (w) for acceptance as discussed in 4.2.3.

It is still unknown, what is the possibility of success of an adversary trying to break the MAC scheme with an adaptive chosen-message attack [35]. With the restriction of the length parameter of 32 bits for the MAC, the time duration parameter of 4 seconds and the one-time only scheme seem appropriate choices in our effort in trying to keep the unforgeability of the MAC. Future research can further, experiment with different values for the parameter, or even a different deconfliction algorithm.

Further analysis of the protocol, even a different approach of provable security is possible, since even if our analysis, answers and indicates important security concerns, there are still open questions that can be answered from a different perspective.

List of References

[1] "Software-defined networking: The new norm for networks - Open networking foundation," Apr. 2012 [Online]. Available: https://opennetworking.org/sdn-resources/whitepapers/software-defined-networking-the-new-norm-for-networks/

[2] M. Sjoholmsierchio, B. Hale, D. Lukaszewski, and G. Xie, "Strengthening SDN security: Protocol dialecting and downgrade attacks," *Proceedings of the 2021 IEEE Conference on Network Softwarization: Accelerating Network Softwarization in the Cognitive Age, NetSoft 2021*, pp. 321–329, 2021.

[3] M. Sjoholmsierchio, "Software-defined networks: Protocol dialects," M.S. thesis, Dept. of Information Sciences, NPS, Monterey, CA, USA, 2019 [Online]. Available: https://calhoun.nps.edu/handle/10945/64066

[4] 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. 4–6, 2014.

[5] S. Barros, M. Simplicio, T. Carvalho, M. Rojas, F. Redigolo, E. Andrade, and D. Magri, "Applying software-defined networks to cloud computing," in *33rd Brazil-ian Symposium on Computer Networks and Distributed Systems*, 2015.

[6] R. Enns, "Netconf configuration protocol," RFC 4741, Dec. 2006 [Online]. Available: https://www.rfc-editor.org/info/rfc4741

[7] Open Networking Foundation, "OpenFlow management and configuration pro-tocol (OF-Config 1.1.1)," pp. 1–36, March 23, 2013 [Online]. Available: http://opennetworking.wpengine.com/wp-content/uploads/2013/02/of-config-1-1-1.pdf

[8] B. Pfaff and B. Davie, "The open vswitch database management protocol," RFC 7047, Dec. 2013 [Online]. Available: https://www.rfc-editor.org/info/rfc7047

[9] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[10] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "Elastictree: Saving energy in data center networks." in *NSDI*, 2010, vol. 10, pp. 249–264.

[11] S. Kent, C. Lynn, and K. Seo, "Secure border gateway protocol (S-BGP)," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 4, pp. 582–592, 2000.

[12] Fiber Optic Solutions, "Backbone networks analysis," Jul. 1, 2016 [Online]. Available: https://www.fiber-optic-solutions.com/analysis-backbone-networks.html

[13] A. Sonba and H. Abdalkreim, "Performance comparison of the state of the art openflow controllers," M.S. thesis, Dept. of Information Technology, Halmstad University, Halmstad, Sweden, 2014 [Online]. Available: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A783679&dswid=-7750

[14] Wikipedia contributors, "Firmware – Wikipedia, the free encyclopedia," 2022 [Online]. Available: https://en.wikipedia.org/w/index.php?title=Firmware&oldid=1071875365

[15] M. Jammal, T. Singh, A. Shami, R. Asal, and Y. Li, "Software defined networking: State of the art and research challenges," *Computer Networks*, vol. 72, pp. 74–98, 2014.

[16] CISA, "Understanding denial-of-service attacks," 2019 [Online]. Available: https://www.cisa.gov/uscert/ncas/tips/ST04-015

[17] NIST, "Man in the middle attack," 2022 [Online]. Available: https://csrc.nist.gov/glossary/term/man_in_the_middle_attack

[18] Open Networking Foundation, "Openflow switch specification (version 1.5.1)," *Current*, vol. 0, pp. 1–36, 2015 [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf

[19] R. Klöti, V. Kotronis, and P. Smith, "Openflow: A security analysis," in *2013 21st IEEE International Conference on Network Protocols (ICNP)*, 2013, pp. 1–6.

[20] C. Boyd, A. Mathuria, and D. Stebila, *Protocols for Authentication and Key Establishment*, 2nd ed. Heidelberger Platz 3, 14197 Berlin, Germany: Springer, 2020.

[21] Y. Sung, P. K. Sharma, E. M. Lopez, and J. H. Park, "FS-opensecurity: A taxonomic modeling of security threats in SDN for future sustainable computing," *Sustainability*, vol. 8, no. 9, p. 919, 2016.

[22] B. Hale and C. Boyd, "Computationally analyzing the iso 9798–2.4 authentication protocol," *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8893, pp. 236–255, 12 2014. Available: https://link.springer.com/chapter/10.1007/978-3-319-14054-4_14

[23] J. S. Coron and J. B. Nielsen, "0-RTT key exchange with full forward secrecy," *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10211 LNCS, no. Eurocrypt, pp. V–VI, 2017.

[24] C. Adams, G. Kramer, S. Mister, and R. Zuccherato, "On the security of key derivation functions," *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3225, pp. 134–145, 2004.

[25] D. Stebila, "An introduction to provable security," *Lecture Notes in AMSI Winter School on Cryptography*, pp. 1–15, July 10-11, 2014 [Online]. Available: http://files.douglas.stebila.ca/files/teaching/amsi-winter-school/Lecture-2-3-Provable-security.pdf

[26] M. Bellare and P. Rogaway, "Entity authentication and key distribution," *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 773 LNCS, pp. 232–249, 1994.

[27] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk, "On the security of TLS-DHE in the standard model," *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7417 LNCS, pp. 273–293, 2012.

[28] D. Basin, C. Cremers, and S. Meier, "Provably repairing the ISO/IEC 9798 standard for entity authentication," *Journal of Computer Security*, vol. 21, no. 6, pp. 817–846, 2013.

[29] M. Barbosa and P. Farshim, "Security analysis of standard authentication and key agreement protocols utilising timestamps," *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5580 LNCS, pp. 235–253, 2009 [Online]. Available: https://link-springer-com.libproxy.nps.edu/chapter/10.1007/978-3-642-02384-2_15

[30] H. Massias, X. S. Avila, and J.-J. Quisquater, "Timestamps: Main issues on their use and implementation," *Proceedings. IEEE 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'99)*, pp. 178–178, June 1999.

[31] A. Buldas and M. Saarepera, "On provably secure time-stamping schemes," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2004, pp. 500–514.

[32] B. Van Rompay, B. Preneel, and J. Vandewalle, "The digital timestamping problem," in *Symposium on Information Theory in the Benelux*. Werkgemeenschap voor Informatie-en Communicatietheorie, 1999, pp. 71–78.

[33] L. Gong, "A security risk of depending on synchronized clocks," *ACM SIGOPS Operating Systems Review*, vol. 26, no. 1, pp. 49–53, 1992.

[34] J. Kim, A. Biryukov, B. Preneel, and S. Hong, "On the security of HMAC and NMAC based on HAVAL, MD4, MD5, SHA-0 and SHA-1," in *International Conference on Security and Cryptography for Networks*. Springer, 2006, pp. 242–256.

[35] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal on Computing*, vol. 17, no. 2, pp. 281–308, 1988.

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California