UNIVERSITÉ DU LUXEMBOURG

PhD-FSTM-2020-52
The Faculty of Sciences, Technology and Medicine

DISSERTATION

Defence held on 15/09/2020 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Diego KREUTZ
Born on 29 November 1979 in Santa Rosa (Brazil)

LOGICALLY CENTRALIZED SECURITY FOR SOFTWARE-DEFINED NETWORKING

Acknowledgements

I am very proud of and grateful to my supervisors Paulo Esteves-Veríssimo and Fernando M. V. Ramos because I have learned so much with them. One of the few things I am sure of is my eternal debt with them. I am (arguably) sure I am a better researcher now because of them. In fact, I am still learning everyday something new with them.

Paulo was also an awesome sort of coach in my personal life as well. He is an impressively smart and knowledgeable person. I have to say, in some ways, he is indeed better than a regular doctor. I know that (probably) just me and him will truly understand the meaning of this statement.

I had one of the most interesting and productive times after I met Jiangshan Yu. He is a very smart, easy going and inspiring researcher and person. Because of him, I started to better understand and to have joy in designing security protocols. JY inspired me in different ways. I consider myself lucky to have met him.

I cannot forget to mention Natalie Kirf, Jessica Giro and the BED in general. Jessica Giro was of outstanding administrative support inside the University of Luxembourg. Jessica managed to figure out and propose simple and straightforward solutions to every single hurdle during my affiliation with the University. I must say I am really lucky to have met such sympathetic, positive and professional people like Jessica, always ready to help us in the best way possible.

I have no words to thank my family, friends and god for the constant support and comfort. I have to gratefully thank my wife (Rosa Maria) and my mother (Elizabeta Ilzi). Their unbelievably strong faith (and energy) made me arguably stronger at the last minute, when I really needed it. Truth to be said, family and friends are two of the most important things in my life. During the past few years, both my family and friends were of utter most importance to get through the different health issues I had. I got through all of it because of them, no doubt.

Finally, I feel the need to mention some friends of mine, that helped me in different ways through out this journey, namely Vinicius Vielmo Cogo, Laly Ortiz, Pedro Costa, Wojciech Agzo, Alberto Mengali, Danilo Spano, Grabiela Gregory, Sina Maleki, Divane Marcon, and Jader Franklin (a.k.a., Deko). Their warmth friendship made me a better person and provided me joy and comfort in several moments of my journey.

Declaration

I, Diego Kreutz, declare that this thesis "Logically centralized security for Software-Defined Networking" and the work presented therein are my own. I confirm that:

- this work was done wholly or mainly while in candidature for the degree Docteur de l'Université du Luxembourg;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this university or any other institution, this has been clearly stated;
- where I have consulted the published works of others, these are clearly attributed;
- where I have quoted from the works of others, the sources are always given;
- where the work presented in this thesis is based on work done by myself jointly with others, I have clearly outlined what was done by others and what I contributed;
- with the exception of such quotations, this is entirely my own work; and
- I have acknowledged all main sources of help.

Signed: ______________________________

Date: ______________________________

Abstract

Software-Defined Networking (SDN) decouples the control and data planes of traditional networks, logically centralizing the functional properties of the network in the SDN controller. While this centralization brought advantages such as a faster pace of innovation, it also disrupted some of the natural defenses of traditional architectures against different threats. Until now, SDN research has essentially been concerned with the functional side, despite some specific works relating to non-functional properties like 'security', 'dependability', or 'quality of service'.

Security is an essential non-functional property of SDN. The lack of reliable security by-design mechanisms can quickly lead to the compromise of the entire network. For instance, most of the current security mechanisms in SDN controllers lead to exploitable vulnerabilities that allow adversaries to easily control or even shut down the entire control plane. The growing concern regarding insider threats substantially amplifies the problem. The reason lies in the fact that current Software-Defined Networks (SDNs) (e.g., OpenFlow-enabled networks) rely on weak protection mechanisms. To address these crucial security issues in the SDN control plane, it is necessary, though not sufficient, that we start by securely identifying, authenticating, and authorizing all devices before allowing them to become part of the network.

Though SDN security is the central tenet of this thesis, we believe that the problem is much more generic. In essence, there is still a lack of a systematic approach to ensuring such relevant non-functional properties as security, dependability, or quality of service. Current approaches are mostly ad-hoc and piecemeal, which has led to efficiency and effectiveness problems. This reflection led us to claim that the successful enforcement of non-functional properties as a pillar of SDN robustness calls for a systematic approach. We further advocate, for its materialization, the re-iteration of the successful formula behind SDN– 'logical centralization'.

In consequence, we propose ANCHOR, a subsystem architecture for SDN that promotes the logical centralization of non-functional properties. We start by presenting the general concept and architectural principles, suggesting how they can satisfactorily enhance the current state of the art with regard to any non-functional property (security, dependability, performance, quality of service, etc.). We claim and justify that centralizing such mechanisms is vital for their effectiveness, by allowing us to: define and enforce global policies for those properties; reduce the complexity of controllers and forwarding devices; ensure higher levels of robustness for critical services; foster interoperability of the non-functional property enforcement mechanisms; and finally, better foster the resilience of the architecture itself. We focus on 'security' as a use case in the rest of the thesis, discussing the specialization of the ANCHOR architecture to logically-centralized enforcement of security properties. However, by presenting a principled solution to the main problem of the thesis (SDN security), we also show the effectiveness of the general ANCHOR concept, opening avenues for further research on its extension to other desirable non-functional properties, such as dependability and Quality of Service (QoS).

We identify the current security gaps in SDNs, and investigate the adequate security mechanisms that should populate the architecture middleware, globally and consistently. ANCHOR sets out to provide — in a homogeneous manner to all controllers and forwarding devices — essential security mechanisms such as strong entropy, resilient pseudo-random generators, secure device registration, association and recommendation, amongst other crucial services. We present the design of those mechanisms and protocols. With the objective of promoting generalized use of encryption and authentication in the control plane, we additionally propose and describe a secure control plane communication infrastructure, Keep It Simple and Secure (KISS), based on a novel lightweight mechanism for generating cryptographic secrets — integrated Device Verification Value (iDVV). iDVV can be used in a number of ways, in a number of protocols, and outperforms widely used alternatives. In the context of this thesis, the KISS infrastructure is set up by ANCHOR and used to ensure the security of interactions amongst it, controllers and forwarding devices.

Being conceptually logically-centralized, ANCHOR presents a single-point-of-failure (SPoF) challenge, which we address, through incremental measures, some of which can be selectively present in concrete designs. As a baseline, we harden the design, by endowing it with robust functions in the different modules. We increase assurance by discussing and informally proving correctness of all mechanisms and algorithms, and we also formally verify the main algorithms through a proof-assistant. By only using symmetric cryptography, we make the system Post-Quantum Secure (PQS). We also embed measures to achieve Perfect Forward Secrecy (PFS) in all algorithms, protecting pre-compromise communications in the presence of successful attacks. Finally, for higher criticality systems, we take additional algorithmic and architectural measures to mitigate the effects of possible security failures. We provide for Post-Compromise Security (PCS) through the semi-automatic restart of operation after a full compromise of ANCHOR. We present as well a design of resilience mechanisms — the continued prevention of failure/compromise by automatic means — through fail-fast recovery techniques.

The prototypes' implementation aspects and the evaluation of the two fundamental pieces of our work (ANCHOR and KISS) are performed in the respective chapters. The above-mentioned discussion and informal proof of correctness of all mechanisms and algorithms is given in appendices. We also formally machine-verified the main algorithms.

Contents

List of Figures

List of Tables

Abbreviations

AAA Authentication, Authorization, and Accounting

AAIs Authentication and Authorization Infrastructures

AES Advanced Encryption Standard

API Application Programming Interface

ASIC application-specific integrated circuit

AuS Automatic Security

BFT Byzantine Fault Tolerance

BYOD Bring Your Own Device

CAPEX Capital Expenditure

CIA Confidentiality, Integrity, Availability

CLI command-line interface

COTS commercial off-the-shelf

CPU Central Processing Unit

DDoS Distributed Denial-of-Service

DH Diffie-Hellman

DHE Diffie-Hellman key Exchange

DIMM Dual In-Line Memory Module

DNS Domain Name System

DoS Denial-of-Service

DPID OpenFlow Datapath ID

EAP Extensible Authentication Protocol

ECDHE Elliptic Curve Diffie-Hellman key Exchange

FIFO First In, First Out

ForCES	Forwarding and Control Element Separation
HLHA	Host Location Hijacking Attack
HMAC	Hash-based Message Authentication Code
HSM	Hardware Security Module
HTTPS	Hypertext Transfer Protocol Secure
iCVV	integrated Card Verification Values
ID	IDentification
IDS	Intrusion Detection System
iDVV	integrated Device Verification Value
IoT	Internet of Things
IP	Internet Protocol
IPS	Intrusion Prevention System
IS-IS	Intermediate System-to-Intermediate System
IT	Information Technology
ITM	Insider Threat Mitigation
KDC	Key Distribution Center
KDF	Key Derivation Function
KDS	Key Distribution Service
KISS	Keep It Simple and Secure
KYE	Know Your Enemy
LAN	Local Area Network
LANs	Local Area Networks
LOC	Lines Of Code
LTS	Long Term Support

MAC Media Access Control

MEF MEF

MITM Man-in-the-Middle

MAC Message Authentication Code

NaCl Networking and Cryptography library

NIDS Network Intrusion Detection System

NIST National Institute of Standards and Technology

NOS Network Operating System

NS Needham-Schroeder

NSA National Security Agency

NTP Network Time Protocol

ODL OpenDaylight

ODTN Open and Disaggregated Transport Network

ONF Open Networking Foundation

ONL Open Network Linux

ONOS Open Network Operating System

OPEX Operational Expenditure

OSPF Open Shortest Path First

OVS Open vSwitch

PC Personal Computer

PCR Post-Compromise Recovery

PCS Post-Compromise Security

P4 Programming Protocol-independent Packet Processors

PFS Perfect Forward Secrecy

PKI	Public Key Infrastructure
POF	Protocol Oblivious Forwarding
PRF	Pseudo Random Function
PRG	Pseudo Random Generator
QoS	Quality of Service
RADIUS	Remote Authentication Dial-In User Service
ReS	Resilient Services
RNG	Random Number Generator
SC	Secure Component
SDN	Software-Defined Networking
SDNs	Software-Defined Networks
SNBI	Secure Network Bootstrapping Infrastructure
SNMP	Simple Network Management Protocol
SPoF	single-point-of-failure
SSH	Secure Shell
SSL	Secure Sockets Layer
STS	Statistical Test Suite
TCB	Trusted Computing Base
TCP	Transport Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
USB	Universal Serial Bus
VANETs	Vehicular Ad Hoc Networks
VoIP	Voice over Internet Protocol
WAN	Wide Area Network

Chapter 1

Introduction

In traditional computer networks, we have issues, such as vendor lock-in and heterogeneous devices with low interoperability, which lead to higher costs in terms of Capital Expenditure (CAPEX) and Operational Expenditure (OPEX). Typically, network operators try to enforce global policies by configuring networking devices manually or with the use of device-specific scripts. And it gets challenging to keep policies in place while networks are semi-autonomously reacting to faults and load changes. Additionally, current networks are vertically integrated, which means that both control and data planes are deployed inside the networking devices. This is one of the hardest obstacles to innovation in networking infrastructures.

SDN is a paradigm that is causing a major change in the networking industry. SDN has set a new pace of innovation by breaking the vertical integration of traditional networks. Data plane devices (e.g., switches, routers) become simple forwarding devices while the control logic is implemented outside, in a logically centralized controller, or Network Operating System (NOS), simplifying policy enforcement. Running on top of a NOS, network applications (e.g., routing protocol) have access to a global network view. From a developer perspective, it becomes easier to think about network behavior, leading to a new pace of innovation in networks. An overview of this decoupling of control and data planes is shown in Figure 1.1.

1.1 Problem statement

SDN research and development has been primarily concerned with functional properties, such as improved routing and traffic engineering to avoid over-provisioning, increase the resilience of the network data plane, provide fast recovery in case of failures, reduce energy consumption, efficient load balancing, and reduce OPEX and CAPEX costs [Wan+18]; [ZFC17]; [Hel+10]; [JYR11]; [Jai+13]; [RME13];

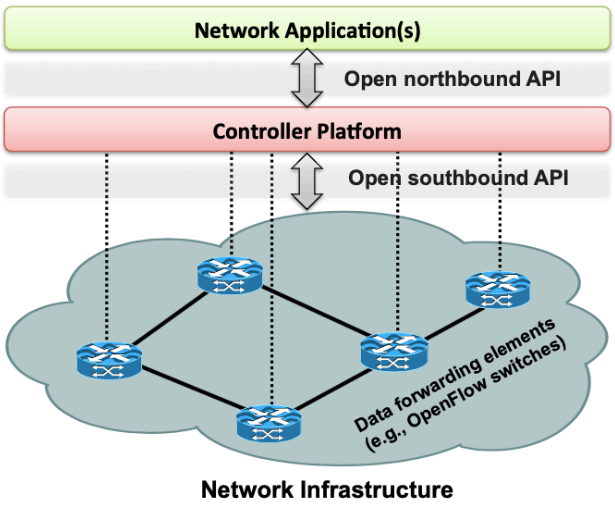

Figure 1.1: SDN overview.

[Wan+15]; [Hua+16]; [Jin+16]; [Chi+16]; [Men+17]; [Alv+17]; [Yan+15]; [NDK16]; [PST18]; [JCL19]; [MR19]; [AMA19].

While gaps in the enforcement of non-functional properties (e.g., security, dependability, QoS) are critical to the deployment and widespread use of SDN in the enterprise [Kre+15]; [Por+15]; [Arb+16]; [TZN16]; [Sco17]; [NK19]; [IK18]; [Han+19], they have received less attention. For instance: insecure control plane associations and communications, network information disclosure, spoofing attacks, Denial-of-Service (DoS) attacks on control and data planes, Transport Layer Security (TLS) Man-in-the-Middle (MITM) attacks on control plane communications, teleportation attacks, and hijacking of devices, can easily compromise the network operation; performance crises can escalate to affect QoS globally; unavailability and lack of reliability of controllers, forwarding devices, or clock synchronization parameters, can considerably degrade or even compromise the network operation [KKS13]; [Akh+15]; [TSS17]; [Yoo+17a]; [SNS16]; [Dac+17]; [Akh+16]; [RR17]; [Jer+17b]; [AvW18]; [IK18]; [Han+19].

One of the most vital components of the SDN architecture is the communication between forwarding devices and controllers. For instance, fake forwarding devices can easily connect to the controller and wreak havoc with the network. A fake forwarding device can use teleportation to exploit key SDN functionalities such as flow (re-)configuration, switch identification, and out-of-band forwarding [TSS17]. With teleportation, malicious forwarding devices can exploit the logically centralized control plane to transport information, bypassing data plane devices (e.g., other forwarding devices, middleboxes, firewalls, Intrusion Prevention System (IPS)). Similarly, fake controllers can also compromise and disrupt the network operation.

One fundamental reason for these problems is that, in spite of the benefits of SDN, the decoupling of control and data planes, relying on a de facto standard southbound API, namely OpenFlow, has removed crucial natural protections of traditional networks, such as the heterogeneity of management solutions, the diversity of configuration protocols and operations, and the diversity of security and fault-tolerance mechanisms. For instance, on a traditional network, an attacker would need to (arguably) compromise multiple protocols and different security mechanisms of rather heterogeneous forwarding devices. Likewise, cascading faults and performance crises would be mitigated by natural fault independence and diversity across different architectures.

Each specific and proprietary configuration protocol, from traditional manufacturers such as Cisco, HP, D-Link, and Huawei, has its own set of operations, i.e., instructions to change the configuration of the specific device. Hence from e.g., a security perspective, SDN introduces new threat vectors, changing the threat surface of networks [BCS13]; [KKS13]; [Kre+15]; [SNS16]; [Dac+17]; [Akh+16]; [AvW18]; [Han+19].

Not surprisingly, security has been recurrently pointed out as one of the major open issues in all sorts of SDN deployments, such as Local Area Networks (LANs), optical networks, Open and Disaggregated Transport Network (ODTN), Internet of Things (IoT), mobile networks, transport networks, ad-hoc networks, Vehicular Ad Hoc Networks (VANETs), and industrial networks [Bas+13]; [Ku+14]; [Yan+15]; [Kre+15]; [Thy+16]; [SNS16]; [Che+17]; [BMV17]; [NDK16]; [Han+19]; [Gio+20]. Whilst some authors consider security and dependability as necessary inherent features of the architecture of future networks and systems [PPJ11]; [Car17]; [Han+19], the fact is that most existing projects still do not consider security by design, i.e., leave security as an afterthought [Dac+17]; [Pas14]; [Sco17]; [Dac+17]; [Han+19].

Most of the available SDN controllers, including well-known ones such as Open Network Operating System (ONOS) and OpenDaylight (ODL), have several vulnerabilities such as malformed control messages, handshake without hello message, packet-in flooding, control message drop, infinite loop, system variable manipulation, system command execution, eavesdrop, application eviction, and state manipulation and man-in-the-middle attacks [Lee+17]; [Sec+17]; [Noh+16]; [Yoo+17a]; [Xu+17]; [Sco17]; [Guo+16]; [Xu+17]; [Dix+18]; [Han+19]. Some of these vulnerabilities can severely compromise the network operations, as studied in detail in Chapter 2.

To address these challenges we need to answer the following questions: *How to securely deploy and install devices (registration, authentication, authorization, and* *association) in network infrastructures? How to ensure the dependable and secure operation of controllers and forwarding devices, in spite of faults and attacks?*

1.2 Proposal

Though SDN security is the central tenet of this thesis, we believe the problem is more generic. In essence, there is still lack of a systematic approach to ensuring such relevant non-functional properties as security, dependability, or quality of service. Current approaches have been addressing these challenges in an ad-hoc, piecemeal way, which may work, but will inevitably lead to efficiency and effectiveness problems. Several specific works concerning non-functional properties have recently seen the light, e.g., in dependability [Son+17]; [Kre+15]; [Ber+14]; [Akh+16]; [Men+18]; [Viz+18]; [Viz18]; [Can+18]; [NK19] or security [Shi+13a]; [Shi+13c]; [Shi+14]; [Amb+15]; [SNS16]; [Akh+16]; [Cho+17b]; [RR17]; [Sha+17]; [Yoo+17b]; [AvW18]; [Han+19]. However, these are point solutions, and a panoply of different vulnerabilities and attacks (e.g., application eviction, lack of strong and global authentication and authorization services to access controllers, lack of mechanisms for device identification and authentication, complexity, vulnerabilities and misconfigurations of TLS implementations, Distributed Denial-of-Service (DDoS) among controllers, TLS MITM attacks on control plane communications), and Internet-scale attacks (e.g., more stealthy and sophisticated low-bandwidth DDoS attacks [Hic17]; [Inf13]), still represent challenges in SDN infrastructures [Che+16]; [SNS16]; [Yan+16]; [MK17]; [Dac+17]; [PP17]; [Jer+17b]; [Thi+18]; [Dix+18]; [AvW18]; [Han+19].

This reflection led us to claim that the successful enforcement of non-functional properties as a pillar of SDN robustness calls for a systematic approach. An opinion seconded by recent research [KSM13]; [MDP15]; [SNS16]; [Han+19]. We further advocate, for its materialization, the *re-iteration of the successful formula behind SDN– 'logical centralization'*. It is worth emphasizing that the centralization of security services has been proposed a few times to solve different problems of traditional and SDN networks. For instance, the use of centralized cryptography schemes and sources of trust to authenticate and authorize known entities have been pointed out as a solution for improving the security of Ethernet networks [KSM13]. Similarly, recent research suggests network security as a service to provide the protocols and mechanisms required for enterprise-grade SDN networks [SNS16]. However, centralization has its drawbacks, not least creating bottlenecks and single points of failure.

In consequence, to reap the benefits of centralization and avoid its pitfalls, we propose ANCHOR, a subsystem architecture for SDN that promotes the logical centralization of non-functional properties, by standing aside the functional SDN architecture with its payload controllers and forwarding devices, *not modifying but rather adding to it.* It 'anchors' crucial functionality and properties, and 'hooks' to the former components, in order to secure the desired properties. We claim and justify that centralizing such mechanisms is vital for their effectiveness, by allowing us (with minimal interference with the payload SDN architecture) to: (1) define and enforce global policies for those properties; (2) reduce the complexity of controllers and forwarding devices; (3) ensure higher levels of robustness for critical services; (4) foster interoperability of the non-functional property enforcement mechanisms; and finally, (5) better foster the resilience of the architecture itself. We conjecture that the general concept and architectural principles can satisfactorily enhance the current state of the art with regard to any non-functional property (security, dependability, performance, quality of service, etc.).

Focusing on 'security' as a use case, we will thoroughly demonstrate and validate our proposal, discussing the specialization of the ANCHOR architecture to logically-centralized enforcement of security properties. By presenting a principled solution to the main problem of the thesis (SDN security), we also show the effectiveness of the general ANCHOR concept, opening avenues for further research on its extension to other desirable non-functional properties.

We propose to identify the current security gaps in SDN, and investigate the adequate security mechanisms that should populate the architecture middleware with the appropriate security mechanisms, globally and consistently. ANCHOR sets out to provide these essential security mechanisms, in a homogeneous manner to all controllers and forwarding devices.

With the objective of promoting generalized use of encryption and authentication in the control plane, we additionally propose a secure control plane infrastructure, KISS, which intends to bring simplicity and performance to the provision of integrity, authenticity, and confidentiality of control plane communications, outperforming widely used alternatives. The KISS infrastructure, set up by ANCHOR, is crucial to ensure the security of interactions amongst it, controllers and devices.

ANCHOR being conceptually centralized, even if 'logically' for a start, presents a SPoF challenge. We propose to address the challenge through incremental measures, some of which can be selectively present in concrete designs. As a baseline, we propose to harden the design, by endowing it with robust functions in the different modules, incorporate the capacity of post-quantum security (PQS) by only using symmetric cryptography, and also embed measures to achieve perfect forward secrecy (PFS) in all algorithms. Finally, for higher criticality systems, we propose to take additional stronger algorithmic and architectural measures to mitigate or prevent the effects of possible security failures. We will provide for PCS through the semi-automatic restart of operation in the event of a full compromise of ANCHOR. We will investigate as well resilience mechanisms — the continued prevention of failure/compromise by automatic means — through fail-fast recovery techniques.

Besides attempting at proving the correctness of mechanisms and algorithms, we will design all the protocols, and evaluate proof-of-concept prototype implementations.

1.3 Contributions

In short, the main contributions of our work include the following:

1. The principle of using logical centralization for the successful enforcement of generic non-functional properties in SDN, materialized through the blueprint of the ANCHOR architectural concept.

2. The first comprehensive study of the threat plane impending on SDN systems, identifying and organizing a systematics of threat vectors and agents, easing the study of solutions improving security.

3. The definition and design and implementation of a specialization of the ANCHOR architecture to logically-centralized security, providing mechanisms and protocols for essential security services, such as, but not only, strong entropy, resilient pseudo-random generators, secure device registration, association and controller recommendation.

4. The definition and design and implementation of an infrastructure, KISS, bringing simplicity and performance to the provision of security integrity, authenticity, and confidentiality of control plane communications, introducing cryptographic mechanisms outperforming widely used alternatives.

5. An approach to the systematic mitigation of the risk of logical centralization, from baseline mechanisms — design hardening with robust functions, symmetric cryptography for post-quantum security, formal verification for increased assurance, embedding of perfect forward secrecy measures in all algorithms, manual post-compromise security — to additional mechanisms in the path of resilience, materialized by an extension of the architecture called R-ANCHOR— mitigation of insider threats, minimization of the syndrome of SPoF of the management site, by automatic recovery for fast turnaround and post-compromise security after full compromise or failure.

We show that, compared to the state-of-the-art in SDN security, our solution preserves at least the same security functionality, but achieves higher levels of implementation robustness, by vulnerability reduction, while providing high performance. While we attempting to prove our point with security, our contribution is generic enough to inspire further research concerning other non-functional properties (such as dependability or quality-of-service). It is also worth emphasizing that the architectural concept that we propose in this work would require a more significant deployment effort in traditional networks, due to the heterogeneity of the infrastructure and its vertical integration.

The work presented in this thesis was independently validated through peer-reviewed publications in international conferences and journals of this area. The list of publications related to this research and chapters they refer to is:

- Diego Kreutz, Fernando M.V. Ramos, and Paulo Verissimo. "Towards secure and dependable software-defined networks". In: *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. HotSDN '13. Hong Kong, China: ACM, 2013, pp. 55–60. doi: `10.1145/2491185.2491199`.
  Identification of the global threat plane of SDN, showing the pressing security problems arising (Chapters 1 and 2).

- Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. "Software-defined networking: A comprehensive survey". In: *Proceedings of the IEEE* 103.1 (2015), pp. 14–76.
  Comprehensive survey allowing a deep knowledge of the SDN problem space and guiding towards the inception of the logical-centralisation idea (Chapter 2 and 3).

- D. Kreutz, J. Yu, P. Esteves-Veríssimo, C. Magalhães, and F. M. V. Ramos. "The KISS Principle in Software-Defined Networking: A Framework for Secure Communications". In: *IEEE Security Privacy* 16.5 (Sept. 2018), pp. 60–70. doi: `10.1109/MSP.2018.3761717`. An extended version available online at `http://arxiv.org/abs/1702.04294`.
  Design, implementation and evaluation of the KISS infrastructure (Chapter 4).

- Diego Kreutz, Jiangshan Yu, Fernando M. V. Ramos, and Paulo Esteves-Verissimo. "ANCHOR: Logically Centralized Security for Software- Defined Networks". In: *ACM Transactions on Privacy and Security* 22.2 (Feb. 2019), 8:1–8:36. doi: `10.1145/3301305`. An extended version available online at `https://arxiv.org/abs/1711.03636`.
  Design, implementation and evaluation of the ANCHOR architecture (Chapters 3, 5 and 6).

1.4 Structure of the Thesis

The thesis is organized as follows.

Chapter 1 — Introduction of the problem and outline of our approach to the solution, logically-centralization of non-functional properties.

Chapter 2 — Review and analysis of landscape of security in SDN, systematizing the overall threat plane impending on it. Subsequent analyses of the state of the art with regard to solutions to security problems in SDN.

Chapter 3 — Delineates our approach in detail. First, we present the merits of an approach based on a logically centralized root of trust. Second, we investigate the specialization to security, by identifying the current security gaps in SDNs. Third, we propose the adequate security mechanisms that should populate the architecture middleware.

Chapter 4 — Presents the design, implementation and evaluation of mechanisms and protocols for a secure control plane communication infrastructure, KISS, based on a novel lightweight mechanism for generating cryptographic secrets (iDVV).

Chapter 5 — Presents the design, implementation and evaluation of mechanisms and protocols for ANCHOR, including essential security mechanisms such as strong entropy, resilient pseudo-random generators, secure device registration, association and recommendation.

Chapter 6 — Presents our approach to the systematic mitigation of the risk of logical centralization. In this chapter, we recapitulate the prevention and assurance mechanisms promoting robustness, already incorporated in the baseline designs of ANCHOR and KISS described in the previous chapters. We complement those by presenting an optional extension of the architecture, R-ANCHOR— for, e.g., more critical applications — with the design of mechanisms promoting resilience, through mitigation of insider threats, as well as automatic recovery for fast turnaround and post-compromise security after full compromise or failure.

Chapter 2

Background and Related Work

In this chapter, we introduce software-defined networking (Section 2.1), its threat plane ((Section 2.2), security requirements (Section 2.3), and related work on securing control plane communications (Section 2.4). The main idea of SDN is the decoupling of the control and data planes of networking devices, introducing new levels of flexibility by allowing simpler and more productive network programming paradigms.

After the initial insights about SDN, we address the central tenet of this thesis, the security problems of SDN. We present a contribution of this thesis, a comprehensive study of the threat plane impending on SDN systems, identifying and organizing systematics of threat vectors and agents, easing the study of solutions improving security. We introduce the main threat vectors and specific security issues of SDN.

2.1 Software-Defined Networking

The term SDN was initially coined to represent the ideas and work around OpenFlow at Stanford University [Gre09]. As initially defined, SDN refers to a network architecture where a remote control plane manages the forwarding state of the data plane. The networking industry has on many occasions shifted from this original view of SDN, by referring to anything that involves software as being SDN. We, therefore, attempt to provide a less ambiguous definition of software-defined networking.

We define an SDN as a network architecture with four pillars:

1. The control and data planes are *decoupled*. Control functionality is removed from network devices that will 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). The flow abstraction allows unifying the behavior of different types of network devices, including routers, switches, firewalls, and middleboxes. Flow programming enables unprecedented flexibility, limited only to the capabilities of the implemented flow tables [McK+08].

3. Control logic is moved to an external entity, the so-called SDN controller or 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.

Note that the logical centralization of the control logic, in particular, offers several additional benefits. First, it is simpler and less error-prone to modify network policies through high-level languages and software components, compared with low-level device-specific configurations. Second, a control program can automatically react to spurious changes of the network state and thus maintain the high-level policies intact. Third, the centralization of the control logic in a controller with a global knowledge of the network state simplifies the development of more sophisticated networking functions, services, and applications.

Following the SDN concept [Sch11], it can be defined by three fundamental abstractions: (*i*) forwarding, (*ii*) distribution, and (*iii*) specification. In fact, abstractions are essential tools of research in computer science and information technology, being already a ubiquitous feature of many computer architectures and systems [Alk+14].

Ideally, the *forwarding abstraction* should allow any forwarding behavior desired by the network application (the control program) whilst hiding details of the underlying hardware. OpenFlow is the first practical realization of one such abstraction, which can be seen as the equivalent to a "device driver" in an operating system.

The *distribution abstraction* should shield SDN applications from the vagaries of distributed state, making the distributed control problem a logically centralized one. Its realization requires a common distribution layer, which in SDN resides in the NOS. This segment has two essential functions. First, it is responsible for installing the control commands on the forwarding devices. Second, it collects status information about the forwarding layer (network devices and links), to offer a global network view to network applications.

The last abstraction is *specification*, which should allow a network application to express the desired network behavior without being responsible for implementing that behavior itself. This can be achieved through virtualization solutions, as well as network programming languages. These approaches map the abstract configurations that the applications express based on a simplified, abstract model of the network, into a physical configuration for the global network view exposed by the SDN controller. Figure 2.1 depicts the SDN architecture, concepts and building blocks.

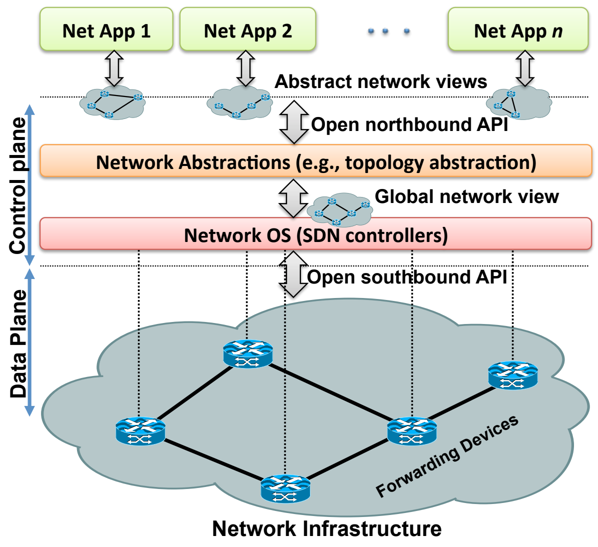

Figure 2.1: SDN architecture and its fundamental abstractions.

As previously mentioned, the strong coupling amongst control and data planes has made it difficult to add new functionality to traditional networks. The introduction of new features requires the inclusion of expensive and hard-to-configure equipment in the network – load balancers and firewalls are common examples. These middleboxes need to be placed strategically in the network, making it even harder to change the network topology, configuration, and functionality later. This can be observed in Figure 2.2. For instance, a Network Intrusion Detection System (NIDS) might need to receive a cloned copy of the traffic of all switching devices of the network through specific physical and or logical links.

In contrast, introducing new functionality in SDN is made simply by adding a new software application to run on top of the NOS. This approach has several advantages:

- It becomes easier to program these applications since the abstractions pro-

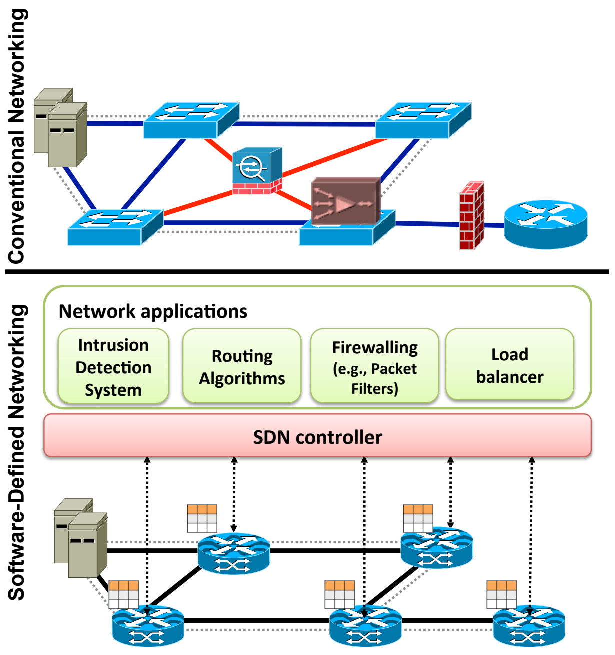

Figure 2.2: Traditional networks versus Software-Defined Networks.

vided by the control platform or the network programming languages can be shared.

- All applications can take advantage of the same network information (the global network view), leading (arguably) to more consistent and effective policy decisions whilst re-using control plane software modules.

- These applications can take actions (i.e., reconfigure forwarding devices) from any part of the network. There is, therefore, no need to devise a precise strategy about the location of the new functionality.

- The integration of different applications becomes more straightforward. For instance, load balancing and routing applications can be combined sequentially, with load balancing decisions having precedence over routing policies.

For further details about the SDN deployment scenarios and infrastructures (e.g., Local Area Network (LAN), Wide Area Network (WAN), optical networks, wireless networks, mobile networks, transport networks, ODTN, IoT, cloud computing) and SDN layers and components such as southbound APIs, northbound APIs, west-/eastbound APIs, network operating systems, programming languages, debugging and troubleshooting, and a vast ecosystem of applications (e.g., routing, traffic engineering, load balancing, firewalls, anomaly detection, monitoring, and so forth), as well as many open challenges, the reader, can check some of the most comprehensive surveys available online [Kre+15]; [Tro+16]; [Alv+17]; [Yan+15]; [NDK16]; [Men+17]; [Thy+16]; [BMV17]; [SNS16]; [RR17]; [Dac+17]; [Yoo+17a]; [ARS18]; [BSM18]; [Far+19]; [Xie+19]; [Sul+19]; [OO18]; [AMA19]; [MR19]; [JCL19]; [Isl+19]; [Gio+20].

2.2 The Threat Plane

Security should (arguably) be an inherent feature of future systems and Internet architectures [Car17]; [PPJ11]. However, several projects still do not consider security by design, i.e., security is usually implemented as an afterthought. Despite security being of crucial importance for the deployment and widespread adoption of SDN [Kre+15]; [Por+15]; [Arb+16]; [TZN16]; [Sco17]; [Han+19], it has been a commonly neglected property [Pas14]; [Kre+15]; [Dac+17]; [Sco17]; [Dac+17]; [Thi+18]; [Han+19]; [Lee+20].

One of the fundamental aspects of SDN is the logical centralization of the control plane, placing the controller at the core of these infrastructures. Its security is though of uttermost importance to ensure the network's correct operation. However, currently available SDN controllers, including production-quality ones such as ONOS and ODL, have several vulnerabilities, including malformed control messages, packet-in flooding, control message drops, infinite loops, flow rule modification, eavesdrop, application eviction, flaws in access control, and authentication components, Host Location Hijacking Attack (HLHA), and state manipulation and man-in-the-middle attacks [Guo+16]; [Noh+16]; [Yoo+17a]; [Lee+17]; [Sec+17]; [Xu+17]; [Sco17]; [TNK18]; [Sec+19]; [Han+19]; [Lee+20]. Worse still, attacks such as *Persona Hijacking* [Jer+17a], where a malicious host fools forwarding devices into believing that it is the legitimate owner of the victim's identifiers, affect even SDN security enforcement solutions such as TopoGuard [Hon+15], SPHINX [Dha+15], and SE-Floodlight [Shi+13b].

Some of the existing vulnerabilities can severely compromise the network operation. For instance, application eviction allows a malicious application to unload another application dynamically (e.g., firewall application), leaving the network defenseless. Another example is handshake without hello message[1], which leads to weak authentication during the handshake setup. This vulnerability allows an attacker to launch a switch identification spoofing attack [Dov17]. With such an attack, a malicious user can either register a fake forwarding device in the network or launch a man-in-the-middle attack. Furthermore, an attacker can use handshake messages to launch a resource exhaustion attack against the internal storage

[1]An issue fixed in the latest versions of controllers such as ONOS and ODL [Lee+20], but still a good example of what can happen in case of a careless handshake.

of the controller, which can result in a controller shutdown [Lee+17].

In the past decade, several threat vectors have been identified in SDN architectures [Kre+15]; [Yoo+17a]; [Lee+20], as well as numerous security and dependability weaknesses in all layers of SDNs (i.e., from forwarding devices up to applications running on controllers) [KKS13]; [WH13]; [SG13]; [Por+12]; [BCS13]; [Yoo+17a]; [SNS16]; [Kha+17]; [Akh+16]; [LMK16]; [Dac+17]; [ZJZ17]; [Dar+17]; [AvW18]; [Par+18]; [Mat+19]; [JKK19]; [NK19]; [Sha+20]; [Lee+20]. Whilst some threat vectors are common to traditional networks, others are more specific to SDN, namely the attacks on control plane communications and targeting the controllers. Most threats are independent of the technology or the protocol, because they represent threats on conceptual and architectural layers of SDN itself.

2.2.1 A systematic analysis of the SDN control plane

In this section, we perform a comprehensive analysis of the threat plane of SDN, attempting at giving a structured view of the threat vectors to SDN [KRV13]; [Kre+15]. As shown in Figure 2.3 and Table 2.1, we identify nine threats vectors in SDN architectures. Threat vector number one consists of forged or faked traffic flows in the data plane, which can be used to attack forwarding devices and controllers. Additionally, it can lead to other specific attacks such as eavesdropping, flow rule modification, and control message drop [Lee+17]. These attacks are made easier when control messages are sent through the data plane (using in-band control plane communication channels), which is usually the case in SDN deployments.

Threat vector number two allows an attacker to exploit vulnerabilities of forwarding devices, and consequently wreak havoc with the network or use the device to outsmart network security, steal sensitive information, and so forth. For instance, a malicious switch can be used by an adversary to outsmart network security, i.e., bypassing middleboxes in the data plane (e.g., firewalls, NIDS) [TSS17].

Threat vectors three, four, and five are arguably the most critical ones. Attacks on control plane communications, controllers, and applications can readily grant an attacker control of the network. For instance, a faulty or malicious controller can be used to reprogram the entire network for data theft purposes.

One specific threat vector we address in this thesis (Chapter 5) is threat number three, which is fundamentally related to the lack of reliable security services and properties, such as device identification, device authentication, device authorization, device association, and confidentiality, integrity, and authenticity of communications. This threat vector opens the doors for different kinds of attacks such as switch identification spoofing, eavesdropping, man-in-the-middle, DoS, flow rule modification, and control message drop [Lee+17]; [Lee+20].

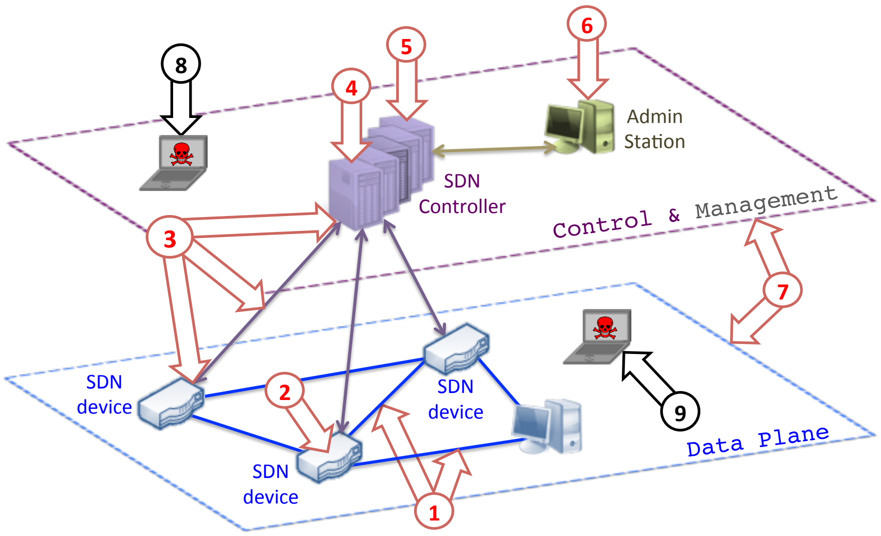

Figure 2.3: Threat vectors of SDN architectures

Threat vector number six is linked to attacks on and vulnerabilities in administrative stations. A compromised critical computer, directly connected to the control network, will empower the attacker with resources to launch more easily an attack to the controller, for instance. Threat vector number seven represents the lack of trusted resources for forensics and remediation, which can compromise investigations (e.g., forensic analysis) and preclude fast and secure recovery modes for bringing the network back into a safe operating condition.

Finally, threat vectors numbers eight and nine represent insider threats such as malicious employees with physical access or special authorizations inside the enterprise infrastructure. Whilst the control plane can be physically isolated, making it harder for most insiders to get access, the data plane is usually accessible to employees of the company and visitors, even though the latter may have access restrictions.

In this thesis we also pay special attention to threat vector nine, particularly the ability to add fake devices (e.g., controllers, switches) in the network. These devices can be used to cause harm to the network (e.g., launch DoS attacks on the control plane, create forwarding loops with the goal of flooding specific links in the network), steal sensitive information, compromise critical systems by evading security mechanisms, and so forth.

It is worth emphasizing that attacks carried out by malicious insiders are particularly dangerous because they are typically done by people who have the required access, knowledge, and know-how for launching effective attacks, leading to severe implications for data security [Nur+14]; [MHP14]; [Kee+05]; [SD16]; [Liu+18]; [Ver19]; [Hom+19]. A malicious insider who has access to the control and management planes (threat vector eight) can easily compromise the entire network. For instance, s/he can launch DoS attacks that can shut down the controller, exploit physical and logical vulnerabilities to get access to the controller to start other kinds of attacks such as application eviction, system command execution, memory exhaustion, internal storage misuse, and system variable manipulation [Lee+17].

Table 2.1: SDN specific vs. non-specific threats

Threat vector	**Specific to SDN?**	**Consequences in software-defined networks**
Vector 1	no	Open door for DDoS attacks, eavesdropping, control message drop, etc.
Vector 2	no	Potential attack inflation.
Vector 3	yes	Exploiting logically centralized controllers.
Vector 4	yes	Compromised controller may compromise the entire network.
Vector 5	yes	Development and deployment of malicious applications on controllers.
Vector 6	no	Potential attack inflation.
Vector 7	no	Negative impact on fast recovery and fault diagnosis.
Vector 8	no	Exploiting control plane communications channels, protocols and systems.
Vector 9	no	Exploiting data plane resources by compromising or adding fake forwarding devices.

As can be observed in Table 2.1, threat vectors three to five are specific to SDN as they stem from the separation of the control and data planes and the consequent introduction of a new entity in these networks — the logically centralized controller. The other vectors were already present in traditional networks. However, the impact of these threats could be larger than today — or at least it may be expressed differently — and as a consequence, it may need to be dealt with differently. For instance, threat vector eight is easier to exploit (e.g., by an insider) in SDN than in traditional networks. Whilst in the latter the attacker has to compromise different vertically integrated control planes, from different manufacturers, in SDN s/he needs to compromise just one single entity, the controller. Other examples of increased impact are threat vectors two and nine. The consequences of attacks exploiting vulnerable or fake virtual switches, which are common on SDNs, are much higher compared to traditional switches [Thi+18].

Furthermore, as previously exemplified, OpenFlow-enabled networks are subject to a variety of security and dependability problems such as spoofing, tampering, repudiation, information disclosure, denial of service, the elevation of privileges, unauthenticated upload of applications, lack of reliability of control plane communications, and single points of failure [KKS13]; [Kre+15]; [SNS16]; [Yoo+17a]; [JKK19]; [Sec+19]. For instance, attacks such as Know Your Enemy (KYE) allow malicious users to gather configuration information of the network without being detected [CDM17]. KYE exploits the on-demand installation of flow rules of OpenFlow-enabled networks, allowing the attacker to discover which conditions are triggering the installation of a given flow rule. This type of information can be afterward used to evade the actions of security systems of the network, such as IPS, Intrusion Detection System (IDS), and other anomaly detection systems. For a comprehensive list of attacks and vulnerabilities on OpenFlow networks, we refer the reader to surveys and papers such as [Kre+15]; [SNS16]; [Yoo+17a]; [TSS17]; [CDM17]; [Kha+17]; [AvW18]; [Sec+19]; [JKK19]; [Lee+20].

2.3 Security requirements of control plane communications

In this section, we summarize security requirements and principles related to securing control plane communications one of the core target of our thesis. We start with traditional security properties (Section 2.3.1), such as confidentiality, integrity, and authenticity. Then, we discuss stronger and more advanced properties (Section 2.3.2), such as post-compromise security, perfect forward secrecy, and post-quantum security. Finally, in Section 2.3.3 we summarize the security principles advocated by the Open Networking Foundation (ONF) [ONF15].

2.3.1 Traditional security properties

Confidentiality, integrity, and authenticity are fundamental security properties of any system and it is not different with SDN [Mat+19]. The lack of any of these properties can lead to severe consequences for the communicating parties such as controllers and forwarding devices. Without confidentiality, an adversary can gather sensitive information such as the data plane configuration and security policies being enforced in the network. The malicious user can use this information to attack specific systems or evade security appliances through techniques such as teleportation [TSS17], for instance.

The lack of integrity can lead to man-in-the-middle attacks where the adversary changes the data being sent from C (e.g., controller) to F (e.g., forwarding device).

Without integrity checks, the latter will have no way to know if the received data is indeed the same sent by C. In this case, configurations sent by the controller to a forwarding device can be changed in real-time by an adversary. Consequently, the switch might be running the adversary's network configuration instead of the one defined by the network operator.

Finally, authenticity allows the receiver F to be sure whether the data was sent by C or not. In practice, by sharing a session key, C and F can check the authenticity of messages, using message authentication codes, as long as the session key does not leak. No third-party will be able to forge an authentic message without knowing the shared secret key known only by C and F. In short, confidentiality, integrity, and authenticity are fundamental requirements for enforcing security on control plane communications. However, these essential properties are not enough for ensuring the security of present, past, and future communications. Other properties are required as well.

2.3.2 Advanced security properties

Systems resilient to attacks from quantum computers (i.e. provide *post-quantum security (PQS)* can be build using symmetric cryptography (or new quantum-resistant cryptographic algorithms) [Ber09]. For instance, several of the current implementations of TLS are not post-quantum secure because they rely on the classic Public Key Infrastructure (PKI) model and signature algorithms. This requires traditional asymmetric cryptography to bootstrap the protocol, i.e., generate the symmetric session keys. Unfortunately, a quantum computer can efficiently compromise this first step of the protocol, leading to a complete failure of the system regarding the security guarantees. However, it is worth noting that organizations such as the National Institute of Standards and Technology (NIST) are currently proposing new quantum-resistant cryptographic algorithms. For instance, post-quantum signature algorithm candidates (e.g., for authentication in TLS 1.3) have been under evaluation recently [SKD20]; [PST20].

Perfect forward secrecy (PFS) is another security property of communication protocols. When a protocol P has PFS, past short-term keys (e.g., session keys or passwords) remain safe even if long-term keys are compromised [MVV96]; [Wu+98]. It means that recorded encrypted communications of past short-term keys cannot be decrypted after a future compromise of the long-term keys, even if the adversary actively interfered in the past communications. Such property is crucial to safeguard the sensitive information of the network, i.e., an adversary should not be able to decrypt past control plane communications amongst controllers and forwarding devices. In this way, the attacker will not be able to reconstruct the configurations of the network that were enforced before the compromise.

Post-compromise security (PCS) is used to protect future communications, i.e., happening after a compromise [CCG16]. Let us assume that C and F are two communicating parties using a protocol P (e.g., TLS) and the secrets of F have been compromised. In this case, we can say that P is post-compromise secure only if C still has a security guarantee about the communication with F, i.e., C needs a way to know if it is communicating with F or with an attacker in possession of F's secrets.

2.3.3 ONF security principles

Table 2.2 summarizes the security principles for SDN as defined by ONF [ONF15]. Security principles 2, 4, 6, and 8 are amongst the central pillars for building secure systems. For instance, the properties of principle 8, such as clear security assumptions, are fundamental when proposing new systems or protocols. The lack of explicit assumptions can void any conclusion or proof about the security of the system [Wan+13]; [GT16]. Similarly, other properties such as simplicity, scalability, and automation are crucial for building robust and state of the art systems [JSV17]; [RWW17]; [RLM19]; [VM15]. Another interesting example is principle 2. Currently, SDN lacks services and mechanisms for robust identification, authentication, authorization, and association of devices in the network, for instance. This leads to several security issues as previously discussed in Section 2.2.

2.4 Securing Control Plane Communications

There are several projects addressing security issues such as user, application, and host authentication through common Authentication, Authorization, and Accounting (AAA) services. In general, those projects rely essentially on password or PKI certificate-based user as well as host authentication (e.g., Remote Authentication Dial-In User Service (RADIUS)/802.1x) [Eng12]; [Tos+14]; [CKM16]; [RHE16]; [Kam+16]; [Wan+19a]; [MD16]; [ONO16]; [Hua+17]; [Ope18d]; [Fan+19]; [JKK19]; [Han+19]; [Mah+19]; [Mol+19]. Additionally, the 802.1x has some drawbacks, such as the individual configuration that must be performed on each switch/host, which can lead to human errors and security policy issues.

To avoid the drawbacks of the 802.1x, people are trying different approaches for IoT and networks that promote the Bring Your Own Device (BYOD) concept. For instance, SAFE proposes a new set of services for SDN, allowing a greater diversity of authentication mechanisms [Kam+16]. SAFE is a controller for isolating unauthenticated hosts that connect to the network. Each non-authenticated host is mapped to a specific network slice, using as reference the host's Media Access Control (MAC) address and the port on which it is connected to the SDN switch.

Table 2.2: Security principles for SDN

#	Principle	Goal
1	Clearly define security dependencies and trust boundaries.	Simplify risk analysis and security control evaluation.
2	Assure robust identity.	Provide a strong identity for authentication, authorization, and accounting.
3	Build security based on open standards.	Promote portability and interoperability of systems.
4	Protect the information security triad.	Ensure confidentiality, integrity, and availability (CIA) on the entire ecosystem.
5	Protect operational reference data.	Ensure the integrity of the reference data (e.g., credentials and sequence numbers, nonces).
6	Make systems secure by default.	Provide multiple security levels to meet the needs of the different system use cases.
7	Provide accountability and traceability.	Ensure comprehensive logging data for auditing/-forensic purposes.
8	Properties of manageable security controls.	Provide a set of properties for new security controls for SDN (e.g., clear assumptions, simplicity, scalability, automation).

Once the host has been authenticated, it can be moved to a different network slice. However, whilst those projects are an important step to improve the security of the network, one of the essential requirements of SDNs is secure and trustworthy authentication and authorization services for the network infrastructure itself. Only then it will be possible to ensure the security of control plane communications amongst controllers and forwarding devices.

Recently, some works have been trying to address security issues related to control plane communications, such as the lack of strong identification and authentication amongst control plane devices [ACW16]; [LP17]; [BF18]; [Mah+19]; [Wan+19a]; [Lam+18]. For instance, lightweight authentication mechanisms, such as HiAuth [AW19], have been proposed for protecting SDN controllers from DoS attacks. HiAuth uses an information hiding technique to verify the legitimacy of forwarding devices through bitwise operations on OpenFlow's XID header filed (or transaction identifier). At the same time that such lightweight technique can be used to mitigate DoS attacks, it does not address other security issues of control plane communications – such as authenticity, integrity, confidentiality, and data freshness.

Furthermore, a considerable number of proposals rely on PKI and TLS to provide essential security properties and security services for control plane communications [CPP15]; [Lam+16]; [Ope18c]; [ONO16]; [AS17]; [LP17]; [MT19]; [Cao+19]; [Yig+19]. Whilst they can provide security properties such as authenticity, integrity, and confidentiality for messages in transit amongst forwarding devices and controllers, it does not (by itself) solve issues such as switch identification spoofing (e.g., OpenFlow Datapath ID (DPID) overwrite), eavesdrop, and man-in-the-middle [KPY15]; [Dov17]; [Lee+17]; [Sec+17]; [Son18]; [Lee+20]. Indeed, several open-source controllers (e.g., ODL, OpenIRIS) are prone to these vulnerabilities. One of the problems related to DPID is that some controllers simply overwrite existing switch connections with new ones. In other cases, controllers just send packets to all switches with the same DPID. Even though OpenFlow uses a DPID to identify the data planes, the protocol does not provide means to authenticate the switch's DPID.

It is also worth noting that the setup process of TLS connections amongst forwarding devices and several controllers is typically manual[2]. This makes them more prone to attacks. For instance, the setup of an Open vSwitch is done by typing a command on the command-line interface (CLI). Firstly, it means that you can connect the forwarding device to any (fake or not) available controller. Secondly, if an attacker spoofs the controller's Internet Protocol (IP), then all forwarding devices will be connecting to a fake controller instead of the real one. From a security perspective, we might need more than the controller's IP address, port number, and a certificate to ensure the security of control plane communications. In Chapters 4 and 5, we provide further information on different issues of PKI-/TLS-based solutions, such as the complexity of the code base, that recurrently leads to vulnerabilities being discovered and new attacks.

There are several related works in the literature, however, in what follows we choose the ones that had a better correlation with our proposal, as summarized at the end of this section. To do that, we analyzed HiAuth (a lightweight authentication mechanism, which was already discussed), ODL (representing controllers that support TLS, such as ONOS, ODL, and Floodlight), R-AAA (Resilient RADIUS for 802.1x authentication using Extensible Authentication Protocol (EAP)-TLS), PKI (needed for issuing certificates for TLS-based solutions), DECIM (detecting the compromise of long term secret keys), PQDSS (post-quantum digital signature schemes), Fleet (SDN controller to mitigate insider threats), as well as ANCHOR

[2]There are some exceptions though. For instance, the Secure Network Bootstrapping Infrastructure (SNBI) of the ODL controller is used to bootstrap devices through IPv6 addressing, neighbor discovery, and 802.1AR credentials [Ope18c]; [Sco17]; [IEE18].

and R-ANCHOR mechanisms.

Differently from most SDN controllers, ODL has a bootstrap service named SNBI [Ope18c]. This service takes advantage of manufacturer-installed IEEE 802.1AR certificates [IEE18] to secure communications and bootstrap forwarding devices. In short, if the device has a valid IEEE 802.1AR certificate, it will be identified, registered, and authorized within the controller's domain. However, we have to rely on PKI as well as TLS and trust the secrets and certificates issued by the manufacturer of the forwarding devices. This might be a problem in a post-Snowden era and because of the issues related to PKI and TLS, as we further elaborate on the following chapters. To give an example, the IEEE 802.1AR standard [IEE18] itself warns users about the quality of the device identification (DevID) secret, which can impair the quality of the generated certificates. Both poor randomness and unfitting management of the secret generation process can compromise security. In Chapter 5, we introduce the design and implementation of high-quality secret generators, such as strong sources of entropy and resilient pseudorandom generators.

PKI [Mau96]; [Her+00]; [Hou+02] is required, by controllers that support TLS, to generate and manage the certificates that can be used for device registration, authentication, and association. However, the controller can only verify if the certificate is valid or not, which means that we do not have fine-grained (or reliable) control of services such as device registration (i.e., with an explicit acknowledgment of network managers). The track record of the managers responsible for each device added to the network can be used for accounting and auditing purposes, for instance. One could also argue that certificate attributes can be used to identify which controllers a forwarding device can associate with. However, this could increase the complexity and OPEX of the network.

Perfect forward secrecy ($_a$) of traditional solutions, such as those provided by the different implementations of TLS, is not easy or simple to enforce. Firstly, despite TLS providing ciphers that offer PFS, in practice, different cipher suites do not feature it [SHS15]. This means that not all implementations and deployments of TLS offer PFS or provide it with very low encryption grade [Hua+14]; [Nam19]; [Dig17]. To give an example, widely deployed web servers, such as Apache and Nginx, may suffer from weak PFS configuration [Dig17]. Research findings also show that most Diffie-Hellman key Exchange (DHE)- and Elliptic Curve Diffie-Hellman key Exchange (ECDHE)-enabled servers use weak Diffie-Hellman (DH) parameters or practices that greatly reduce the protection afforded by PFS, such as private value reuse, TLS session resumption, and TLS session tickets, i.e., provide a false sense of security [Hua+14]; [Adr+15a]; [SDH16].

Some SDN controllers, such as ONOS and ODL, offer resilience ($_b$) in the sense that they provide a distributed control plane. Therefore, if one controller instance goes down, there are still other instances capable of taking over. Nevertheless, in practice, they are neither designed to overcome benign faults (e.g., software bugs affecting all replicas at the same time) nor malicious faults (e.g., DoS attacks, compromise of controller instances) yet.

R-AAA [Kre+14], or resilient RADIUS, is a new fault- and intrusion-tolerant architecture for Authentication and Authorization Infrastructures (AAIs). It uses trusted components and state machine replication to tolerate both crash and Byzantine faults, such as resource exhaustion attacks [Kre+16]. RADIUS is a traditional AAA service using third-party backend services such as OpenLDAP to provide user registration, identification, and authentication $(_e)$. With RADIUS in place, authenticated users are authorized $(_f)$ by the forwarding devices to establish an association (e.g., user/supplication <=> switch) and use networking resources (e.g., access the intranet or Internet). In the case of R-ANCHOR $(_g)$, our initial goal is to provide fail-fast mechanisms, i.e., a way of automatically recovering after a failure or compromise. A full fault- and intrusion-tolerant version of ANCHOR is an interesting future work to pursue.

Differently from the previous systems, PQDSS [Cha+17] solves rather particular issues. PQDSS is one example of a new class of post-quantum digital signature schemes that use only symmetric-key primitives. A system based on such primitives is believed to be quantum-secure. Similarly, we use only symmetric-key primitives in ANCHOR and R-ANCHOR.

DCIM [YRC18] is a new approach for detecting the compromise of a participant in an end-to-end communication (e.g., instant messaging applications such as Signal [Ope19a]). It provides means for the automatic detection of compromised long-term keys through a public append-only Merkle tree log. Once the compromise is detected, users are notified and can take measures to ensure post-compromise security, such as getting a new key/certificate from the underlying PKI. While it does not provide automatic security to the post-compromise recovery process, which is one of our goals, it does automatically detect compromises $(_d)$.

Fleet [MHP14] proposes a new SDN controller to mitigate insider threats (ITM), i.e., to reduce the privilege of network administrators. It uses threshold cryptography and assumes that up to k out of n network managers can be malicious without compromising data plane configuration updates on a multi-controller scenario (e.g., one network manager per controller). In R-ANCHOR, to mitigate insider threats, we require the explicit acknowledgment of $f+1$ managers to register a new forwarding device in the network, for instance.

Table 2.3 summarizes the related work regarding essential security properties, functions, and services that provide (or can be used to) secure control plane communications amongst controllers and forwarding devices.

Table 2.3: Securing control plane communications

Functionality		HiAuth	Fleet	ODL	PKI	DECIM	PQDSS	R-AAA	ANCHOR	R-ANCHOR
Reference		[ACW16]	[MHP14]	[Ope18a]	[Mau96]	[YRC18]	[Cha+17]	[Kre+14]	[Kre+19]	
Registration		✗	✗	✓	✗	✗	✗	✓$_{(e)}$	✓	✓
Identification		✓	✗	✓	✓	✗	✗	✓$_{(e)}$	✓	✓
Authentication		✓	✗	✓	✓	✗	✗	✓$_{(e)}$	✓	✓
Authorization		✗	✗	✓	✓$_{(c)}$	✗	✗	✓$_{(e)}$	✓	✓
Association		✗	✗	✗	✓	✗	✗	✓$_{(f)}$	✓	✓
Properties	CIA	✗	✗	✓	✗	✗	✗	✓	✓	✓
	PFS	✗	✗	✓$_{(a)}$	✗	✓	✗	✓$_{(a)}$	✓	✓
	PCS	✗	✗	✗	✗	✓	✗	✗	✓	✓
	PQS	✗	✗	✗	✗	✗	✓	✗	✓	✓
Robustness	ITM	✗	✓	✗	✗	✗	✗	✗	✗	✓
	ReS	✗	✗	✓$_{(b)}$	✗	✗	✗	✓	✗	✓$_{(g)}$
	AuS	✗	✗	✗	✗	✓$_{(d)}$	✗	✗	✗	✓$_{(h)}$
Encryption		✗	✗	✓	✗	✓	✗	✓	✓	✓

Confidentiality, Integrity, Availability (CIA);
Insider Threat Mitigation (ITM);
Resilient Services (ReS);
Automatic Security (AuS) (e.g., automatic PCS/Post-Compromise Recovery (PCR)).

Finally, it is worth noting that works such as FortNOX [Por+12], AVANT-GUARD [Shi+13c], LineSwitch [Amb+15], TopoGuard [Hon+15], CPMF [Son+17], SDNsec [Sas+16], SE-Floodlight [Shi+13b], SPHINX [Dha+15], SFaaS [Kuo+18], SODA [Kim+19], Defense4All [Ope19b], amongst others [SNS16]; [RR17]; [LMK16]; [IK18]; [Han+19], provide different security and resilience features, as well as services, to SDN, such as protection mechanisms against DoS attacks on control and data planes, innovative algorithms for detecting malicious switches, efficient control path recovery, application isolation, automatic conflict resolution for concurrent flow rules and incremental validation of network configuration updates. Others, such as FRESCO [Shi+13a], introduce frameworks for simplifying and accelerating the composition of new security services (e.g., firewalls) using OpenFlow. However, all these proposals are complementary to ours. In fact, there is (arguably) an open avenue for integrating solutions such as AVANT-GUARD, LineSwitch, FRESCO, SODA, Defense4All, and ANCHOR/R-ANCHOR, for instance.

2.5 Final Remarks

Several attacks against the SDN infrastructure exploit different vulnerabilities of the control plane, such as the lack of authentication, authorization, and other security properties, namely integrity, confidentiality, and data freshness [AAS14]; [Kre+15]; [SNS16]; [Han+19]. However, despite the growing number of works addressing security issues of SDN [Por+12]; [SG13]; [SOS13]; [Shi+13a]; [Shi+14]; [WAv17]; [ANH17]; [Han+19]; [NK19]; [AAP19], not much attention has been given to device identification, authentication, and authorization, as well as control plane associations and communication amongst devices, the major aspect we address in this thesis.

Implementing trust amongst controllers and forwarding devices is one of the main requirements for ensuring that malicious elements cannot harm the network without being detected. For instance, an attacker can spoof the IP address of the controller and make switches connect to its fake controller. This is currently the case since most controllers and switches establish insecure Transport Control Protocol (TCP) connections or lack trust management solutions [Sam15]; [Azi+18]; [Han+19]; [Mat+19]. Moreover, even the controllers that support TLS have vulnerabilities that allows an adversary to launch attacks such as switch identification spoofing [Lee+17]. One way of mitigating this lack of trust amongst controllers and forwarding devices is by using a secure logically centralized root-of-trust, as we propose.

Furthermore, to the best of our knowledge, none of the related works provides a comprehensive and by design approach for solving security issues of SDN effectively. Our solution, ANCHOR, is designed to provide all properties and functionalities as aforementioned. Furthermore, an architectural approach such as the one we propose here (which ultimately led to following the SDN philosophy of "logical centralization") is lacking. Importantly, this approach allowed us to gain a global perspective of the relevant gaps in SDN and the limitations of existing solutions to the problem. This first step gave insight into one of the most relevant problems of SDN (as noted as well by ONF and MEF (MEF) security groups [ONF19]; [MEF20]): the security of the associations and communications amongst devices.

Finally, taking into account the current status-quo, it is worth emphasizing a few recommendations and open challenges:

1. Security should be baked in from the start (i.e., security by design, as examples in the literature show [Cas+18]; [LS17]; [CHH16]; [Pas14]; [Car17]; [Dac+17]; [Han+19]; [FM19]).

2. We must make sure that the security amongst forwarding devices and controllers is configured properly [Pas14].

3. The SDN ecosystem is becoming too complex, which means that it is time to re-think the design and modularization of the SDN architecture [TZN16]; [Guo+16]; [Dac+17]; [Lee+17]; [Sec+17]; [Sco17]; [AvW18].

4. Several OpenFlow controllers supporting TLS are vulnerable to several attacks on applications and control plane communications [Noh+16]; [Lee+17]; [Sec+17]; [Yoo+17a]; [Xu+17]; [MT19].

5. Proper dynamic device registration and association, as well as trust management amongst controllers and forwarding devices, are essential requirements to SDN [ONF15]; [Kre+15]; [Yoo+17a]; [Par+17]; [Han+19].

6. Current OpenFlow controllers implement only a fraction of the security mechanisms required to address the security issues introduced in previous sections. IP check, user authentication, anti-DoS from computing capacity exhaustion, closing unnecessary ports/services, authorization for access to sensitive data, and privileged control of applications are examples of security checks and mechanisms currently implemented in OpenFlow controllers [Guo+16]; [McB+13]; [Han+19].

Our main goal is to work towards a holistic (and by design) solution for addressing recommendations and open challenges from 1 to 5. To achieve our goal, we propose the logical centralization of non-functional properties, taking security as our use case.

Chapter 3

Logical Centralization Revisited

In this chapter, we start by reviewing and elaborating on the principle of using logical centralization for the successful enforcement of generic non-functional properties in SDN, showing its advantages and presenting the generic blueprint of the ANCHOR architectural concept (Section 3.1). Then, we apply the roadmap to specializing the architecture to a given non-functional property class. In our case, we identify the broad security challenges of SDN and extract requirements to be fulfilled by the architecture (Section 3.2). We conclude the chapter by presenting the outline of a logically-centralized security architecture based on ANCHOR (Section 3.3), whose building blocks will be detailed in the next chapters.

3.1 ANCHOR Blueprint

As explained in the introduction, the logical centralization of the provision of non-functional properties allows us to: (1) define and enforce global policies for those properties; (2) reduce the complexity of controllers and forwarding devices; (3) ensure higher levels of robustness for critical services; (4) foster interoperability of the non-functional property enforcement mechanisms; and finally (5) better promote the resilience of the architecture itself. Let us explain the rationale for these claims.

Define and enforce global policies for non-functional properties. One can enforce non-functional properties through piecewise, partial policies. But it is easier and less error-prone, as attested by SDN architectures with respect to the functional properties, to enforce e.g., security or dependability policies, from a central trust point, in a globally consistent way. Especially when one considers changing policies during system lifetime.

Reduce the complexity of controllers and forwarding devices. One of the most powerful ideas of SDN was exactly to simplify the construction of devices, by stripping them of functionality, centralized on controllers. We are extending the scope of the concept, by relieving both controllers and devices from ad-hoc and redundant implementations of sophisticated mechanisms that are bound to have a critical impact on the entire network.

Ensure higher levels of robustness for critical services. Enforcing non-functional properties like dependability or security has a critical scope, as it potentially affects the entire network. Unfortunately, the robustness of devices and controllers is still a concern, as they are becoming rather complex, which leads to several critical vulnerabilities, as amply exemplified in [SNS16]; [Dac+17]; [Akh+16]; [Ahm+15]; [IK18]; [AAP19]; [Viz+19]. For these reasons, a single device or controller may become a single point of failure for the network. A centralized concept as we advocate might considerably improve on the situation, exactly because the enforcement of non-functional properties would be achieved through a specialized subsystem, minimally interfering with the SDN payload architecture. A dedicated implementation, carefully designed, would be re-usable, not re-implemented, by the payload components.

Foster interoperability of the non-functional property enforcement mechanisms. Different controllers require different configurations today, and a potential lack of interoperability in terms of non-functional properties arises. Global policies and mechanisms for non-functional property enforcement would also mean an easy path to foster controller and device interoperability (e.g., East and Westbound APIs) in what concerns the former. This way, mechanisms can be modified or added, and have a global repercussion, without the challenge of having to implement such services in each component. As another example, it is easier to ensure the security and dependability of a modularized and small code base than a huge controller. Differently from a controller, the "logically-centralized" non-functional properties do not have stringent performance requirements, such as latency. This allows us to add advanced protocols and mechanisms to ensure the resiliency of the architecture. Moreover, similarly to control applications, new functionalities can be added to the "logically-centralized" architecture by adding new modules. In most cases, this could be simpler than to change an SDN controller. For instance, ODL has already more than 3.7M lines of code [TFS19].

Better promote the resilience of the architecture itself. Having a specialized subsystem architecture already helps for a start, since for example, its operation is not affected by latency and throughput fluctuations of the (payload) control platforms themselves. However, the considerable advantage of both the decoupling and the centralization, is that it becomes straightforward to design in security and dependability measures for the architecture itself, such as advanced techniques and mechanisms to tolerate faults and intrusions (and in essence overcome the main disadvantage of centralization, the potential single-point-of-failure risk).

The means to fulfill the "logically-centralized" perspective of non-functional property enforcement and achieve the above-mentioned objectives, is the ANCHOR subsystem architecture, whose blueprint and general outline is depicted in Figure 3.1. ANCHOR does not modify the essence of the current SDN architecture with its payload controllers and forwarding devices, but rather stands aside, 'anchors' (logically-centralizes) crucial functionality and properties and 'hooks' to the latter components, in order to enforce the desired properties.

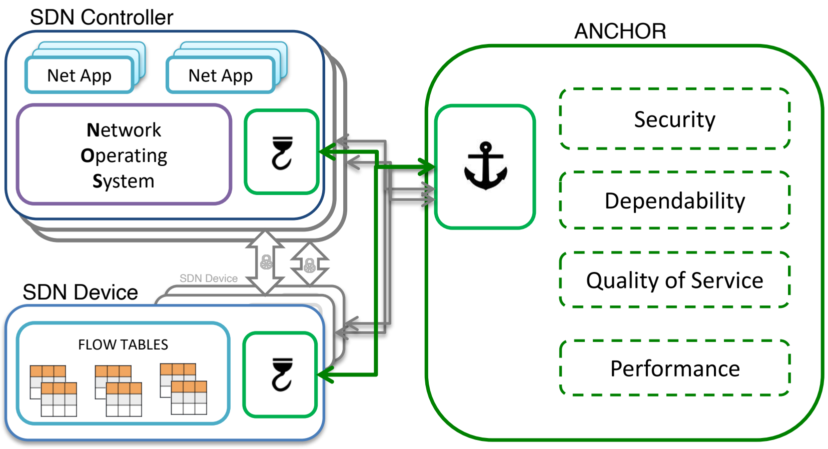

Figure 3.1: Logically-centralized enforcement of non-functional properties

Thus, it acts as a centralized anchor of trust, a specific middleware whose main aim is to ensure that certain properties – for example, the security of control plane associations and of communication amongst controllers and forwarding devices, or the dependability of controller operations – are met throughout the architecture. So, on commercial off-the-shelf (COTS) forwarding devices and controllers, we just need to add the local counterparts to the ANCHOR middleware mechanisms and protocols, or HOOKs, to interpret and follow the ANCHOR's instructions.

The question is now how to specialize ANCHOR for a given purpose. We identified at least four steps to achieve such goal: (a) select the class(es) of properties to enforce (security, dependability, quality-of-service, etc.); (b) identify the current gaps that stand in the way of achieving such properties in SDNs; (c) design a logically-centralized subsystem architecture and middleware, with hooks to the main SDN architectural components, in a way that they can inherit the desired specific properties; (d) populate the middleware with the appropriate mechanisms and protocols to enforce the desired properties/predicates, across controllers and forwarding devices, in a global and consistent manner.

We conjecture that the general concept and architectural principles can satisfactorily enhance the current state of the art with regard to non-functional properties (e.g., security, dependability, performance, quality of service). We are going to validate our proposal with *logically-centralized security*, and we leave it to be validated in the future for other property classes.

3.2 Security Challenges

Our approach of using an architecture enforcing logically-centralized security, for solving the main security problems for SDN and meets some challenges, inspired by the comprehensive state-of-the-art review performed in Chapter 2. We organized them around a few gaps that have to be bridged in designing the architecture.

In the following subsections, we discuss the main challenges posed by these gaps — (i) security-performance; (ii) complexity-robustness; (iii) global security policies; (iv) resilient roots-of-trust; (v) simplicity, automation and security by default; and (vi) insider threats — as well as the requirements they put on such a logically-centralized architecture for enforcing security as a non-functional system property.

3.2.1 Security *vs* performance

The security-performance gap comes from the conflict between ensuring high performance and using secure primitives. This gap affects directly the control plane communication, which is the crucial link between controllers and forwarding devices, allowing remote configuration of the data plane at runtime. Control channels need to provide high performance (high throughput and low latency) while keeping the communication secure.

The latency experienced by control plane communication is particularly critical for SDN operation. The increased latency is a problem *per se*, in terms of reduced responsiveness, but may also limit control plane scalability, which can be particularly problematic in large datacenters [BAM10]; [KDH18]. Most of the existing commercial switches already have low control plane performance on TCP (e.g., a few hundred flows/s [Kre+15], see Section V.A.). Adding crypto worsens the problem: previous works have demonstrated that the use of cryptographic primitives has a perceivable impact on the latency of sensitive communication, such as Voice over Internet Protocol (VoIP) [She+12] (e.g., TLS incurs 166% additional Central Processing Unit (CPU) cycles compared to TCP), network operations protocols such as Simple Network Management Protocol (SNMP) [SM11], Network Time Protocol (NTP) [DSZ16], OpenFlow-enabled networks [Kre+17], and Hypertext Transfer Protocol Secure (HTTPS) connections [Nay+14]. Perhaps not surprisingly, the number of SDN controllers and switching hardware supporting TLS (the protocol recommended by ONF to address security of control plane communication [ONF14]; [ONF15]) is still low [Sam15]; [ACW16]; [SNS16]; [LMK16]; [BEI19].

Recent research has indeed suggested that one of the reasons for the slow adoption is related to the security-performance trade-off [Kre+18].

Recent research confirms that other essential security mechanisms introduce additional overhead on the control plane. For instance, security extensions such as topology permission and header space permission impose an overhead of 5% to 20% on state of the art controllers such as ONOS and ODL [Yoo+17b].

Ideally, we would have both security robustness and performance on control plane channels. Considering the current scenario of SDN, it therefore seems clear the need to investigate lightweight alternatives for securing control plane communication. In the context of the security-performance gap, some directions that we point to in our architectural proposal ahead are, for instance, the careful selection of cryptographic primitives [Kre+18], and the adoption of cryptographic libraries exhibiting a good performance-security trade-off, such as NaCl [BLS12], or of mechanisms allowing per-message one-time-key distribution (e.g., iDVV [Kre+18]). We return to these mechanisms later.

3.2.2 Complexity *vs* robustness

The complexity-robustness gap represents the conflict between the current complexity of security and crypto implementations, and the negative impact this has on robustness and hence correctness, hindering the ultimate goal.

In the past few years, studies have recurrently shown several critical misuse issues of cryptographic APIs of different TLS implementations [Ege+13]; [Buh+15]; [Raz+17]; [FWC16]; [Gre17]; [Cho+17a]; [Aca+17]; [GPM15]; [AY18]; [LW18]; [Wan+19c]. One of the main root causes of these misuse issues is the inherent complexity of traditional cryptographic APIs and the knowledge required to use them without compromising security. For instance, more than 80% of the Android mobile applications make at least one mistake related to cryptographic APIs.

Recent studies have also found different vulnerabilities in TLS implementations and have shown that longstanding implementations, such as OpenSSL[1], including its extensive cryptography, is unlikely to be completely verified in the near future [Beu+15]; [FWC16]. To address this issue, a few projects, such as miTLS [Bha+13] and Everest [Bha+17], propose new and verified implementations of TLS. Other initiatives, such as TLSAssistant [MRS19], take a different approach. TLSAssistant tries to analyze TLS configurations and recommend appropriate mitigations for viable attacks. However, different challenges remain to be addressed before having a solution ready for wide use, such as empirically prove

[1]OpenSSL suffers from different fundamental issues such as too many legacy features accumulated over time, too many alternative modes as result of trade-offs made in the standardization, and too much focus on the web and Domain Name System (DNS) names.

that verified software is better, deploy small-Trusted Computing Base (TCB) versions to demonstrate resistance to practical attacks, reconcile low level structured memory model with "big array of bytes" view of memory, and time of (eventual) widespread adoption (e.g., it might take one decade) [Bha+17].

While the problem persists, the number of dangerous occurrences proliferates. Recent examples include vulnerabilities that allow recovering the secret key of OpenSSL at a low cost [YB14], timing attacks that explore vulnerabilities in both PolarSSL and OpenSSL [AF13]; [BT11], several fault attacks on OpenSSL's implementation of elliptic curve cryptography [TT19], secret security patches on OpenSSL, LibreSSL, and BoringSSL with several lagging days, allowing attackers to exploit "0-day" vulnerabilities for long periods of time [Wan+19b], MITM attacks on the control plane of SDN [Jer+17b]. On the other hand, failures in classical PKI-based authentication and authorization subsystems have been persistently happening [Cro17]; [PwC14]; [Hil13]; [Meu13]; [Hil13]; [MB16]; [Hep+19]; [SHC19], with the sheer complexity of those systems being considered one of the root causes behind these problems. Some renowned cryptographers are even arguing that (in a post-Snowden era) "PKI is too flawed and dangerous. We need to come up with something new." [Sha15].

Considering the widely acknowledged principle that simplicity is key to robustness, especially for secure systems, we advocate and try to demonstrate that the complexity-robustness gap can be significantly closed through a methodical approach toward less complex but equally secure alternative solutions. NaCl [BLS12], which we mentioned in the previous section, can be rightly called again in this context: it is one of the first attempts to provide a less complex, efficient, yet secure alternative to OpenSSL-like implementations. Mechanisms simplifying key distribution, authentication and authorization, such as iDVVs [Kre+18], could help mitigate PKI's problems. By following this direction, we are applying the same principle of vulnerability reduction used in other systems, such as unikernels, where the idea is to reduce the attack surface by generating a smaller overall footprint of the operating system and applications [WK16].

In summary, we can argue that the complexity of the most common TLS implementations (e.g., OpenSSL) and the required PKI, leads to somewhat unbalanced complexity-robustness tradeoffs. On the other hand, less complex and equally secure libraries, such as NaCl [BLS12], can provide the same levels of security with high performance. Indeed, NaCl (or libsodium in Linux systems) is being widely used in research, applications, projects, programming languages, and companies [Pet+13]; [NM16]; [Akh17]; [YB14]; [The19c]; [Par19]; [VFV15]; [SDW17]; [Lua+19]; [Fre+19]; [TB19]. The community around this crypto library is proliferating. The following excerpt from "the anatomy of a bad idea" [Gre12], by cryptographer Matthew Green, defines and compares NaCl to OpenSSL in simple words.

> "OpenSSL is the space shuttle of crypto libraries. It will get you to space, provided you have a team of people to push the ten thousand buttons required to do so. NaCl is more like an elevator – you just press a button and it takes you there. No frills or options.
>
> I like elevators"

Recent search confirms this excerpt by concluding that APIs (e.g., NaCl) designed for simplicity can provide security benefits – reducing the decision space, i.e., preventing the choice of insecure parameters – through convenient and straightforward interfaces [Aca+17]. But, this by itself is not enough. Proper documentation with secure and easy-to-use code examples is also a requirement.

3.2.3 Global security policies

The impact of the lack of global security policies can be illustrated with different examples. Although ONF describes data authenticity, confidentiality, integrity, and freshness as fundamental requirements to ensure the security of control plane communication, it does so in an abstract way, and these measures are often ignored, or implemented in an ad-hoc manner [SNS16]; [Han+19]. Another example is the lack of strong authentication and authorization in the control plane. Recent reports show that widely used controllers, such as Floodlight and OpenDaylight, employ weak network authentication and other security mechanisms [WAv17]; [SNS16]; [Hon+15]; [Dov13]; [WW18]; [Viz+19]; [Dix+18]. This leads to any forwarding device being able to connect to any controller. However, fake or hostile controllers or forwarding devices should not be allowed to become part of the network, in order to keep the network in healthy operation.

From a security perspective, it is non-controversial that device identification, authentication and authorization should be among the forefront requirements of any network. All data plane devices should be appropriately registered and authenticated within the network domain, with each association request between any two devices (e.g., between a switch and a controller) being strictly authorized by a security policy enforcement point. In addition, control traffic should be secured, since it is the fundamental vehicle for network control programmability. This begs the question: why aren't these mechanisms employed in most deployments?

A strong reason for the current state of affairs is the lack of global guiding and enforcement policies. It is necessary to define and establish global policies, and design, or adopt, the necessary mechanisms to enforce them and meet the essential requirements in order to fill the policy gap. With policies put in place, it becomes easier to manage all network elements, with respect to registration, authentication, authorization, and secure communication.

3.2.4 Resilient roots-of-trust

A globally recognized, resilient root-of-trust, could improve the global security of SDN, since current approaches to achieve trust are ad-hoc and partial [ACW16]; [Han+19]. Solving that gap would assist in fostering global mechanisms to ensure trustworthy registration and association between devices, as discussed previously, but the benefits would be greater. For instance, a root-of-trust can be used to provide fundamental mechanisms (e.g., sources of strong entropy or pseudo-random generators), which would serve as building blocks for specific security functions.

As a first example, modern cryptography relies heavily on strong keys and the ability to keep them secret. The core feature that defines a strong key is its randomness. However, the randomness of keys is still a widely neglected issue [VH14]; [DK16]; [Str16]; [AM18]; [MGM19]; [Wan+20], and not surprisingly, weak entropy, and weak random number generation have been the cause of several significant vulnerabilities in software and devices [HFH16]; [Alb+15]; [KHL13]; [Hen+12]; [Str16]; [Wan+20]. For instance, findings show that even longstanding cryptographic libraries, such as OpenSSL, have critical security weaknesses on the Random Number Generator (RNG) due to the lack of strong entropy [Str16]. Research has also shown that there are still non-negligible problems for hosts and networking devices [Hen+12]; [Alb+15]; [HFH16]. For instance, a common pattern found in low-resource devices, such as switches, is that the random number generator of the operating system may lack the input of external sources of entropy to generate reliable cryptographic keys. Even long-standing cryptographic libraries such as OpenSSL have been recurrently affected by this problem [KHL13]; [Ope16]. As weak seeds can compromise the output of a Pseudo Random Generator (PRG) [VH14], it is essential to have sufficient entropy in the system. Otherwise, weak seeds can lead to crypto systems as strong as a "wall of cards".

Similarly, as a second example, sources of accurate time, such as the local clock and the network time protocol, have to be secured to avoid attacks that can compromise network operation, since time manipulation attacks (e.g., NTP attack [Mal+16]; [Ste15]) can affect the operation of controllers and applications. For instance, a controller can be led to deliberately disconnect forwarding devices if it wrongly perceives the expiration of heartbeat message timeouts.

It is worth emphasizing that the resilient roots-of-trust gap lies exactly in the relative trust that can be put in partial, local, ad-hoc implementations of critical functions by controller developers and manufacturers of forwarding devices, in contrast to a careful, once-and-for-all architectural approach that can be reinstantiated in different SDN deployments. The list not being exhaustive, we claim that strong sources of entropy, resilient, indistinguishable-from-random number generators, and accurate, non-forgeable global time services, are fitting examples of such critical functions to be provided by logically-centralized roots-of-trust, helping close the former gap.

3.2.5 Automation, simplicity, and security by default

Recurrently, reports suggest that human error is one of the main causes of network downtime [Jun08]; [Bed16]; [Gov+16]; [Sah+17]; [Ace+18]; [Pat19]; [Niz19]. Automation and simplicity are among the core 'tools' to address this problem. For instance, network operators are starting to replace higher layer devices (e.g., IP routers) with Ethernet switches [Sof09]; [KSM13]; [Maa+18]; [Mor+18]; [Kou+19]; [MBM17]. Another example is a recent joint research between academia and industry for developing simplified methods for configuring security services in SDN-based networks [Par+19].

The main reason for the widespread adoption of Ethernet networks is its simple and straightforward configuration. However, Ethernet does not provide the necessary mechanisms to enforce strong security in the network. To address this issue, the use of centralized cryptography schemes and centralized sources of trust to authenticate and authorize known entities has been pointed out as a direction for improving the security of Ethernet networks [KSM13]. Similarly, network security as a service as been suggested to provide the required security for enterprise SDN networks [SNS16]; [Han+19].

As recommended by organizations such as ONF [ONF19], security should be enforced by default in SDN. It becomes clear that networks must have and enforce crucial services such as device registration, authentication, and authorization. Additionally, in a post-Snowden era, it is utterly important to ensure security properties such as confidentiality, integrity, authenticity, perfect forward secrecy, post-compromise security, and post-quantum security for control plane communications.

Finally, the automation of security services is also fundamental. For instance, installing a new device in the network should be as simple as to physically (or logically) connect it to the network and provide a bootstrap authentication code, meaning that we should have a minimal manual intervention. Other security services, such as device association and system recovery after a compromise, should be automated as well, avoiding error-prone manual configuration.

3.2.6 Insider threats

Insider threats (e.g., physical attacks carried out by internal employees) are already one of the top security threats [Gog17]; [Ver19]; [Ekr19]; [Bet19]; [The19a]; [Liu+18]; [Hom+19]; [Cob20]; [Pon20]. Indeed, statistics show that insider attacks happen in practice more often than thought. In 2019, we saw a rise of

31% in the cost of insider threats and an average of seventy-seven days to contain them [Cob20]; [Pon20], for instance.

Insider attacks are particularly dangerous because they are typically done by insiders who have the required knowledge and know-how for launching highly effective attacks, leading to severe implications for data security [Kee+05]; [SD16]; [Nur+14]; [MHP14]. Indeed, reports show that leaks of sensitive data and theft of intellectual proprietary have become a concern for organizations of all sizes [Nur+14]; [SD16]; [Cob20]; [Pon20]. Surprisingly, more than 40% of Information Technology (IT) professionals, with high privileges in the network and systems, see themselves as the most significant threat to the enterprise [NOL17]. One of the main reasons is that they usually hold the 'keys to the kingdom'.

As a result of this scenario, security threats in SDN infrastructures are even more concerning when taking into account malicious insiders (e.g., malicious network administrators) [MHP14]; [Gar+19]; [Cho+19]. Indeed, SDN opens new doors for malicious insiders. For instance, a malicious administrator, using a single fake software-based forwarding device, can easily mirror internal network traffic or attack the control plane. Therefore, it is vital to mitigate the threats posed by malicious insiders.

3.3 Logically-Centralized Security

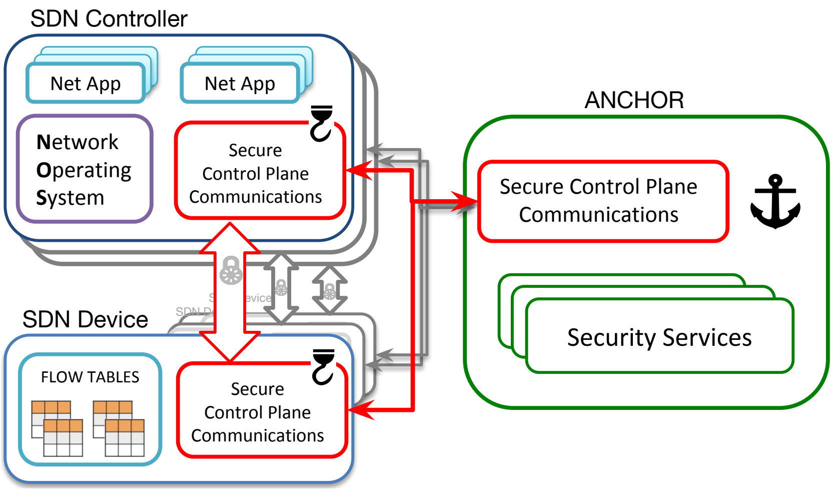
SDN Controller
Net App
Net App
Network Operating System
Secure Control Plane Communications
ANCHOR
Secure Control Plane Communications
Security Services
SDN Device
FLOW TABLES
Secure Control Plane Communications

Figure 3.2: ANCHOR architecture for logically-centralized security

We now approach the last steps of specializing ANCHOR for a given purpose — security in this case: (d) populate the middleware with the appropriate mechanisms and protocols to enforce the desired properties/predicates, across controllers and devices, in a global and consistent manner.

Figure 3.2 gives a snapshot of the proposed logically-centralized security architecture. From the analysis made, we consider two major challenges to be met at architectural level:

- The design and implementation of a root of trust to enforce logically-centralized security, providing mechanisms and protocols for essential security services.
- The design and implementation of an infrastructure bringing simplicity and performance to the provision of secure control plane communications.

These form the main concrete blocks of the architecture, to be discussed in detail in Chapters 4 (KISS) and 5 (ANCHOR).

We refer to our comprehensive analysis of the threat plane of SDN presented in Section 2.2, in order to get insight about concrete threats that have to be addressed when responding to the general requirements reviewed in this section. Since we assume security as our non-functional property use case, that leaves us with nine threat vectors to attend to of which, as our initial goal, we choose threat vectors three, eight and nine, as shown in Figure 3.3 and Table 3.1. We describe at least nine vulnerabilities associated with these threat vectors.

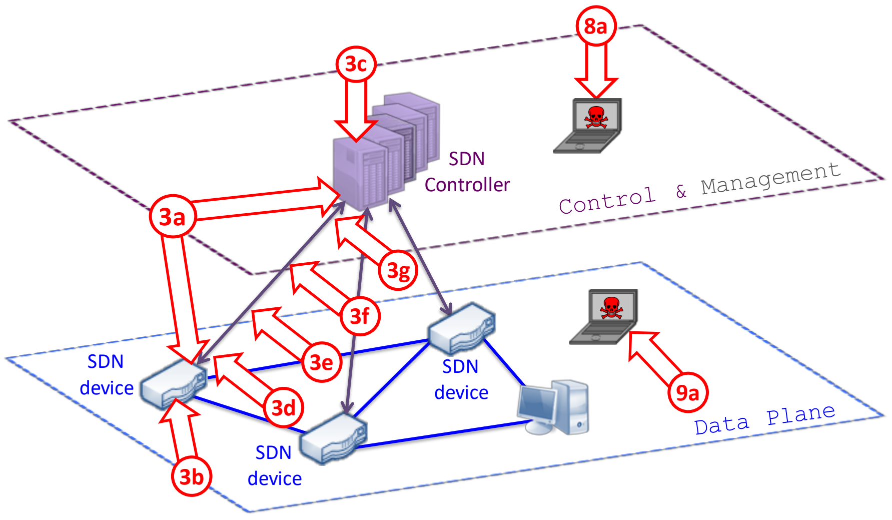

Figure 3.3: Vulnerabilities related to threat vectors three, eight and nine.

Table 3.1: Vulnerabilities and main consequences.

Vul.	Description	Main consequence
3a	Weak identification, authentication, and authorization mechanisms.	It becomes easy to add fake forwarding devices and controllers.
3b	Fake forwarding device.	High impact attacks on control and data planes.
3c	Fake controller.	Easy take over of the network's control.
3d	Lack of confidentiality, integrity, and authenticity.	Leak sensitive information about the network configuration.
3e	Lack of perfect forward secrecy.	Leak pre-compromise sensitive information.
3f	Lack of post-compromise security.	Leak sensitive information after a compromise.
3g	Lack of post-quantum security.	Security properties become useless against attacks using quantum computers.
8a	Malicious control plane insiders.	The security of control plane services and applications gets jeopardized.
9a	Malicious data plane insiders.	Fake and malicious forwarding devices in the network.

Vulnerabilities 3d, 3e, 3f, and 3g are related to security properties such as confidentiality, integrity, perfect forward secrecy, and post-compromise security. The lack (or weak enforcement) of such properties can lead to security issues such as the leak of sensitive information about the network configuration (see Table 3.1). Such information can be used by adversaries to launch targeted attacks or to evade security enforcement middleboxes. For instance, by knowing the configuration of the network, an attacker can evade the NIDS.

In summary, we can list a few questions to be answered when designing the concrete blocks of the architecture in Chapters 4 and 5:

1. How to provide essential security by design (and by default)?
2. How to securely add and remove devices to/from the network?
3. How to ensure the security of past communications between devices in case of a compromise?

4. After a post-compromise recovery, how to ensure the security of future communications between devices?

5. How to mitigate insider threats?

6. How to add security to control plane communications without heavily impairing performance (e.g., the throughput of 20M flows/s [Voe+13] and latency of 10 µs [BAM10]) and without increasing CAPEX/OPEX?

Chapter 4

Secure Control Plane Communications

In this chapter, we introduce the KISS infrastructure, a secure SDN control plane communications architecture, which aims to increase the robustness of control communications whilst enhancing their performance, by decreasing the complexity of the support infrastructure, as an alternative to current approaches based on classic configurations of TLS and PKI. We compare our solution with traditional ones and show how it can improve the performance without impairing security.

We have organized the chapter as follows. In Sections 4.1 and 4.2, we introduce the KISS architecture and its core novel component, the iDVV, a deterministic but indistinguishable-from-random secret code generation protocol. The concept was inspired by the integrated Card Verification Values (iCVV) used in credit cards to authenticate and authorize transactions in a secure and inexpensive way. We develop and extend the idea for SDN, proposing a flexible method of generating iDVVs by adapting proven one-time password-like techniques. iDVV codes allow the safe decentralized generation/verification of keys at both ends of the channel, at will, even on a per-message basis.

To understand and minimize the cost of security, we quantify (Section 4.3.1) the impact of secure primitives on the performance and scalability of control plane communications, through a comparison study of different implementations of TCP vs. TLS, complemented by a deeper study of underlying hashing and Message Authentication Code (MAC) primitives. This in-depth study lead to the selection of the Networking and Cryptography library (NaCl) cryptographic library [BLS12], and the best performing MAC and hash primitives — Poly1305 and SHA512 OpenSSL – as the baseline secure channel technologies for KISS.

Furthermore, in Section 4.3 we evaluate the iDVV design in terms of performance, security and randomness. Key generation latency of iDVV compares favorably with common implementations of key derivation functions. On the security side, we prove the indistinguishability-from-random and determinism of the iDVV generator. Finally, the iDVV successfully passed several empirical randomness tests, further confirming its indistinguishability-from-random, and showing its suitability for highly-robust key generation. We end the chapter with a short discussion.

4.1 KISS architecture

In this section we present our proposal for KISS, a secure control plane communications architecture for SDN offering alternatives to classic configurations of secure channel and authentication protocols and subsystems followed in TLS and PKI.

Figure 4.1 details the integration of the KISS infrastructure in the architecture for logically-centralized security that had been suggested earlier in Chapter 3 (Figure 3.2). In the following explanations, we assume a typical SDN architecture, composed of off-the shelf controllers and forwarding devices, and we further assume that device registration and association services are in place (see Chapter 5 for further details).

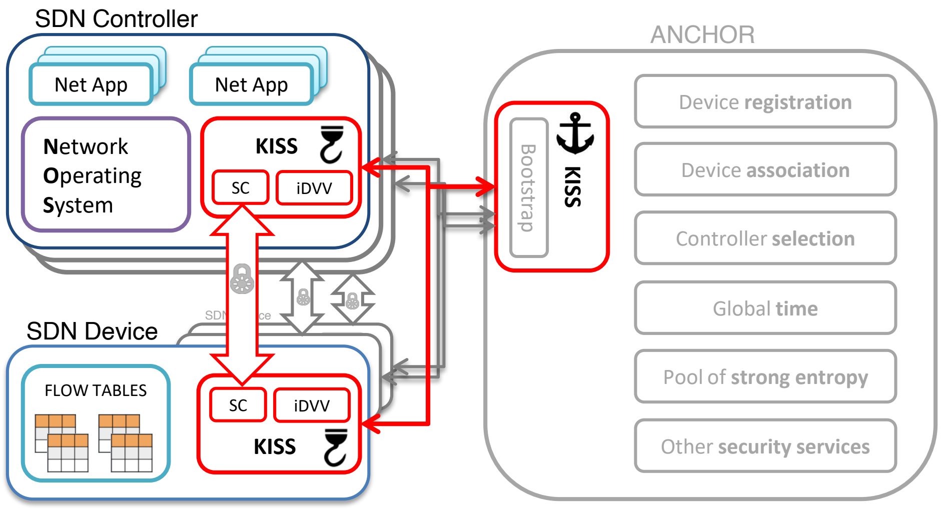

Figure 4.1: KISS framework overview

The two components encapsulated by the KISS boxes are the crucial components of the architecture, and the main subject of our study: a secure channel protocol suite, composed of a judicious choice of state-of-the-art mechanisms and protocols, which we dub Secure Component (SC) for convenience of description, and a novel deterministic but indistinguishable-from-random secret code generation protocol, which we call iDVV (integrated device verification value).

We have considered using TLS implementations (e.g., OpenSSL) as the baseline protocol for SC. However, the experiments in Section 4.3.1 have alerted us to: the sheer performance cost of cryptographic communication; and the further impact of sub-optimal choices of cryptographic primitives. This motivated us to adopt NaCl [BLS12], a high performance yet secure cryptographic library, as the crypto library substrate of SC, complemented by the MAC and hash primitives with best performance – Poly1305 and SHA512 OpenSSL.

The iDVV, a novel component we propose, helps to further enhance the security of SC, through strong crypto material generated at a low cost (e.g., one-time keys, per-message authentication and authorization codes) to be used by NaCl ciphers. The indistinguishability-from-random allied to the determinism allow the safe decentralized generation/verification of per-message keys at both ends of the channel.

4.1.1 System and threat model

For simplicity and without loss of generality, we assume that the controllers and forwarding devices are registered and associated through a secure and robust key distribution service provided by a Key Distribution Center (KDC) or Key Distribution Service (KDS), such as KDCs like Kerberos Key Distribution Center [NT94] and KDSs like ANCHOR registration, association and recovery protocols (see Chapter 5).

The device registration process is by default invoked by network administrators to the KDS, to register new devices. As a result of device registration, the device and the KDS securely share a symmetric key. We denote K_{kc} the shared key between the KDS authority and a registered controller, and K_{kf} the shared key between the KDS authority and a registered forwarding device.

Registered controllers and forwarding devices must be securely associated, also through the KDS authority, as a precondition to communicate securely. The most common case is a forwarding device f_i requesting an association to a controller c_j, through the KDS. After associating, a controller and a forwarding device share two symmetric secrets (of size 256 bits), namely a $seed_{ij}$ and a key_{ij}. The key is generated by the KDS and the seed is generated by the KDS in cooperation with the controller. These secrets will be used to bootstrap the iDVV module, as we discuss ahead.

As a threat model, we consider a Dolev-Yao style attacker [DY83]; [Cer01], who has a complete control of the network, namely the attacker logs all messages, and can arbitrarily delay, drop, reorder, insert, or modify messages. We assume the security of the used cryptographic primitives, including MAC (i.e. Poly1305), hash function (i.e. SHA-512), and symmetric encryption algorithm (e.g., Advanced Encryption Standard (AES)). We will prove the security of the iDVV codes in

Section 4.3. We also assume that the device registration and association services can rely on robust pseudo-random number generators.

4.1.2 Security goals

The main goal of KISS is to provide security properties including authenticity, integrity, and confidentiality for control plane communications, while minimizing cost and complexity.

The secure communication between participants can be easily guaranteed when a secure encryption algorithm is used, as long as the shared secret key is kept secure. A shared key can be leaked by compromise of any of KDS, controller, or forwarding device. Should that happen, it is our goal that a robust SDN system must provide PFS of communications when shared long-term secrets are exposed to an attacker. That is, secrecy of past communications from the time the key became active, to the time it became known to the attacker.

On the devices side, we make no claim about their sheer resilience, since this is largely dependent on vendors. More precisely, when a controller and/or a forwarding device is compromised, we consider that the attacker is able to obtain all knowledge of the victim device(s), including all stored secrets and the session status. However, it is our goal to guarantee the confidentiality of all past communications. We discuss the measures to achieve perfect forward secrecy in Section 4.3.3.

4.2 iDVV: Keep It Simple and Secure

Integrated device verification values are sequentially generated to protect and authenticate requests between two networking devices. The generator is conceived so that its output sequence has the indistinguishability-from-random and determinism properties. In consequence, the same sequence of random-looking secret values is generated on both ends of the channel, allowing the safe decentralized generation/verification of per-message keys at both ends. However, if the seed and key initial values and the state of the generator are kept secret, there is no way an adversary can know, predict or generate an iDVV. In other words, an iDVV is a unique secret value generated by a device F (e.g., a forwarding device), which can be locally verified by another device C (e.g., a controller). The iDVV generation is made flexible to serve the needs of SDN. iDVVs can therefore be generated: (a) on a per message basis; (b) for a sequence of messages; (c) for a specific interval of time; and (d) for one communication session. The main advantages of iDVVs are their low cost and the fact that they can be generated locally, i.e., without having to establish any previous agreement.

4.2.1 iDVV bootstrap

As discussed before, the association between two SDN devices, e.g., forwarding device f_i and controller c_j, happens through the help of KDS, under the protection of the long-term secret keys obtained from registration (K_{kf}, resp. K_{kc}). The outcome of the association protocol is the distribution of two random secrets to both devices: a seed $seed_{ij}$, and an association key key_{ij}. The iDVV mechanism is bootstrapped by installing these two secret values in both the controller and the switch, to animate the iDVV generation algorithms, which we describe next.

Note that the set-up and generation of the iDVV values are performed in a deterministic way, so that they can be done locally at both ends. However, as iDVVs will be used as keys by cryptographic primitives such as MAC or encryption functions, they have to be indistinguishable from random. Hashing primitives are natural choices for our algorithms, since they provide indistinguishable-from-random values if one or more of the input values are known only by the sender and the receiver. This explains why it is crucial that seed and association key are sent encrypted and therefore known only to the communicating devices. Moreover, in order to prevent information leakage, all variables *seed*, *key*, and *idvv* in the following algorithms should have the same length, which we chose to be 256 bits in our design. This length is commonly considered robust, and the evaluation in Section 4.3.4 confirms that. From our experiments reported in Section 4.3.1, the hashing primitive to be used is SHA512, which yields 512 bits, of which we will use the most-significant q bits if we need to reduce the output length to q (as recommended by [Cal+07]). For example, we use the most-significant 256 bits of the SHA512 output as the key for symmetric ciphers.

The initial iDVV value is deterministically created at both ends of the association between two devices[1], by calling function `idvv_init`, which performs hashing on the concatenation of the initial *seed* and *key*, as illustrated by Algorithm 1. After set-up, the generator is ready for first use, as described in the following section.

Algorithm 1: iDVV set-up

```
1: idvv_init()
2:     idvv ← H(seed || key)
```

4.2.2 iDVV generation

After the bootstrap with the initial *idvv* value, the `idvv_next` function is invoked on-demand (again, synchronously at both ends of the channel) to autonomously

[1]For readability, we omit the device-identifying subscripts in the variables.

generate authentication or encryption keys that will be used for securing the communications, as illustrated by Algorithm 2.

The *key* remains the only constant shared secret between the devices. The *seed* evolves to a new indistinguishable-from-random value each time `idvv_next` is invoked to generate a new iDVV. The new seed is the outcome of a hashing primitive H over the current *seed* and current *idvv* (line 2). The *new idvv*, output of function `idvv_next`, is the outcome of a hashing primitive H over the concatenation of the *new seed* and association key *key*.

Algorithm 2: iDVV generation

```
1: idvv_next()
2:     seed ← H(seed || idvv)
3:     idvv ← H(seed || key)
```

4.2.3 iDVV synchronization

The iDVV mechanism is agnostic w.r.t. secure communication protocols, and can be used in a number of ways, in a number of protocols, as a key-per-message or key-per-session, etc. The only key issue about iDVV generation, is to keep it synchronized in both extremes of the channel. So, we discuss recommendations in this regard.

As a generic baseline robustness technique, communication should be authenticated (encrypt-then-MAC recommended), such that any messages failing crypto (decryption or MAC verification), can be simply discarded and that fact handled by existing recovery mechanisms. This brings in robustness against de-synchronization, or malicious attacks, as we show in what follows.

iDVVs can get out of sync for a number of reasons, like speed differences, omission errors, or even DoS attacks. When de-synchronization happens, a baseline technique consists of advancing the iDVV of the "slower" end, to catch up. This lets us introduce another baseline robustness technique: when say, $idvv^k$ is advanced to $idvv^l$ ($k < l$) to re-synchronize, and the operation is not successful (crypto fails), the old $idvv^k$ is restored, and the message motivating the recovery, is discarded.

Suppose an attacker can forge a re-synchronization request to claim that it is in a future state (i.e. with a more advanced iDVV), and fool the recipient to advance its iDVV to catch up: then the attacker is able to play DoS attacks by repeatedly asking all devices to synchronize to an advanced iDVV. This is foiled by the first robustness technique, since the attacker cannot mimic valid crypto, so the message is discarded, and the second robustness technique ensures that the node gets back to the original iDVV state.

Now we discuss some styles of using iDVVs, and possible protocol classes they serve:

Simple iDVV- used as is, works for lock-step, or producer-consumer communication, where the advance is, respectively, either round based, alternatively dictated by each end, or dictated by the producer.

If the channel is unreliable, packet losses may occur, and then the receiver (R) gets out of sync and is not able to verify the next received message from sender S. If the network has a bounded omission degree (maximum number of consecutive omissions), say Od, R can perform a simple recovery process: its iDVV is successively advanced up to $Od + 1$ times, until it is able to verify the incoming message. If the process fails, the message is discarded and the iDVV goes back to the original value (as per the techniques discussed above).

If packet losses can be unexpectedly high, or both ends send competitively and/or in a non-synchronized way, this algorithm is not suitable.

Indexed iDVV- iDVVs are indexed by the generation number. Also, they are operated in "one key per direction" mode, i.e., at each end, one iDVV is generated for each communication direction. This way, they support competitive, non-synchronized correspondents. This mode also supports unreliable, connectionless protocols like User Datagram Protocol (UDP).

Each iDVV generated is indexed by a sequence number (the initial iDVV being $idvv^0$) and the sequence number is included in the message where the respective *idvv* is used. This way, each receiving end (this works in either direction, as we have two pairs of iDVVs) can know the exact *idvv* number that should be used and, for example, detect and recover from omissions, by generating *idvv*'s the necessary number of times to resynchronize. Again, the process is robust: if it fails, the message is discarded and the iDVV goes back to the original value.

Session iDVV- iDVVs now mark sessions, inside which sets of messages are sent that use crypto related to the current session iDVV. It is quite suitable for example, for connection-oriented protocols.

Each $idvv^j$ is valid for the entire session j. A session may be a standard, long-duration session a la Secure Sockets Layer (SSL), or artificially short, rolling session, for higher security, e.g., in a timed (e.g., 1-minute) way. Anyway, at the end of the session and start of the next one, the $idvv^j$ is updated to $idvv^{j+1}$.

Messages pertaining to a session j, labelled (j), may all use the same $idvv^j$ key. However, this can be improved: inside a session, rolling per-message keys may be created, based on $idvv^j$, for example, $k_N = H(idvv^j||N)$, used for message labelled (j, N), the N-th message in the j-th session. Whenever a message with label (j, N) is received, if j is the current session, then the device calculates the key $H(idvv^j||N)$ and decrypts or verifies this message. Again, if the process fails, or j does not match, the message is discarded and the iDVV goes back to the original value.

4.2.4 iDVV implementation and application

iDVVs require minimal resources, which means that they can be implemented on any device, from a simple and very limited smart card to most existing devices. In other words, they are a simple and viable solution that can be embedded in any networking device. Just three values per association have to be securely stored — the seed, the association key and the iDVV itself — in order to use iDVV continuously. Furthermore, only hash functions, simple to implement and with a very small code base, are required to generate iDVVs. Such kind of resource is already available on all networking devices that support traditional network protocols and basic security mechanisms.

Finally, we advocate (and demonstrate in Section 4.3.2) that iDVVs are inexpensive and, as a result, can be used on a per-message basis to secure communication. It is worth emphasizing that, from a security perspective, one fresh iDVV per message makes it much harder for attacks such as key recovery [HP08], advanced side channel attacks [BP10], among other general Hash-based Message Authentication Code (HMAC) attacks [Kim+06], to succeed. In fact, the one-time key approach was initially used for generating MACs. Yet, it was let aside (i.e. replaced by keys with a longer lifetime) due to performance reasons. However, as the iDVV generation has a low cost (see Section 4.3), we incur a lower penalty.

4.3 iDVV Evaluation

In this section, we provide performance and complexity evaluation of the KISS-iDVV infrastructure. Informal proofs of security and correctness proofs of Algorithms 1 and 2 can be found in Appendix A.

4.3.1 On the cost of security

In this section we provide a quantitative analysis of the impact of cryptographic primitives on control plane communication. Although the number of use cases is expanding, SDN has been mainly targeting data centers. As such, SDN controllers have to be capable of dealing with the challenging workloads of these large-scale infrastructures. In these environments new flows[2] can arrive at a given forwarding device every 10 µs, with a great majority of mice traffic lasting less than 100 ms [BAM10]. This means that current data centers need to handle peak

[2] In spite of the fact that there are several definitions of flow in SDN [Kre+15], we equate SDN flow with TCP flow for the sake of simplicity.

loads of tens of millions of new flows/s. The control plane has to meet both the network latencies and throughputs required to sustain these high rates. Existing controllers are capable of achieving a throughput of several million flows/s using TCP [Voe+13]; [Kre+15]; [Sal+16].

So any effort to systematically secure control plane communications has to meet these challenges. In the following we try to put the problem in perspective, by analysing the effect of including even the most basic security primitives to ensure authenticity, confidentiality and integrity when considering peak loads of this magnitude.

We start by analyzing the latency impact of TLS, relative to TCP, and then we focus on hashes and MACs as they are the essential primitives for authenticity and integrity of communication. To measure the latency of control plane communication[3] we used Linux's resource usage system call (`getrusage()`) to get the user CPU execution time. This function is commonly used to measure the performance of cryptographic primitives [VAM15]. Then, we compare the performance of 50+ hashing and MAC primitives, including different implementations such as those provided by OpenSSL (version 1.0.0) and PolarSSL (version 1.3.9), two of the most widely used SSL libraries. We evaluate these primitives using a hardware platform that includes two quad-core Intel Xeon E5620 2.4GHz, with 2x4x256KB L2 / 2x12MB L3 cache, 32GB Dual In-Line Memory Module (DIMM) at 1066MHz, with hyper-threading enabled and overclocking and dynamic CPU frequency scaling disabled. These machines run Ubuntu Server 14.04 Long Term Support (LTS) and were connected via Gigabit Ethernet.

The cost of secure channels

Our first experiments assess the compared average latency of TCP and TLS on control plane communication. We analyse the latency of connection setup and of OpenFlow `PACKET_IN`/`FLOW_MOD` messages. The OpenFlow `PACKET_IN` message is used by switches to send packets to the controller (e.g., when there is no rule matching the packet received in the switch). `FLOW_MOD` messages allow the controller to modify the state of an OpenFlow switch. One of the two nodes of the evaluation platform emulates the controller, whereas the other assumes the role of the forwarding devices. The emulation removes the overhead specific to the controller's implementation, for instance. In practice, there is a huge performance gap among different controllers, most of which is due to the chosen technologies and implementation details. Similarly, the performance of switching devices varies also a lot due to implementation details. To eliminate the implementation-specific performance penalty, we wrote a multi-threaded controller and forwarding devices

[3]Time required to send a `PACKET_IN` message and receive a `FLOW_MOD` message without taking into account any further processing time of the controller.

that just send and receive PACKET_IN and FLOW_MOD messages. This also means that the controller sends FLOW_MOD messages in parallel to the forwarding devices.

The emulated controllers and forwarding devices are implemented in C, using the OpenSSL and PolarSSL TLS implementations in their standard configuration (i.e. no library-specific optimizations were applied). Figures 4.2 and 4.3 show the median of the measured latency over 40k executions. The standard deviation is below 3% so we do not include it in the figures.

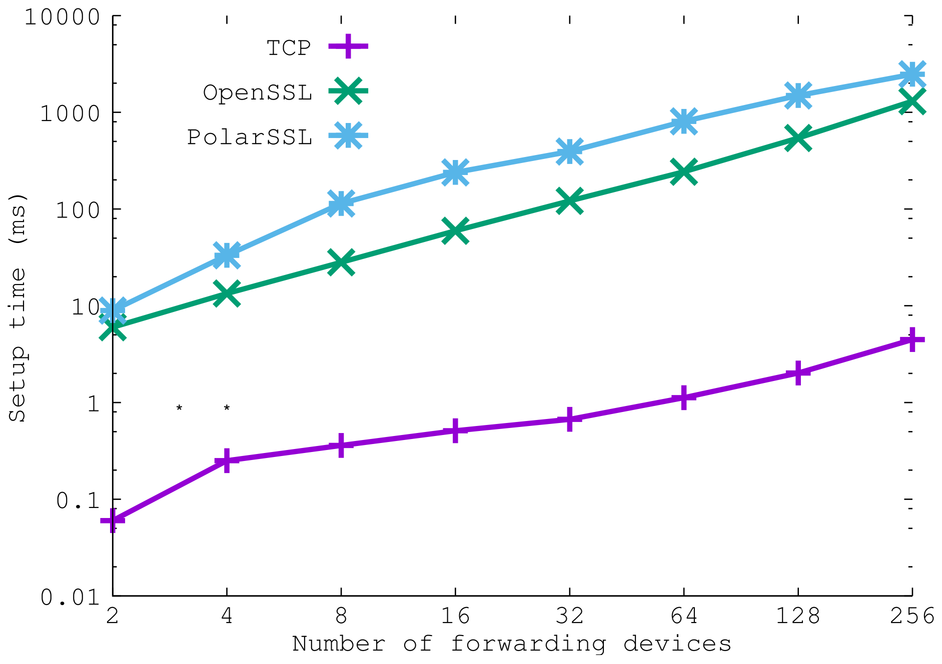

Figure 4.2: TCP and TLS connection setup times (in log scale)

Figure 4.2 shows the connection setup time (per forwarding device). The higher costs of the two TLS implementations are due to the execution of a more elaborate handshake protocol between the devices. While TCP uses a simple three-way handshake, TLS requires a nine message handshake for mutual authentication of the communicating entities. As expected, the overhead increases with the number of forwarding devices. Interestingly, our results also suggest that the choice of implementation has a non-negligible performance impact. For connection setup, PolarSSL induces nearly twice the overhead of OpenSSL.

Although important, a high connection cost can be amortized by maintaining persistent connections. As such, the communications cost is usually considered more relevant. Figure 4.3 shows the latency of FLOW_MOD messages (56 bytes, as specified in OpenFlow 1.4 [ONF13]), averaged over 10k messages. The results with PACKET_IN messages (32 bytes) were similar so we omit them for clarity.

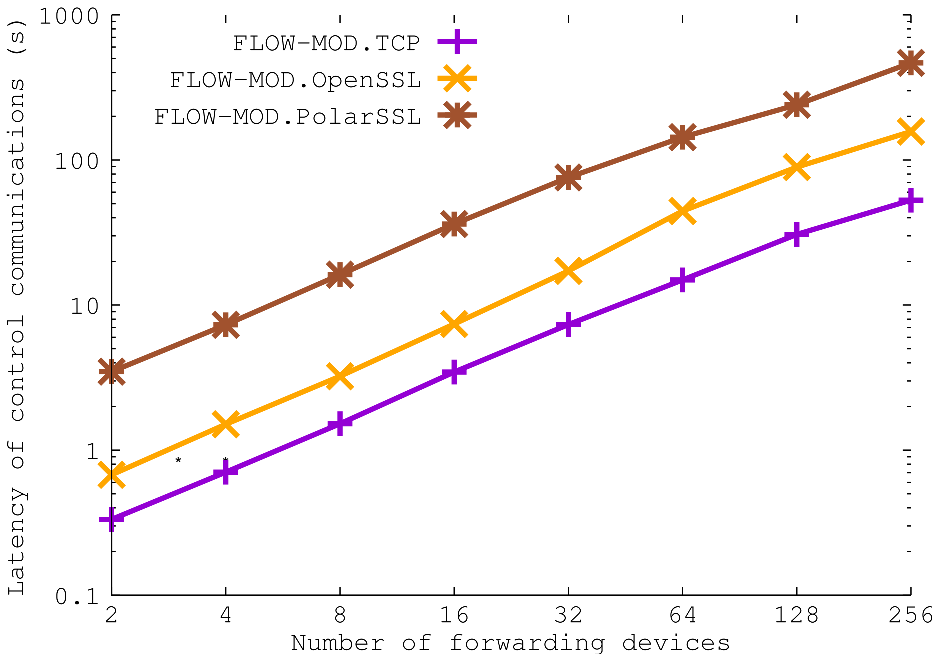

Figure 4.3: `FLOW_MOD` latency (in log scale)

The costs of TCP, OpenSSL and PolarSSL grow nearly linearly with the number of forwarding devices. OpenSSL latency is approximately 3x higher than TCP. This is explained by the high overhead of cryptographic primitives, as we further analyse in the next section. PolarSSL is significantly worse, increasing the latency by up to 7x when compared with TCP.

Conclusions: The main findings of this analysis can be summarised in two points. First, different implementations of TLS present very different performance penalties. Second, the additional computation required by the cryptographic primitives used in TLS leads anyway to a non-negligible performance penalty in the control plane. In consequence, we turn to lightweight cryptographic libraries, such as NaCl [BLS12] and TweetNaCl [Ber+15], which are starting to be used in different applications. NaCl has been designed to be secure and to be embedded in any system [Alm+13], taking a clean slate approach and avoiding most of the pitfalls of other libraries (e.g., OpenSSL – misuse issues). First, it exposes a simple and high-level API, with a reduced set of functions for each operation. Second, it uses high-speed and highly-secure primitives, carefully implemented to avoid side-channel attacks. Third, NaCl is deprived of low-security options and makes conservative choices of cryptographic primitives. One of the most important takeaways of NaCl is its reduced size and its minimal complexity in terms of interface (less error prone) and sub-protocols (if any). Likewise, TweetNaCl (a part of NaCl) was announced as the smallest implementation of a high-security cryptographic library, which makes it a good candidate for inclusion into trusted code bases of computer systems.

A closer look at the cost of cryptography

To understand in more detail the cause of the previous findings we now perform a fine-grained analysis of two main classes of security primitives used in secure channel protocols: hashing and MAC.

To measure the overhead of these primitives we disabled hyper-threading, in order to remove noise and randomness due to the implied resource sharing. As commodity switching devices do not implement direct cache access, we have ensured that the data to be hashed resides in main memory. This avoids artificial performance boosts when operating on cached data[4]. To mimic the behaviour of a switch, we circulated over an input buffer that is twice as large as the last-level cache (L3) to ensure that every read resulted in a cache miss. The numbers in the following graphs represent the median of 1M executions, with a standard deviation below 3%.

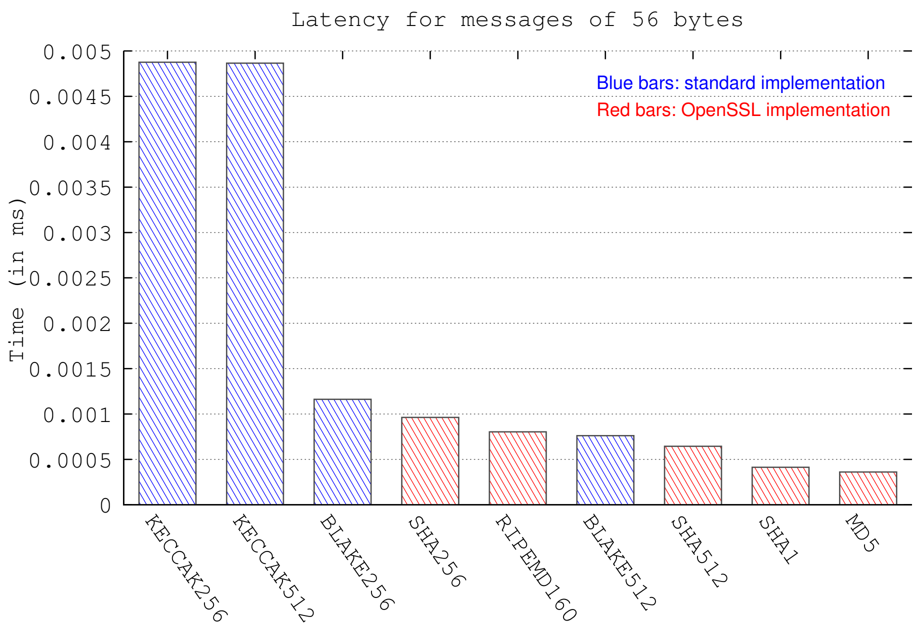

Figure 4.4: Hashing primitives

[4]With cached data, we observed artificial gains of up to 20% for hashing and of 12% for MAC primitives.

We analyze the performance of nine hashing primitives. The results are presented in Figure 4.4. The red bars represent primitives that are provided by OpenSSL, while white bars (BLAKE and KECCAK) indicate the original implementation of primitives that are not part of OpenSSL. From Figure 4.4, we observe that the primitives with smaller digest sizes (SHA-1 and MD5) achieve better performance, as expected. The stronger versions of the SHA and BLAKE families achieve comparable performance (slightly slower), with higher security guarantees. Interestingly, SHA-512 outperforms SHA-256. This behavior is explained by the fact that the former performs 80 rounds of its compression function, over 1024 bits of data at a time, while the latter, though doing only 64 rounds, does them over just 512 bits of data. In the case of KECCAK the difference in performance is due to the additional computational complexity of the mechanisms employed. For instance, this solution requires 24 rounds of permutation on each compression step, while BLAKE requires up to 16 rounds.

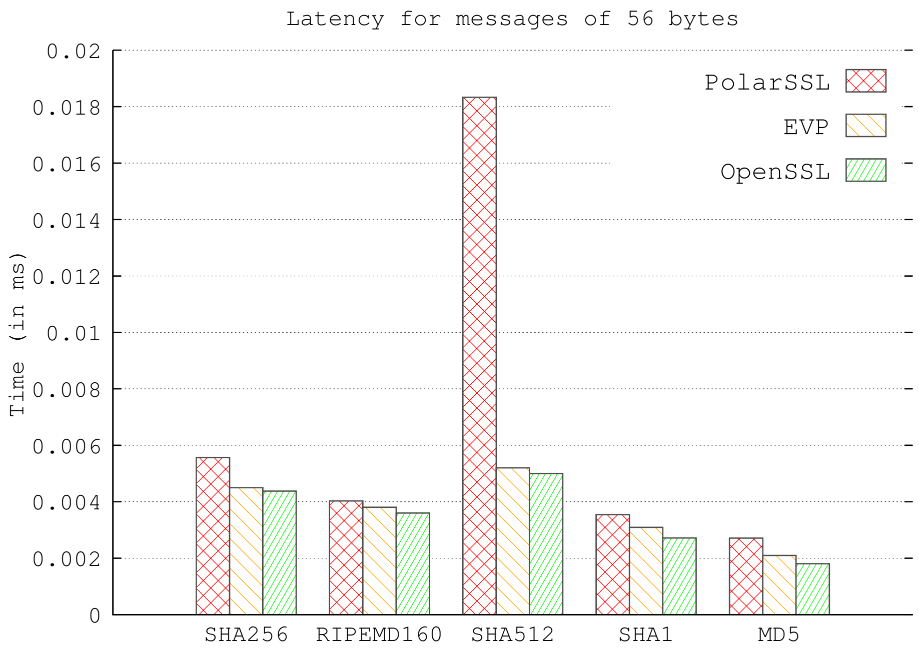

Figure 4.5: Implementations of hashing primitives

To understand the variance between different implementations, we present in Figure 4.5 the costs of the five hashing primitives for which different implementations were available. OpenSSL implementations are the ones showing the best performance for hashing primitives. The PolarSSL implementation always presented higher message latencies. In addition to OpenSSL and PolarSSL, we included EVP (`https://wiki.openssl.org/index.php/EVP`), a library that provides a high-level interface to cryptographic functions. Its main purpose is the ability to replace cryptographic algorithms without having to modify applications. The added flexibility comes at a cost, as we can observe in the results. The same OpenSSL primitives used through an EVP interface experience a penalty between 3% and 15%.

Finally, Figure 4.6 shows the results of the latency analysis of six MAC primitives. It is clear that Poly1305 outperformed all other primitives, being approximately two times faster than OpenSSL's HMAC-SHA1, and close to four times faster than HMAC-SHA512, for instance. For MAC primitives, the choice of specific implementations remains relevant. Curiously, in this case the PolarSSL implementation always outperformed the equivalent OpenSSL implementation. The reason may lie in the fact that OpenSSL does not provide native HMAC implementations, but rather highly configurable HMACs through EVP interfaces. These primitives thus carry the overhead of EVP and the extra costs of configurability.

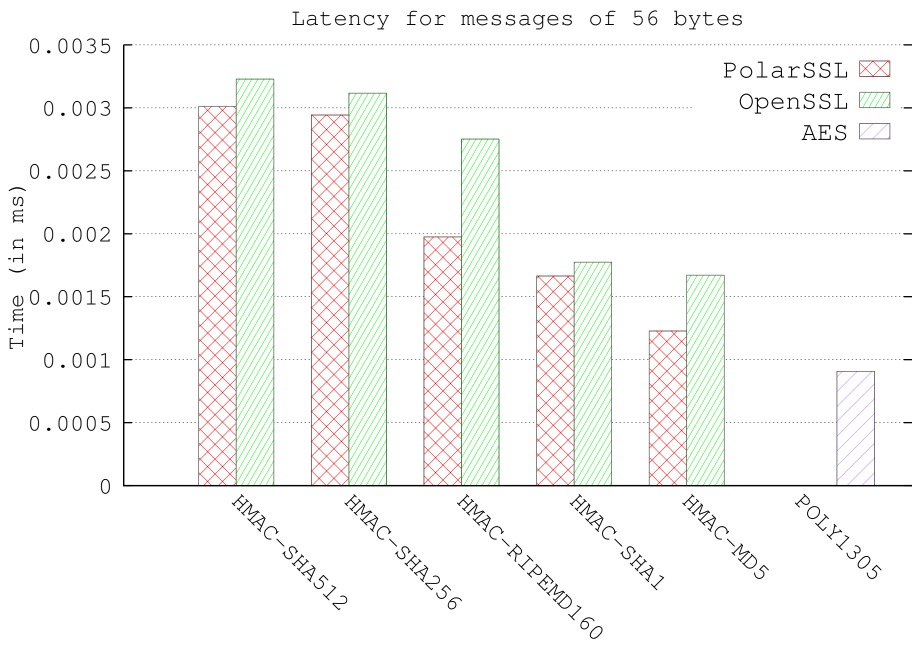

Figure 4.6: MAC primitives

Conclusions: From the results of Figure 4.6, considering the MAC primitive with best performance in the analysis (Poly1305 with 0.001ms per message), around 20 dedicated cores are needed to compute a MAC in order to maintain a rate of 20M flows/s. To understand the importance of judiciously selecting the security primitives implementation, the HMAC-SHA512 OpenSSL (worst case performance in the analysis) would require over three times more cores (up to 65) to compute MACs at these rates. From the hashing primitives analysis, we conclude that SHA-512 performs best, even better than SHA-256. Concerning MAC primitives, the performance of HMAC-SHA512 disappoints, and it is clear that Poly1305 outperformed all other primitives, providing security with high speed and low per-message overhead.

In summary, our findings in this section indicate that (i) the inclusion of cryptographic primitives results in a non-negligible performance impact on the latency and throughput of the control plane; and that (ii) a careful choice of the primitives used and their respective implementations can significantly contribute to reduce this performance penalty and enable feasible solutions in certain scenarios. Taking the outcome of our analysis into consideration, we have selected the NaCl lightweight cryptographic library, and the MAC and hash primitives with best performance – Poly1305 and SHA512 OpenSSL – as the baseline SC secure channel component technologies. Taken together they provide, as per our evaluation, the best trade-off between security and performance for control plane communications in SDN. NaCl is complemented in our architecture with the iDVV mechanism, as we show ahead, to generate crypto material (e.g., keys) used by NaCl ciphers. We evaluate the overall result in the next section.

4.3.2 iDVV Performance Evaluation

Figure 4.7 shows the performance of different primitives for generating cryptographic material. We compare the iDVV generator using SHA512 (iDVV-S5), with an implementation of a common Key Derivation Function (KDF) with different values for the exponent c (128, 64, 32, and 16, respectively), the Diffie-Hellman implementation used by OpenSSL (DH-OSSL), and the `randombytes()` function (NaCl-R) provided by NaCl. The latency of a KDF is very high, increasing linearly with the number of iterations. Our results for DH are compatible with other publicly available performance measurements done on service providers such as Amazon [Mav11], showing a latency several times higher than the iDVV generator. The *randombytes()* primitive of NaCl, used to generate random keys, is the second fastest after iDVV, but still results in a latency at least 2.6x higher. NaCl-R's main latency lies in I/O operations required to read the special random number generator device of the Linux kernel, the `/dev/urandom`. Last, but not least, it is worth emphasizing that NaCl-R cannot be used for the same purposes of iDVVs, since it only generates non-sequential random values, i.e., the values would be different on both ends of the communication channel, defeating our initial purpose.

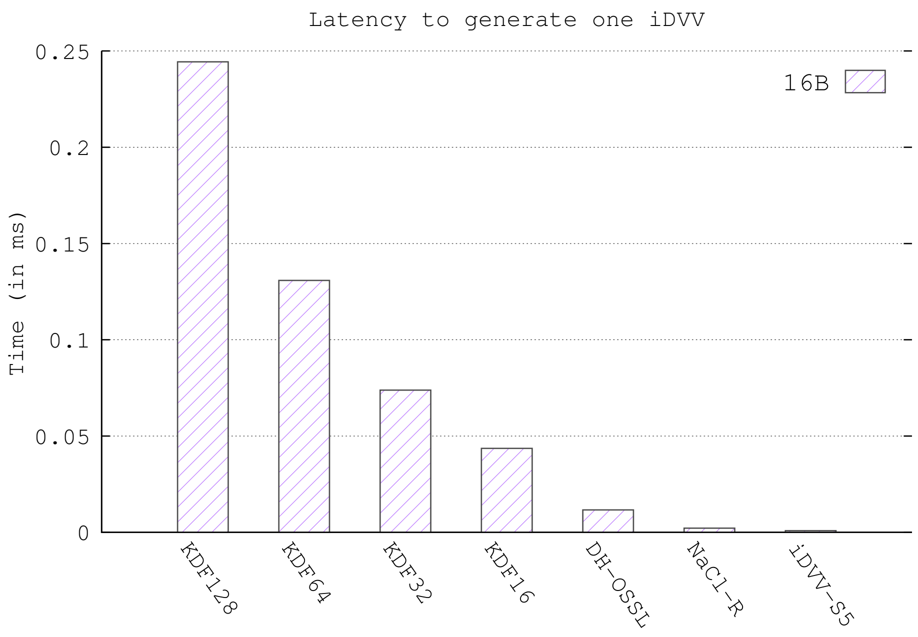

Figure 4.7: Latency to generate keys

4.3.3 Perfect forward secrecy

In this section, we provide a discussion about the perfect forward secrecy properties of our protocols, in face of compromise of any of KDS, controller, forwarding device. We re-state our goal in that case: safeguard secrecy of past communications from the time the key became active, to the time it became known to the attacker.

Note that when the assumed key distribution authority is compromised, then the attacker is able to obtain all the shared secrets K_{kc} (resp. K_{kf}) between the authority and every controller (resp. every forwarding device). In this case, the attacker would be able to decrypt the past communication that delivered the initial *seed* and *key* to the associated devices, and in consequence, decrypt past conversations, since the generation of iDVVs is deterministic from the initial state (see `idvv_init` in Section 4.2.1).

Although providing secure and robust key distribution services is an open challenge and orthogonal to KISS, we provide a simple mechanism for providing PFS even when the authority is compromised. We achieve it by updating the shared key (between the authority and registered devices) each time a forwarding device is associated with a controller. The key is updated as follows: $K_{kc} \leftarrow H(K_{kc})$ and $K_{kf} \leftarrow H(K_{kf})$. This way, a shared key captured cannot decrypt any past messages, since they have been encrypted with previous generations of that key, which have been "forgotten" in the system, given the irreversible nature of hashes.

As far as devices are concerned, when they are compromised, the current values of *seed*, *key* and *idvv* are captured. However, note that *seed* is rolled forward everytime a new iDVV is generated. Only *key* stays as the original secret, but short of having as well the initial *seed* as sent at the end of the association procedure, the attacker will also not be able to synthesize any past iDVVs since day one and so, cannot also decrypt past conversations, achieving PFS, as we sought.

4.3.4 Randomness

We empirically assessed the quality and confidence of the iDVV generator using two techniques. First, we generated more than 200 billion iDVVs to verify if there was any repetition, i.e., the same iDVV generated more than once. There was not a single repeated iDVV. This indicates that our solution is (indeed) suitable for short term iDVVs (e.g., one per message).

Second, pseudorandom generators should be always empirically tested [Bas+10]. Again, we used NIST's test suite [NIS18] to statistically assess the confidence of the iDVV generator. For the sake of our tests, we generated 1M iDVVs of 64 bytes. The file, containing 1M iDVVs, was used as input for the test suite. The streams of bits corresponding to the iDVVs passed all tests, i.e., there was no single failure. This gives us a good level of confidence on the robustness of the iDVV generator.

We also used `ent` [Wal08], which is a pseudorandom number sequence test program, to evaluate the serial correlation coefficient of our implementation. While non-random and predictable sequences of bytes have a serial correlation coefficient of approximately 0.5 and 1.0, respectively, a random byte stream should have a coefficient near to zero. Our implementation, featuring SHA512, had a serial correlation coefficient of 0.0004. Alternative implementations, using MD5 and SHA1, presented the worst case coefficients, as high as 0.035. Typical pseudo-random functions or methods provided by a programming language, such as *rand()* from C and *SecureRandom* from Java, have a serial correlation of approximately 0.0148 and 0.0127, respectively. This shows us that SHA512 is indeed a strong candidate to securely generate iDVVs.

4.4 Discussion

4.4.1 On the cost of the infrastructure

Our proposal compares well with traditional solutions such as EJBCA (`http://www.ejbca.org/`) and OpenSSL, two popular implementations of PKI and TLS, respectively.

The first interesting take away is that our solution has nearly one order of magnitude fewer Lines Of Code (LOC) (85k) and uses four times fewer external libraries and only four programming languages. This makes a huge difference from a security and dependability perspective. For instance, to formally prove more than 717k LOC (OpenSSL + EJBCA) is by itself a tremendous challenge. And it gets considerably worse if we take into account eighty external libraries and eleven programming languages. Moreover, it is worth emphasizing that libraries such as OpenSSL suffer from different fundamental issues such as too many legacy features accumulated over time, too many alternative modes as result of trade-offs made in the standardization, and too much focus on the web and DNS names.

Second, OpenSSL is complex and highly configurable. This leads not only to performance penalties as shown in Section 4.3.1 (e.g., HMAC primitives implemented using EVP), but it has been also the source of many security incidents, i.e., developers and users frequently use the library in an inappropriate way [Ege+13]; [Fah+12]. It has also been shown that the majority of the security incidents are still caused by errors and misconfiguration of systems [Zha+14]; [Raw15]; [Sym16]; [Sch19]; [Hel19]; [SOP20]. Lastly, recent research has uncovered new vulnerabilities in TLS implementations [Beu+15].

In contrast, our proposed architecture exhibits gains in both performance and robustness, contributing to solving the dilemma we outlined in the introduction. By having fewer LOC, we significantly reduce the threat surface – by one order of magnitude – and by combining NaCl and the iDVV mechanism, we provide a potentially equivalent level of security, but quite increased performance/robustness product, as keys can be rolled even on a per message basis.

4.4.2 Size and complexity matter

The more complex the system, the higher the probability of having vulnerabilities and hence a broader attack surface. Nowadays, this is still one of the major problems faced by the technology industry. Specialized security reports have recurrently highlighted the complexity and size of systems as one of the most important security challenges [Cis14]. The time for re-thinking the security of communication channels may have come, and that is also the position we take in proposing the KISS framework.

Renowned cryptographers and security experts have been claiming that simplicity is one of the keys in securing computer systems [BLS12]; [Gre12]; [Sta15]; [Pre15]. In fact, the trusted computing community has been advocating simple interfaces and concerned with the size and complexity of components for a long time [HMK12]; [Raj+11].

These positions have in essence been echoed in our KISS work (starting with the name metaphor, **k**eep **i**t **s**imple, **s**tupid). We methodically selected high performance MAC and hashing primitives for KISS– Poly1305 and SHA512 OpenSSL

– and actually showed the penalty to be paid by less attentive choices. We also turned to lightweight but comparatively secure cryptographic libraries for secure channel support, like NaCl. NaCl was complemented in our architecture with the iDVV mechanism, to generate secrets to be used for example by NaCl ciphers, again in a fast, very simple and decentralized way.

4.4.3 On the cost of iDVV

Similarly to iCVVs, iDVVs are a low overhead solution that requires minimal resources. This solution is thus feasible to be integrated into compute-constrained devices as commodity switches. Our preliminary evaluation has revealed that the iDVV mechanism is faster than traditional solutions, namely, the key-exchange algorithms embedded in the OpenSSL implementation. Considering a setup with 128 switching devices, our results show our proposed solution (iDVV + NaCl's ciphers) to be more than 30% faster than an OpenSSL-based implementation using AES256-SHA (the most common high performance cipher suite, used by IT companies such as Google, Facebook, Microsoft, and Amazon). Importantly, we were able to outperform OpenSSL-based deployments while still providing the same security properties: authenticity, integrity, and confidentiality. In addition, we achieved this result not only while offering the same properties, but also with stronger security guarantees: the tests were made by generating one iDVV*per packet*, while the OpenSSL-based implementation uses a single key (for symmetric ciphering) for the entire communication session.

4.5 Final remarks

In this chapter, we set out to explore and confirm our intuition for the possible reasons behind a slower than expected adoption of security mechanisms in SDN, and based on those findings, we proposed KISS, a modular secure SDN control plane communications architecture.

We started by investigating the impact of essential cryptographic primitives and TLS implementations on the control plane performance. We showed that whilst even the most basic security primitives add a non-negligible degradation of performance, a judicious choice of these primitives and their specific implementations can mitigate the penalty significantly. This is particularly important for the typical SDN scenario that resorts to commodity hardware, sometimes with modest computing capabilities.

The second problem we explored was the complexity of the centralized support infrastructure for authentication and key distribution. We proposed iDVV, a simple and robust decentralized mechanism for generating and verifying the secrets necessary for secure communications between network devices. iDVVs team-up with NaCl, in order to safely replace the cryptographic primitives and key-exchange protocols and key derivation functions commonly used in TLS. As a result, the NaCl-iDVV compound, while achieving the same functional level of security, is simpler, potentially leading to a higher level of implementation robustness by vulnerability reduction. In fact, we estimate the proposed security architecture footprint to be smaller than TLS-PKI alternatives with traditional protocols, by an order of magnitude, in terms of the number of lines of code (LOC). Such a differential also points to reducing the cyclomatic complexity [GK91]; [TK14]; [EC16]. These metrics are used to assess the robustness and estimate verifiability of software systems.

Our results are encouraging in terms of an increase of performance — 30% improvement over OpenSSL — and robustness — an order of magnitude reduction in the number of LOC, and implied cyclomatic complexity. This also means that formal verification is more tractable.

We believe that this is one first step towards lightweight but effective security for control plane communication, and potentially for SDN in general. We make a "call to arms" to foster developments on securing SDN communications without impairing performance, a fundamental precondition for widespread adoption by future SDN deployments.

Chapter 5

ANCHOR: Design and Implementation

In this Chapter we introduce the design, implementation and evaluation of mechanisms and protocols for ANCHOR, including essential security mechanisms such as strong entropy, resilient pseudo-random generators, secure device registration, association and recommendation. We show that compared to the state-of-the-art in SDN security, our solution preserves at least the same security functionality, but achieves higher levels of implementation robustness, by vulnerability reduction, while providing high performance. Whilst we try to prove our point with security, our contribution is generic enough to inspire further research concerning other non-functional properties (such as dependability or quality-of-service).

We have structured the chapter as follows. Section 5.1 describes the enforcement of the logically-centralized security approach that we propose, by populating the ANCHOR architecture with the adequate functionality, and describing the design of its mechanisms and algorithms. Then, in Sections 5.2 and 5.3, we discuss implementation aspects of the architecture, and present evaluation results. In Section 5.4, we do a critique of our choices, discuss some design options of the architecture, and make some final remarks about ANCHOR.

5.1 Enforcing Logically-centralized Security

In this section we present our proposal for the specialization of the ANCHOR architecture for logically-centralized security. Figure 5.1 details the integration of the ANCHOR blocks in the global architecture for logically-centralized security that had been suggested earlier in Chapter 3 (Figure 3.2). In the following explanations, we assume a typical SDN architecture, composed of off-the shelf controllers and forwarding devices, and we further assume that KISS mechanisms are in place (see Chapter 4 for further details).

Our main goal is to provide security properties such as authenticity, integrity, and confidentiality for control plane communication. To achieve this goal, the ANCHOR provides mechanisms and services (e.g., registration, authentication, recommendation, a source of strong entropy, a resilient pseudo-random number generator) required to fulfill some of the major security requirements of SDNs.

As illustrated in Figure 5.1, we "anchor" the enforcement of security properties on ANCHOR. That is, it is a central point for enforcing security policies, which are materialized by means of the above-mentioned mechanisms and services, thereby reducing the burden on controllers and forwarding devices, which just need the local HOOKs, protocol elements that interpret and follow the ANCHOR's instructions. Furthermore, this centralized implementation of crucial mechanisms and protocols, reduces the probability of vulnerabilities in ad-hoc, vendor-specific implementations.

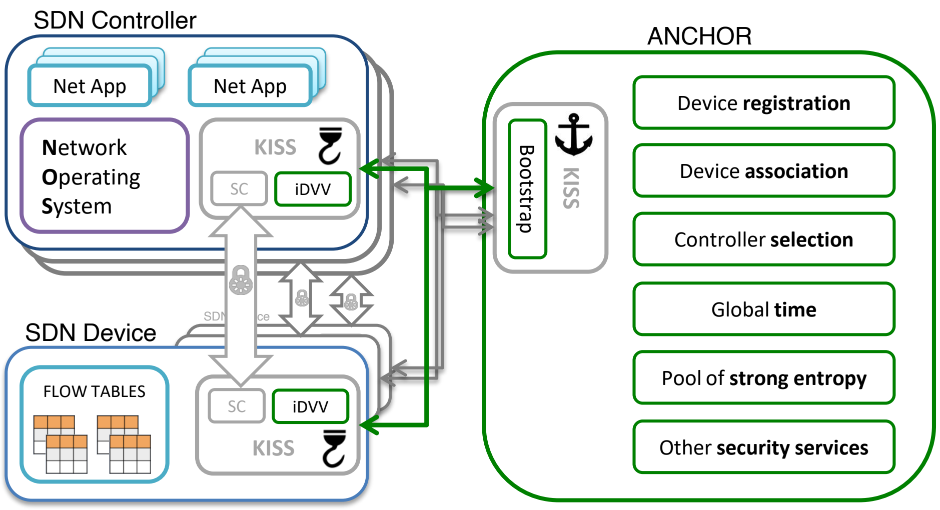

Figure 5.1: ANCHOR: logically-centralized security

Next, we review the components and essential security services provided by ANCHOR.

We propose a source of strong entropy and a resilient pseudo random generator for generating security-sensitive materials. These are crucial components, as attested by the impact of vulnerabilities discovered in the recent past, in sub-optimal implementations of the former in several software packages [BLN16]; [Mim16]; [ZET15]; [Sch12]. We implement and evaluate the robustness of these mechanisms.

After defining system roles and their setup, we present two essential services for secure network operation — device registration and device association. We describe how the above mechanisms interplay with our secure device-to-device communication approach (Section 5.1.8), leveraging the integrated device verification value (iDVV) infrastructure. We implement and evaluate iDVV generators for OpenFlow-enabled control plane communication.

The roster of services of ANCHOR is not closed, and one can think of other functionalities, not described here, including keeping track of forwarding devices association, generating alerts in case of strange behaviors (e.g., recurrent reconnections, connections with multiple controllers), and so forth. These ancillary management tasks are important to keep track of the network operation status.

In what follows, we describe the main ANCHOR services in detail. To help the reader following our descriptions, we summarize the most important notations used in Table 5.1.

Table 5.1: Summary of notations

Notation	**Description**	**Example**
H	Cryptographic hash function	SHA512
MAC	Message Authentication Code algorithm	Poly1305
X, Y	One entity belonging to {A, D$_i$, M, C, F}	Device (e.g., switch) i
Ke$_{XY}$	Encryption secret key. XY denotes: referring to X and Y entities sharing; or the usage the key refers to	256 bits random key
Kh$_{XY}$	MAC/HMAC secret key. XY denotes: referring to X and Y entities sharing; or the usage the key refers to	256 bits random key
E$_{XY}$	Encryption primitive using secret key Ke$_{XY}$	AES
[],HMAC$_{XY}$	keyed-Hash MAC of message [] using secret key Kh$_{XY}$	HMAC-SHA512
KDF	Key Derivation Function	OpenSSL PBKDF2

5.1.1 A source of strong entropy

Entropy still represents a challenge for modern computers because they have been designed to behave deterministically [VH14]. Sources of true randomness (e.g., physical phenomena such as atmospheric noise) can be difficult to use because they work differently from a typical computer.

To avoid the pitfalls of weak sources of entropy, in particular in networking devices, ANCHOR provides a source of strong entropy to ensure the randomness required to generate seeds, pseudorandom values, secrets, among other cryptographic material. The strong source of entropy has the following property:

Strong Entropy - Every value *entropy* returned by *entropy_get* is indistinguishable from random.

Algorithm 3 shows how the external (from other devices) and internal (from the local operating system) sources of entropy are kept updated and used to generate random bytes per function call (*entropy_get()*). The state of the internal and external entropy is initially set by calling the *entropy_setup(data)*. This function requires an input data, which can be a combination of current system time, process number, bytes from special devices, among other things, and random bytes (*rand_bytes()*) from a local (deterministic) source of entropy (e.g., */dev/urandom*) to initialize the state of the entropy generator. As we cannot assume anything regarding the predictability of the input data, we use it in conjunction with a *rand_bytes()* function call (line 2). A call to *rand_bytes()* is assumed to return (by default) 64 bytes of random data.

Algorithm 3: Source of strong entropy

```
1: entropy_setup(data)
2:    e_entropy ← rand_bytes() ⊕ H(data)
3:    i_entropy ← rand_bytes() ⊕ e_entropy
4: entropy_update()
5:    e_entropy ← H($P_i||P_j$) ⊕ i_entropy
6:    e_counter ← 0
7: entropy_get()
8:    if e_counter >= MAX_LONG call entropy_update()
9:    i_entropy ← H(rand_bytes() || e_counter)
10:   entropy ← e_entropy ⊕ i_entropy
```

Function *entropy_update()* uses as input the statistics of external sources and the ANCHOR's own packet arrival rate to update the external entropy. The noise (events) of the external sources of entropy is stored in 32 pools (P_0, P_1, P_2, P_3, ..., P_{31}), as suggested by previous work [FSK11]. Each pool has an event counter, which is reset to zero once the pool is used to update the external entropy. At every update, two different pools of noise (P_i and P_j) are used as input of a hashing function H. The two pools of noise can be randomly selected, for instance. The output of this function is XORed with the internal entropy to generate the new state of the external entropy. It is worth emphasizing that *entropy_update()* is automatically called when *e_counter* (the global event counter) reaches its maximum value and whenever needed, i.e., the user can define when to do the function call.

The resulting 64 bytes of entropy, indistinguishable-from-random bytes (*entropy_get()*), are the outcome of an XOR operation between the external and internal entropy. While the external entropy provides the unpredictability required by strong entropy, the internal source provides a good, yet predictable [VH14], continuous source of entropy. At each time the *entropy_get()* function is called, the internal entropy is updated by using a local source of random data, which is typically provided by a library or by the operating system itself, and the global number of events currently in the 32 pools of noise (*e_counter*). These two values are used as input of a hashing function H.

Such sources of strong entropy can be achieved in practice by combining different sources of noise, such as the unpredictability of network traffic [Gre+09], the unpredictability of idleness of links [Ben+10], packet arrival rate of network controllers, and sources of entropy provided by operating systems. We provide implementation details in § 5.2.1. A discussion about the correctness of Algorithm 3 can be found in §B.2 of Appendix B.

5.1.2 Pseudorandom generator

A source of entropy is necessary but not sufficient. Most cryptographic algorithms are highly vulnerable to weaknesses of random generators [Dod+13]. For instance, nonces generated with weak pseudo-random generators can lead to attacks capable of recovering secret keys. Different security properties need to be ensured when building strong pseudo-random generators, such as resilience, forward security, backward security and recovering security. In particular, the latter was proposed as a measure to recover the internal state of a PRG [Dod+13]. We propose a PRG that uses our source of strong entropy and implements a refresh function to increase its resilience and recovery capability. The pseudo-random generator has the following property:

Robust PRG- Every value *nprd* returned by the function *PRG_next* is indistinguishable from random.

A robust PRG needs three well-defined constructions, namely *setup()*, *refresh()* (or re-seed), and *next()*, as described in Algorithm 4. The internal state of our PRG is represented by three variables, the SEED, the *counter* and the next pseudo-random data *nprd*. The setup process generates a new seed, by using our strong source of entropy, which is used to update the internal state (line 2). In line 3, we initialize the *counter* by calling the *long_uint*() function, which returns a long unsigned int value that will be used to re-seed and to generate the next pseudorandom value. In line 4, we call *entropy_update*() to make sure that the external entropy gets updated before calling one more time the *entropy_get*() function. The first *nprd* is the outcome of an XOR operation between the newly generated seed and a second call to our source of entropy. It is worth emphasizing that the set up of the initial state of the PRG does not require any intervention or interaction with the end user. We provide strong and reliable entropy to set up the initial values of all three variables. This ensures that our PRG is non-sensitive to the initial state. For instance, in a traditional PRG the user could provide an initial seed, or other setup values, that could compromise the quality of the generator's output. The *counter*, which is concatenated with the *nprd* (lines 9 and 13), gives the idea of an unbounded state space [Sta17]. This is possible because we are using cryptographically strong primitives such as a hash function H and the MAC function HMAC. Thus, in theory, we have unbounded state spaces, i.e., we can keep concatenating values to the input of these primitives.

Algorithm 4: Pseudo-random generator

```
1: PRG_setup()
2:     seed ← entropy_get()
3:     counter ← long_uint(entropy_get())
4:     call entropy_update()
5:     nprd ← seed ⊕ entropy_get()
6: PRG_refresh()
7:     seed ← entropy_get()
8:     counter ← long_uint(entropy_get())
9:     nprd ← H(seed ∥ nprd ∥ counter)
10: PRG_next()
11:     counter ← counter - 1
12:     if counter <= 0 call PRG_refresh()
13:     nprd ← HMAC(seed, nprd ∥ counter)
```

The *PRG_refresh()* function updates the internal state, i.e., the SEED, the *counter* and the *nprd*. It uses H to update the state of the *nprd*. Finally, the *PRG_next()* function outputs a new, indistinguishable-from-random stream of bytes, applying HMAC on the internal state. In this function, the *counter* is decremented by one. The idea is for it to start with a very large unsigned 8-bytes value, which is used until it reaches zero. At this point, the *PRG_refresh()* function will be called to update the internal state of the generator. The newly generated *nprd* is the outcome of an HMAC function with a dimension of 128 bits.

The main motivation for having a PRG along with a strong source of entropy is speed. Studies have shown that entropy generation can be rather slow, such as 1.5 s to 2 min for generating 128 bits of entropy [MDP15]. Our source of entropy uses external entropy and random bytes from special devices, whereas the PRG uses an HMAC function, in order to have a fast and reliable generation of pseudo-random values.

In spite of the fact that we could use any good PRG to generate cryptographic material (e.g., keys, nonce), it is worth emphasizing that we introduce a PRG that works in a seamless way with our strong source of entropy, improving its quality. In Section 5.2.2, we discuss the specifics of the implementation. We also evaluate the robustness and level of confidence of our algorithms in Section 5.3.1. A discussion about the correctness of Algorithm 4 can be found in §B.3 of Appendix B.

5.1.3 System deployment

Currently, ANCHOR is designed to work in a single domain, with single ownership, such as a data center, enterprise, or university campus network. ANCHOR supports deployments with multiple controller instances [Kop+10], for scalability and availability of network control, as is required in production systems [Jai+13]. It is worth emphasizing it is part of our plan to extend ANCHOR's features and services to multiple domains with multiple ownership.

ANCHOR is designed to logically centralize non-functional properties of *generic* SDN deployments. As such, it is not restricted to OpenFlow. Other southbound APIs can be used, such as Protocol Oblivious Forwarding (POF), Forwarding and Control Element Separation (ForCES), or Programming Protocol-independent Packet Processors (P4). The ANCHOR connectivity infrastructure, used for communication between SDN devices (controllers and networking gear) and ANCHOR, can use traditional in-band or out-of-band mechanisms (for instance, traditional routing protocols such as Open Shortest Path First (OSPF) or Intermediate System-to-Intermediate System (IS-IS), as is common for control plane channels [Kop+10]).

In our system we assume the existence of management personnel with two different roles: the system Administrator, who controls the operation of central services such as ANCHOR, and the network Manager (a.k.a. network administrator), responsible for the operation of network devices. Every time a new network device (a forwarding device or a controller) is initialized and added to the network, it must first be registered, before being able to operate.

In the current practice, the device registration is a manual process triggered by a network administrator through an out-of-band channel or initiated by a neighbor proxy device using SNBI and 802.1AR credentials, as is the case of ODL [Ope]; [Sco17]. Given the potentially large number of network devices in SDN, such a manual process is unsatisfactory: slow and labour-intensive, and error-prone. Thus, we propose a suite of protocols, described ahead, to fulfill the desire for a robust semi-automated setup of the system.

The ANCHOR server is first set up by the system administrator. Then, a batch setup protocol initiated by ANCHOR ensues, to initialize the managers (Section 5.1.4). Before proceeding, encryption and MAC keys are installed into the new network devices each manager subtends, through some out-of-band channel under its control. Then the procedure is automatically concluded, with a device registration protocol, described in Section 5.1.5, whereby devices can be registered automatically. The network is now about ready for operation.

As a first move to bring security and dependability into control plane operation, we foresee that no two devices are allowed to communicate without being associated. This can be done at any time (through the mediation of ANCHOR) while the system is in service, and it is implemented by a device association protocol, described in Section 5.1.6.

For simplicity and without loss of generality, in what follows we denote the IDentification (ID)s of *A*NCHOR server, *D*evice, network *M*anager, *C*ontroller, and *F*orwarding device, as respectively $\{A, D, M, C, F\}$. We denote as well $\mathrm{E}_{XY}()$ an encryption using encryption key Ke_{XY}, and we omit decryption operations not to clutter the algorithm descriptions: whenever an entity in possession of key Ke_{XY} receives E_{XY}(payload), we assume the cleartext 'payload' elements are readily available in the next algorithm lines. Likewise, we denote [],HMAC_{XY}, respectively, a message field inside [], followed by an HMAC over the whole material within [], using MAC key Kh_{XY}, where $X, Y \in \{A, D_i, M, C, F\}$. We rely on ANCHOR to generate strong keys for the system participants — and eventually distribute them to managers and devices — using a suitable KDF based on the high entropy random material described in the previous sections.

We now define some functions used in the algorithms:

- `generates(data)` - generates the cryptographic material specified in 'data', using the strong crypto tooling provided in ANCHOR;
- `destroys(data)` - deletes the items specified in 'data' from the server;
- `offline(data)` - safeguards (copies, i.e., these items stay online) the items specified in 'data' to some out-of-band storage medium (e.g., a Universal Serial Bus (USB) stick, smart card, etc.);
- `installs(data)` - brings the items specified in 'data' back online to the concerned machine, from some out-of-band storage medium where they had been safeguarded.

5.1.4 Batch setup

Algorithm 5 describes the initial batch setup of ANCHOR, network managers and associated devices.

ANCHOR *setup.* In preparation for the optional provision of PCS, which will be discussed in Chapter 6, two master recovery keys are created for ANCHOR, namely the master recovery encryption key $Ke_{A_{rec}}$ and master recovery MAC key $Kh_{A_{rec}}$. As we will present later, the master recovery keys are only used in three cases, namely (a) upon the initial setup, described below; (b) when a new network administrator is registered with ANCHOR; and (c) when ANCHOR was compromised and is reinstated into a trustworthy state (i.e., the semi-automated or fully-automatic post-compromise recovery processes presented in Chapter 6). In consequence, these two master recovery keys should not be stored regularly in ANCHOR, if they are to recover from a possible full server compromise. We refer the reader to §B.1 of Appendix B for more information (including a visual representation) regarding the three phases of ANCHOR, namely setup, normal operation, and recovery.

So the first steps in Algorithm 5 (lines s_a and s_b) are a prefix of the main protocol execution. The system administrador S (controlling ANCHOR) initiates it. Then, in an offline operation, generates $Ke_{A_{rec}}$ and $Kh_{A_{rec}}$. These keys will then come online in the ANCHOR server, but will only be present for short moments, for the actions defined above, (a),(b),(c)[1].

Lines 1-5 illustrate the first actions of A (ANCHOR), to run the batch setup protocol (initiating all managers). These steps are also the ones used to later introduce a new manager. All steps are recursed for each M. Line 1, as we had suggested, copies $Ke_{A_{rec}}$ and $Kh_{A_{rec}}$ to the server, from offline (either in the sequel of offline boot, or later from secure storage). Then (l.2) the manager recovery key pair (encryption and MAC) are computed, and (l.3) the respective shared secret key pair for secure communication between A and M is generated (Ke_{AM}, Kh_{AM}). All this material is safeguarded offline (l.4), and made available to the entity managing M, for later use online, as we explain ahead. In line 5, all recovery keys are destroyed in the server.

Manager setup. Each network manager (its ID denoted M in the protocols) is registered with ANCHOR manually, by installing the shared encryption key Ke_{AM} and MAC key Kh_{AM} from offline (l.6). This is the *only manual process* to initialize a new M. Initialization proceeds with line 7, where M seeks to batch initialize all

[1]Just to give a real feel, one possible implementation of this principle is: a pristine ANCHOR server image is created; it boots offline in single user mode; it generates $Ke_{A_{rec}}$ and $Kh_{A_{rec}}$ through a strong KDF as discussed above; keys are written into a USB device, and then deleted from the server; first online boot proceeds; keys are imported from the USB whenever needed, and then deleted again.

Algorithm 5: Batch setup of ANCHOR, M, and initial D_1 to D_n.

s_a.	S (offline)	initiates ANCHOR
s_b.	S (offline)	generates($Ke_{A_{rec}}$, $Kh_{A_{rec}}$)
	{For each manager with ID M and its associated devices with ID $\{D_i\}_{i=1}^{n}$}	
1.	A	installs($Ke_{A_{rec}}$, $Kh_{A_{rec}}$)
2.		$Ke_{AM_{rec}}$=H($Ke_{A_{rec}}$ || M); $Kh_{AM_{rec}}$=H($Kh_{A_{rec}}$ || M);
3.		generates(Ke_{AM}, Kh_{AM})
4.		offline(Ke_{AM}, Kh_{AM}, $Ke_{AM_{rec}}$, $Kh_{AM_{rec}}$)
5.		destroys($Ke_{AM_{rec}}$, $Kh_{AM_{rec}}$, $Ke_{A_{rec}}$, $Kh_{A_{rec}}$)
6.	M	installs(Ke_{AM}, Kh_{AM})
7.	M $\rightarrow$ A	[InitD, M, $E_{AM}(\{D_i,x_m^i\}_{i=1}^{n})$)], $HMAC_{AM}$
8.	A	for each D_i, generates(Ke_{MD_i}, Kh_{MD_i}, x_a^i)
9.	A $\rightarrow$ M	[InitD, M, $E_{AM}(\{D_i,x_m^i,x_a^i,Ke_{MD_i}, Kh_{MD_i}\}_{i=1}^{n})$)], $HMAC_{AM}$
10.	M	$\forall i \in [1,n]$ offline(Ke_{MD_i}, Kh_{MD_i})
		{For each device $\{D_i\}_{i=1}^{n}$}
11.	D_i	installs(Ke_{MD_i}, Kh_{MD_i})

devices it subtends, by sending an 'InitD' message to A.

Device setup. A device with identity D_i is either a forwarding device (F) or a controller (C), but we do not differentiate them during the set up and registration processes. The first operation to be made after a device is first brought to the system is the initialization or setup, which concerns the establishment of credentials, for secure management access by the network administrator.

M sends an initialize device(s) (`InitD`) message to A, with the list of identifiers (D_i) of those devices. ANCHOR replies to the (`InitD`) message, by generating and sending back to M (lines 8-9) pairs of keys Ke_{MD_i} and Kh_{MD_i} for M to communicate with each device. This message exchange is protected against confidentiality, authentication and replay attacks, by encryption, MACs and nonces (x_m^i,x_a^i). M safeguards this key pair offline (l.10), and the keys are installed through out-of-band methods in each D_i subtended by this M (l.11). This is the *only manual process* to initialize a new device D. Afterwards, all devices managed by each M can be registered with ANCHOR through our device registration protocol, described next.

5.1.5 Device registration

The device registration protocol is presented in Algorithm 6. We assume that Ke_{MD_i} and Kh_{MD_i}, described above, are already in place.

Algorithm 6: Device registration

```
                {Bootstrap for devices $D_1 - D_n$ }
1.  $M \to A$     [Reg, M, $E_{AM}(\{D_i, x^i_m\}^n_{i=1})$],$HMAC_{AM}$
2.  A             $\forall i \in [1, n]$ generates($Ke_{AD_i}$, $Kh_{AD_i}$, $x^i_a$)
3.  $A \to M$     [Reg, M, $E_{AM}(\{D_i, x^i_m, x^i_a, Ke_{AD_i}, Kh_{AD_i}\}^n_{i=1})$],$HMAC_{AM}$
4.  M             installs($Ke_{AM_{rec}}$, $Kh_{AM_{rec}}$).
                {For each device $D_i$}
5.  M             $Ke_{AD_{i_{rec}}}$=H($Ke_{AM_{rec}}$||$D_i$); $Kh_{AD_{i_{rec}}}$ =H($Kh_{AM_{rec}}$||$D_i$)
6.  $M \to D_i$   [Reg, $E_{MD_i}(x^i_a, Ke_{AD_i}, Kh_{AD_i}, Ke_{AD_{i_{rec}}}, Kh_{AD_{i_{rec}}})$],$HMAC_{MD_i}$
7.  $D_i \to A$   [M, $D_i$, $E_{MD_i}(x^i_a)$],$HMAC_{MD_i}$
8.  $A \to M$     [M, $D_i$, $E_{AM}(x^i_a)$],$HMAC_{AM}$
9.  A             tag($D_i$) = registered;
10.               for t $\in \{C, F\}$, if Type($D_i$) == t, then tList = tList $\cup$ {$D_i$}
11.               $\forall i \in [1, n]$, if tag($D_i$) == registered is True
12.                   $Ke_{AM}$ = H($Ke_{AM}$); $Kh_{AM}$ = H($Kh_{AM}$).
13. $M \to D_i$   [$D_i$, $E_{MD_i}(x^i_a)$],$HMAC_{MD_i}$
14. $D_i$         $Ke_{MD_i}$ = H($Ke_{MD_i}$); $Kh_{MD_i}$ = H($Kh_{MD_i}$).
15. M             tag($D_i$) = registered;
16.               destroys($Ke_{AM_{rec}}$, $Kh_{AM_{rec}}$, $Ke_{AD_i}$, $Kh_{AD_i}$, $Ke_{AD_{i_{rec}}}$, $Kh_{AD_{i_{rec}}}$);
17.               $Ke_{MD_i}$ = H($Ke_{MD_i}$); $Kh_{MD_i}$ = H($Kh_{MD_i}$);
18.               $\forall i \in [1, n]$, if tag($D_i$) == registered is True
19.                   $Ke_{AM}$ = H($Ke_{AM}$); $Kh_{AM}$ = H($Kh_{AM}$).
```

The first part concerns the bootstrap of the registration of a batch of devices with ANCHOR (A), by a network administrator M. Let $\{D_i\}^n_{i=1}$ be the set of n device identities that the administrator wants to register. M requests (line 1) the registration to A, by sending it a 'Reg' message, accompanying each D_i ID with a nonce x^i_m. A generates random nonce x^i_a and keys Ke_{AD_i}, Kh_{AD_i}, for each D_i to communicate with A, and returns them encrypted to M (lines 2,3). This message exchange is protected against confidentiality, authentication and replay attacks, by encryption, MACs and nonces.

The process then follows for each device D_i. In preparation, M starts by installing its recovery key pair (line 4), $Ke_{AM_{rec}}$ and $Kh_{AM_{rec}}$. Those keys are used to compute the device recovery key pair (l.5). Then M sends D_i the relevant cryptographic keys (l.6). Device D_i follows-up confirmation to A, which closes the loop with M, using the original nonce from A (lines 7,8).

A then performs a set of operations (lines 9-12) to commit the registration of D_i, namely by inserting it into the controller or forwarding device list, respectively CList or FList, and updating (rolling) the A-M communication keys. Continuing, in line 13, M closes the loop with D_i, using the original nonce from A, finally confirming D_i's registration. In response, D_i rolls the M-D communication keys (l.14). It is now time for M to execute the final commit operations with regard to the registration of D_i (l.15). In line 16, all recovery keys are destroyed, as well as the A-D communication keys Ke_{AD_i}, Kh_{AD_i} — for which M was a mere mediator. M synchronizes with A and with D_i, by updating (rolling) the respective shared communication keys as well (lines 17-19).

The reader will note that in Algorithm 6, the update of several shared keys (i.e., lines 12, 14, 17 and 19) concerns the achievement of PFS. When a key is updated, the old one is destroyed. Likewise, the process of generation of the recovery keys hierarchy from the earliest setup stages is conducent to achieving PCS, discussed in detail in Chapter 6.

5.1.6 Device association

The association service is required for authorizing control plane channels between any two devices, such as a forwarding device and a controller. A forwarding device has to request an association with a controller it wishes to communicate with. This association is mediated by the ANCHOR. The association process between two devices is performed by the sequence of steps detailed in Algorithm 7. Registered controllers and forwarding devices have been inserted in CList and FList, respectively.

Notation: As previously explained, the registration process has set in place shared secret key pairs between ANCHOR A and all registered devices, generically denoted Ke_{AD_i}, Kh_{AD_i}. For simplicity, and without loss of generality, in the following algorithm, $E_D()$ or [],$HMAC_D$, with D standing for 'C' or 'F', denote crypto operations using those A-D shared keys, identifying one of the registered controllers C or forwarding devices F.

The device association implemented by Algorithm 7 has the following properties:

Controller Authorization - Any device F can only associate to a controller C authorized by the ANCHOR.

Algorithm 7: Device association

{Of forwarding device F with controller C}

1.	F $\rightarrow$ A	[x_{fa}, F, GetCList],HMAC$_F$
2.	A $\rightarrow$ F	[x_{fa}, F, E$_F$(CList(F), x_{fa})],HMAC$_F$
3.	F $\rightarrow$ C	[x_{fa}, GetAiD, F, C, E$_F$(GetAiD, F, C, x_{fc}, x_{fa})]
4.	C $\rightarrow$ A	[x_{fa}, GetAiD, F, C, E$_F$(GetAiD, F, C, x_{fc}, x_{fa}), E$_C$(GetAiD, F, C, x_{ca}, x_{fa})],HMAC$_C$
5.	A $\rightarrow$ C	[x_{fa}, E$_F$(x_{fc}, AiD), E$_C$(x_{ca}, AiD)],HMAC$_C$
6.	A	destroys (AiD)
7.	C $\rightarrow$ F	[x_{fa}, E$_F$(x_{fc}, AiD), E$_{AiD}$(SEED, x_{fa})]
8.	F $\rightarrow$ C	[x_{fa}, E$_{AiD}$(SEED $\oplus$ x_{fa})]
9.	A, F	Ke$_{AF}$ = H(Ke$_{AF}$); Kh$_{AF}$ = H(Kh$_{AF}$)
10.	A, C	Ke$_{AC}$ = H(Ke$_{AC}$); Kh$_{AC}$ = H(Kh$_{AC}$)

Device Authorization - Any device F can associate to some controller, only if F is authorized by the ANCHOR.

Association ID Secrecy - After termination of the algorithm, the association ID (AiD) is only known to F and C.

Seed Secrecy - After termination of the algorithm, the seed (SEED) is only known to F and C.

The algorithm coarse structure follows the line of the Needham-Schroeder (NS) original authentication and key distribution algorithm [NS78], but contemplates anti-replay measures such as including participant IDs, and a global initial nonce as suggested in [OR87]. Unlike NS, it uses encrypt-then-mac to further prevent impersonation. Furthermore, it is specialized for device association, managing authorization lists, and distributing a double secret in the end (association ID and seed). Secure communication protocols running after association can, as explained in Section 5.1.8, use iDVVs on a key-per-message or key-per-session basis, rolling from the initial seed and secret association ID.

The association process starts with a forwarding device sending an association request to the ANCHOR (line 1 in Algorithm 7). This request contains a nonce x_{fa}, the identification of the device F and the operation request GetCList (get list of controllers). The request also contains an HMAC to avoid device impersonation attacks. The ANCHOR checks if F is in FList (registered devices), and if so, it replies (line 2) with a list of controllers (CList(F)) which F is authorized to associate with. The list of controllers (and the nonce x_{fa}) is encrypted using a key (set up during registration) shared between A and F. This protects the confidentiality of the list of controllers, and x_{fa} ensures that the message is fresh, providing protection against replay attacks. A message authentication code also protects the integrity of the ANCHOR's reply, avoiding impersonation attacks. Next, F sends an association request to the chosen controller C (line 3). The request contains a message that is encrypted using a key shared between F and A. This message contains the get association id (GetAiD) request, the identity of the principals involved (F,C), a nonce x_{fc}, and binds to the nonce x_{fa}. The controller forwards this message to A (line 4), adding its own encrypted association request field, similar to F's, but containing C's own nonce x_{ca} instead. This prevents the impersonation of the controller since only it would be able to encrypt the freshly generated x_{fa}. In line 5, C trusts that A's reply is fresh because it contains x_{fa}. The controller also trusts that it is genuine (from A) because it contains x_{ca}. As such, C endorses F as an authorized device and AiD as the association key for F. Future compromise of A should not represent any threat to established communication between C and F. To achieve this goal, A immediately destroys the AiD (line 6) and C and F further share a seed not known by A (line 7).

C forwards both the encrypted AiD message and its seed to F (line 7). The forwarding device trusts that this message is fresh and correct because it contains x_{fa}, and x_{fc} under encryption, together with the AiD, only known to F and C, which it endorses then as the association key. F trusts that C is the correct correspondent, otherwise A would not have advanced to step 5. That being true, future interactions will use AiD. F believes that the SEED is genuine, as random entropy for future interactions, because it is encapsulated by AiD, known only to C and F. The forwarding device also trusts that the message is fresh because it contains x_{fa}. Finally (line 8), C trusts it is associated with F (as identified in step 3 and confirmed by A in step 5), when F replies showing it knows both the AiD and the SEED, by encrypting the SEED XOR'ed with the current nonce x_{fa}, with AiD.

Seeking PFS, replay and impersonation attacks are avoided because all encrypted interactions are dependent on nonces, so will become void in the future. Likewise, at the end of each device association protocol, all keys shared between a device (F or C) and ANCHOR will be updated (rolled) to the hash value of this key (lines 9, 10). All nonces are indistinguishable from random, i.e., not predictable.

A discussion of the correctness of Algorithm 7 can be found in §B.4 of Appendix B.

5.1.7 Controller recommendation

Similarly to moving target defense strategies [Wan+14], devices (e.g., controllers) are hidden by default, i.e., only registered and authenticated devices can get information about other devices. Even if a forwarding device finds out the IP of a controller, it will not be able to establish a connection with the controller unless it is registered and authorized by the ANCHOR beforehand.

Controllers can be recommended to forwarding devices using different parameters, such as latency, load, or trustworthiness. When a forwarding device requests an association with one or more controllers, the ANCHOR sends back a list of authorized controllers to connect with. The forwarding device will be restricted to associate itself with any of the controllers on the list. As a result, forwarding devices will not be allowed to establish connections with other (e.g., hostile or fake) controllers, and fake forwarding devices will be, by default, forbidden to set up communication channels with non-compromised controllers.

5.1.8 Device-to-device communication

Communication between devices is supported by the iDVV protocol. As explained in Chapter 4, an iDVV is a unique value generated by device F (e.g., forwarding device) which can be verified by device C (e.g., controller). An iDVV generator has essentially two interfaces. First, *idvv_setup (seed, secret)*, which is used to set up the generator. It receives as input two secret, random and unique values, the seed and the (higher-level protocol dependent) secret. The source of strong entropy and the robust PRG are, amongst other things, used to bootstrap and keep the iDVV generators fresh. Second, the *idvv_next()* interface returns the next iDVV. This interface can be called as many times as needed. Starting with the same seed and secret, the iDVV generator will generate, for example, at both ends of a controller-device association, the exact same sequence of values. In other words, it is a deterministic generator of authentication or authorization codes, or one-time keys, which are, however, indistinguishable from random. The importance of this property is explained below.

Communication between any two devices happens only after a successful association. Consider the end of an association establishment, as per Algorithm 7, e.g., between a controller C and a forwarding device F: at this point, both sides, and only them, have the secret and unique material $(SEED, AiD)$. Using them, they can bootstrap the iDVV protocol, which from now on can be used at will by any secure communication primitives. As explained earlier, and detailed in [Kre+17], iDVV generation is flexible and low cost, to allow the use: (a) on a per message basis; (b) for a sequence of messages; (c) for a specific interval of time; or (d) for one communication session.

NaCl [BLS12], as mentioned in previous sections, is a simple, efficient, and secure alternative to OpenSSL-like implementations, and is thus our choice for secure communication amongst devices. Indeed, researchers have shown that it is resistant to side channel attacks [Alm+13] and that its implementation is robust [BLS12]. Different from other cryptographic libraries, NaCL's API and implementation is kept very simple, justifying its robustness. Through ANCHOR, the SDN communication channels are securely encrypted using symmetric key ciphers provided by NaCl, with the strong cryptographic material required by the ciphers generated by our logically centralized security mechanisms, allowing secret codes per packet, session, time interval, or pre-defined ranges of packets.

5.2 Implementation

A prototype of ANCHOR has been implemented as envisioned in Figure 5.1. To strengthen our confidence in the effectiveness of deployment of ANCHOR in a production environment, we have implemented two versions of the system. The first uses the POX controller and CBench[2] (OpenFlow switches emulator). This version has approximately 2k lines of Python code and 700 lines of C code (integration with CBench). It uses Google's protobuf (`https://developers.google.com/protocol-buffers/`) for defining the protocols and efficiently serializing the data. The second is a slightly modified version using the Ryu controller and Open vSwitch. In this section, we give an overview of the most relevant implementation details. The evaluation of the different components of the architecture is presented in Section 5.3.

5.2.1 Source of strong entropy

We have 32 pools of events fed by four different sources, (1) incoming packet rate sent by controllers; (2) incoming packet rate of ANCHOR; (3) network statistics of forwarding devices; and (4) random bytes from local systems. Each of the sources feeds the pools in its own way. Sources (1) and (3) use a round-robin approach, whereas sources (2) and (4) randomly select the next pool to put the new event in. In this way, we have a diversity of approaches for feeding the pools of noise, making the "guessing task" of an attacker harder. Each pool needs to store only the digest of the SHA512 hashing function. The current digest and the newly arrived events are used as input of the hashing function. Lastly, once the pool has been used by the source of strong entropy, it is reset to a new initial state, which consists of the digest of a hash function using as input random bytes of a local entropy source such as */dev/urandom*.

To implement the *entropy_update()* function (see Algorithm 3), we can use the pools of noise circularly (e.g., P_0 and P_1, P_2 and P_3, and so forth), in a combined circular and random way (P_0 and P_7, P_1 and P_{31}, and so forth), or in a completely random fashion. The number of pools (32) and this diversity of approaches for

[2]CBench is the default and most widely used tool for benchmarking control plane performance [KAI14]; [ZIR15].

using the pools make it very hard for an attacker to enumerate the possible values for the events used to update the generator's internal state [FSK11].

Even if an attacker is controlling two or more external sources in a timely manner, it will be hard to guess the new state of the external entropy. First, the attacker needs to enumerate the events of the pools being used on each update. This, by itself, is something hard to achieve since the attacker does not know the update sequence of these pools, i.e., which external sources are being used, in which sequence, to update each pool. In other words, he/she would have to know all sources of noise, and the sequence in which they are being used to update the pools. It is also worth emphasizing that the external sources need to have a predefined maximum rate for sending the heartbeats, i.e., compromised sources cannot send data at a higher frequency to influence subsequent updates of the external entropy. Second, the attacker would need to have additional knowledge regarding the internal entropy.

5.2.2 Pseudorandom generator

Our pseudorandom generator combines the implementation strengths of different solutions such as the Pseudo Random Function (PRF) of SPINS [Per+02] (which is based on an HMAC function), provably secure constructions for building robust PRGs [Dod+13]; [FSK11], and unbounded state spaces through cryptographic primitives [Sta17].

As HASH function we have chosen SHA512. As HMAC function, we have chosen the one time authentication function *crypto_ onetimeauth()* from NaCl [BLS12]. This function ensures security and performance while generating outputs of 16 bytes that are indistinguishable from random.

PRG at the devices. As the controllers and forwarding devices do not have a source of strong entropy, we implemented a slightly modified version of the algorithm for these components to use this logically-centralized security service provided by the ANCHOR. Essentially, we replace the *entropy_ get()* function by *entropy_ remote()*. Instead of using local data, this function makes an entropy request to the ANCHOR to obtain a source of strong entropy. This function is essential to provide recovering security by refreshing, improving the resilience of the PRG.

5.2.3 Secure cryptographic key generators

Based on the algorithm proposed in [Kre+17], we have implemented an iDVV-based secure cryptographic keys generator that supports seven different cryptographic primitives. Specifically, we use each of these primitives as input to the *idvv_ next(primitive_ id)* function that is used to generate the next key. In our implementation, we used the following primitives: *MD5*, *SHA1*, *SHA512*, *SHA256*, *poly1305aes_ authenticate*, *crypto_ onetimeauth*, and *crypto_ hash*. While the first four functions are provided by OpenSSL, the last three are provided by an independent implementation of Poly1305-AES and NaCl. As MD5 and SHA1 are deprecated, we use them only for performance comparison purposes.

To understand the rationale for our implementation, we give a bit of context to clarify the difference between our solution and traditional KDFs. Both solutions are used to generate secure cryptographic keys that can resist different types of attacks, such as exhaustive key search attacks [YY05]. KDFs have common design characteristics, such as strong hash functions to compute digests for the raw key material. A secure KDF can be defined as $\mathrm{H}^{(c)}(p||s||c)$ [YY05]. H is a strong hash function such as SHA256 or SHA512. The exponent C represents the number of iterations used to make the task of the attackers harder. A common value for C is 2^{16}. This exponent is particularly necessary if the entropy of the input p (e.g., password, seed, key) is unknown. In practice, the input of the KDF is likely to be of low-entropy [YY05]. While in some use cases a high exponent C might be necessary to increase the cost of an attack trying to recover the key, it also significantly increases the cost of the key derivation function for high performance latency-sensitive applications.

Differently from a traditional key derivation scheme, our implementation using the iDVV generator in the context of ANCHOR uses high-entropy values. In other words, we do not need to recur to the exponent C as a means to compensate a potentially low-entropy p. By using by default two 32 bytes indistinguishable from random values in our generator, we make the task of an attacker very hard. It is also worth mentioning that iDVVs are essentially used in an association basis, i.e., they have a relatively short lifetime.

5.2.4 Implementation using Ryu and Open vSwitch

We have implemented a second, simplified version of the system, focused on the essential registration and authorization functions of ANCHOR. We have used the Ryu controller [Ryu19] for the control plane, and Open vSwitch (OVS) (`https://www.openvswitch.org/`) for the data plane. Ryu fully supports all versions of OpenFlow, including Nicira Extensions, and is officially integrated into OpenStack Networking (Neutron). OVS is the main software switch used in virtualized data centers (e.g., VMWare NSX [VMw20], OpenStack [The19b], OpenShift [Red18]). In addition, physical switches such as the Pica8 family [Pic19] rely on PicOS [Pic18], a user-space application running on top of an unmodified Linux kernel, providing OpenFlow support (version 1.0 to 1.4) through integration with standard OVS. This second implementation further strengthens our case for the effectiveness of deployment of ANCHOR in production SDN systems that include both software and hardware data planes.

Some key building components have made Pica8 attractive for both industry and academia. For example, the unmodified Debian Linux distribution in Pica8 enables faster updates, keeps up with the latest kernel innovations, and allows users to use standard Linux. Virtual application-specific integrated circuit (ASIC) technology is another example, which is a hardware abstraction layer to support multiple hardware and ASICs without modifying any of the PicOS tools. Other examples include the support to traditional switching and routing protocols, and hardware-accelerated Open vSwitch. Some of the key building components, that have made Pica8 attractive for both the industry and academy, are the unmodified Debian Linux distribution (enabling faster updates, keeping up with the latest kernel innovations, and allowing users to use standard Linux), virtual ASIC technology (vASIC®), which is a hardware abstraction layer to support multiple hardware and ASICs without modifying any of the PicOS tools, support to traditional switching and routing protocols, and hardware-accelerated Open vSwitch. The later is achieved by modifying the OVS Userland implementation to interact with the PicOS vASIC to download OpenFlow states to ASICs. The same approach can be used by switching devices relying on the Open Network Linux (ONL) [Big20]. ONL is a specific Linux distribution designed for open hardware switches, i.e., forwarding devices built from commodity components. ONL supports a variety of switching devices such as Mellanox Switches (SN2700, SN2100 and SN2410), Accton AS5512 (Nephos/MediaTek switch), Dell Z9100-ON, Quanta LY6 and LY8, and Wedge 100.

Additionally, as shown by previous research [LHM10]; [YJ15]; [De +14]; [Wan14]; [KSG14]; [Fon+15]; [Bar+17]; [Lan19], emulated OpenFlow-based networks, using Mininet [Lan19] and OVS, provide the requirements and the quality needed to realistically emulate and, afterward, deploy the exact same solutions in production environments.

We modified Open vSwitch (v2.10.0) and Ryu (v4.28) to support the registration and association functions provided by ANCHOR. To evaluate our solution on a realistic scenario, using the Mininet emulator (`https://github.com/mininet/mininet`) [LHM10]; [YJ15]; [De +14]; [Wan14]; [KSG14]; [Fon+15]; [Bar+17] (further details in Section 5.3.4), we modified the behavior of the core hub module (`ryu/lib/hub.py`) of Ryu. Similarly, we changed the behavior of the communication stack (`ovs/lib/stream`) of OVS. Specifically, instead of just opening a new communication channel with the controller, our modified OVS registers itself with ANCHOR (having obtained the network administrator's authorization), and sends an association request to the controller. In Section 5.3.4 we present the results of ANCHOR providing network protection against a rogue switch that is added to the network by an attacker, as an example use case.

5.3 Evaluation

In this section we evaluate the essential security mechanisms and services of our architecture.

For the performance measurements, we used machines with two quad-core Intel Xeon E5620 2.4GHz, with 2x4x256KB L2 / 2x12MB L3 cache, 32GB DIMM at 1066MHz, with hyper-threading enabled. These machines were interconnected by a Gigabit Ethernet switch and ran Ubuntu Server 14.04 LTS.

5.3.1 Source of entropy and PRGs

We empirically evaluate both the source of strong entropy and PRGs through statistical methods and tools, following state of the art recommendations [Bas+10]. To achieve our goal, we used NIST's test suite [NIS18]. We generated one file containing 50MB of random bits per generator. These files were used as input for the test suite tool Statistical Test Suite (STS) [NIS18]. In the end, our generators passed the absolute majority of tests and subtests: they failed only 2 sub-tests out of 188 (passed 146 out of 148 non-overlapping template matching), as summarized in Table 5.2. This gives a high level of confidence in our generators.

5.3.2 On the performance of key generation

In this section, we analyse the performance of our key generator, which is essential to provide low latency and high throughput control plane communication at a low cost.

Figure 5.2 shows the latency of the seven cryptographic primitives we used with our generator. We tested each primitive by generating keys of different sizes (16, 32, 64, and 128 bytes). The best performance is achieved by the implementations based on SHA1 and MD5, as expected. However, these two implementations have also the worst serial correlation coefficient, as shown in [Kre+17]. The generators that use SHA512 or Poly-OTP have good performance, achieving a good security-performance tradeoff.

5.3.3 Device-to-device communication performance

Connection establishment. While a TLS connection takes around 19 ms to be established, a device association using the ANCHOR takes less than 0.06 ms. This means that ANCHOR can easily support large-scale data centers (e.g., 1k switches

Test	**Result**
Frequency	✓
Block Frequency	✓
Cumulative Sums (forward)	✓
Cumulative Sums (backward)	✓
Runs	✓
Longest Run of Ones	✓
Binary Matrix Rank	✓
Discrete Fourier Transform	✓
Non-overlapping Template Matching	146/148
Approximate Entropy	✓
Random Excursions	8/8
Random Excursions Variant	18/18
Serial (first)	✓
Serial (second)	✓
Linear Complexity	✓

Table 5.2: STS: results of the single tests

and 100k hosts [Gre+08]; [ALV08]; [BAM10]) while being orders of magnitude more efficient than traditional solutions for this particular metric. The scale of the improvement of our connection setup process when compared to the TLS handshake is due to three main factors. First, our algorithm has half the number of steps. Second, we use symmetric cryptography only. Third, we use the fast ciphering provided by NaCl.

Communications overhead. Figure 5.3 shows the results of control plane communications using OpenSSL, TCP, and two versions of ANCHOR. For communication of up to 128 forwarding devices, sending 10k control messages each, our solution requires (while offering stronger security guarantees - see below) only half of the resources and time of an OpenSSL-based implementation using AES256-SHA, the most widely available cipher suite.

We can also observe the overhead of confidentiality (TCP-ANCHOR-EMAC) in Figure 5.3. In contrast to providing only authenticity and integrity (TCP-ANCHOR-MAC), confidentiality incurs an overhead of around 15%.

It is worth emphasizing that we achieved these results by ensuring also much stronger security, as we generated one secret key *per packet*. On the other hand, the OpenSSL-based implementation used a single key (for the symmetric ciphering)

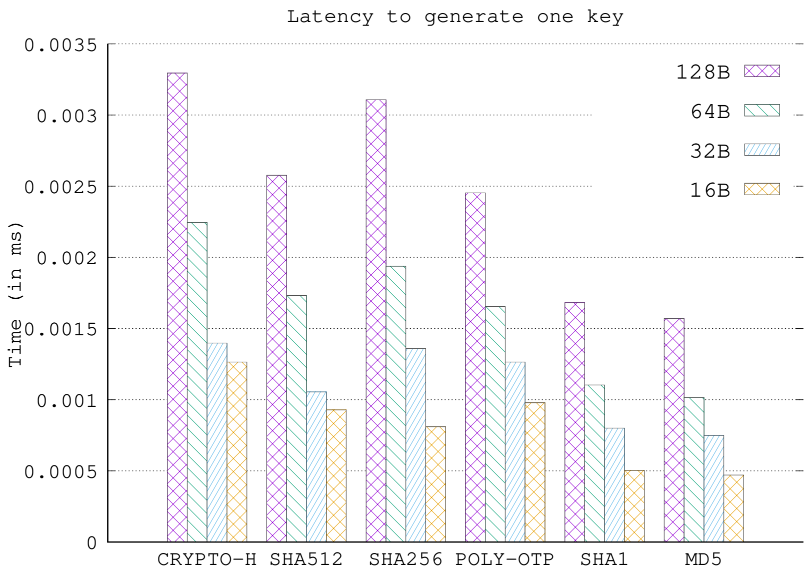

Figure 5.2: Latency of different iDVV generators

for the entire communication session.

5.3.4 Attack prevention

A type of attack that is recurrently presented as an important security threat in the context of SDN is the introduction, by an attacker, of rogue switches in the network (see [KRV13]; [AAS14]; [Chi+15]; [KF15]; [CL17]; [LMK16]). A set of switches under control of an attacker can be used for a DDoS attack, for instance, negatively affecting SDN control. We use this type of attack as an example use case that shows the logical centralization of security services in ANCHOR to enable attack prevention. The defence against this type of attack consists of a switch having to register itself to ANCHOR before being able to associate with the controller. If either the registration or association process fails, the switch connection with the controller is automatically dropped.

To demonstrate this functionality, we set up an experiment using the second version of our system (the one with OVS and Ryu as the data and control planes, respectively). We emulated a small network with Mininet, comprised of five switches (`s0` to `s4`) and five hosts (`h0` to `h4`), following a tree topology with `s0` at the root. Each network host is connected to one switch (e.g., host `h4` is connected to switch `s4`). To emulate the attack, we assumed `s2` to be a rogue device introduced to the network by an attacker. As the network manager has not registered `s2` into the system, this switch should not be able to associate itself with

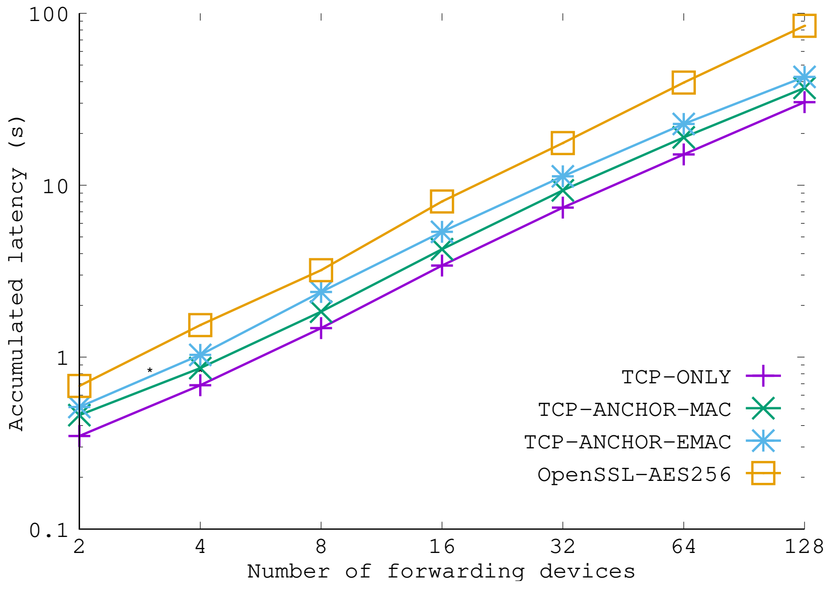

Figure 5.3: Control plane communication costs

the controller. As a result, host `h2` should not be reachable by any other host, and vice-versa.

The outcome of the experiment was as follows. Once the mininet network was up and running, we have run the `pingall` command to verify the reachability of all hosts. We observed an overall packet loss of 40% – the result of 1 out of 5 unreachable hosts (`h2`, in this case). Each host executes a reachability test for all other four hosts. However, the reachability test for host `h2` fails. In case of `h2`, all reachability tests fail. In a closer inspection, we verified that while the simulation was running, switch `s2` periodically tried to associate itself with Ryu, without success, as expected.

5.3.5 Traditional solutions *versus* ANCHOR

In Table 5.3 we provide a summarised comparison between a traditional solution and our ANCHOR. As traditional solutions we considered the EJBCA (`http://www.ejbca.org/`) and OpenSSL, two popular implementations of PKI and TLS, respectively. As we have shown before, our bootstrap process (device registration and association) is much faster and our connection latency is also significantly lower. In addition, our solution has nearly one order of magnitude less LOC and uses four times fewer external libraries. This makes a difference from a resilience perspective. For instance, to formally prove more than 717k LOC (EJBCA + OpenSSL) is by itself a tremendous challenge. Moreover, it gets considerably worse if we take into account eighty external libraries and eleven programming languages.

Table 5.3: Traditional solutions *versus* ANCHOR

Functionality		**Traditional solutions**	ANCHOR
Typical Software		EJBCA (PKI) + OpenSSL (TLS)	ANCHOR + iDVV + NaCl
Device Identification		based on certificates; costs = issue a certificate	based on unique IDs controlled by the ANCHOR; costs = register the device (assign a unique ID)
Device Registration		based on certificates; costs = certificate installation + security control policy/service	registration protocol; costs = register the device + iDVV bootstrap
Device Association & KeyGen		12 step mutual handshake + DH for session keys (incl. certificate validation - any two device can establish an association)	6 step trust establishment with ANCHOR + iDVVs per message, session, interval of time, ... (ANCHOR has to authorize association)
Security Properties	Authenticity	✓	✓
	Integrity	✓	✓
	Confidentiality	✓	✓
	PFS	✓(*)	✓
	PCS	✗	✓
	PQS	✗	✓
Communications		symmetric cryptography (cipher: AES256-SHA)	symmetric cryptography (cipher: Salsa20)
TLS stack		highly configurable and complex (717k LOC)	easy to use, simple (85k LOC), and efficient

Our proposed architecture offers a functionally equivalent level of security with respect to properties such as authenticity, integrity and confidentiality, when compared to traditional alternatives. Additionally, ANCHOR offers a higher level of security by providing post-compromise security and post-quantum security. While the former is ensured through post-compromise recovery, the latter is a consequence of using only symmetric cryptography. Further, the lightweight nature of our mechanisms, such as the iDVV, make them amenable to be used on a per message basis to secure communication, increasing cryptographic robustness. Moreover, by having fewer LOC, we significantly reduce the threat surface.

Finally, it is worth emphasizing that the PFS (*) of traditional solutions, such as those provided by the different implementations of TLS, is not easy or simple to enforce. First, in spite of TLS providing ciphers that offer PFS, in practice, different cipher suites do not feature it [SHS15]. This means that not all implementations and deployments of TLS offer PFS, or provide it with very low encryption grade [Hua+14]; [Dig17]; [Nam19]. To give an example, widely deployed web servers, such as Apache and Nginx [Dig17] and most DHE- and ECDHE-enabled servers suffer from weak PFS configurations [Hua+14]; [Adr+15b]; [SDH16].

5.4 Discussion

We briefly discuss the identified gaps discussed in §3.2. We also show, in §B.5 and §B.6 of Appendix B, to which extent these solutions cover eleven of ONF's security requirements and provide a thorough security analysis of ANCHOR's algorithms. We conclude the section with a critique of our choices and results.

5.4.1 Meeting the challenges

Security vs performance? Control channels need to provide high performance (high throughput and low latency) while keeping the communication secure. However, as it has been shown, security primitives have a non-negligible impact on performance. To mitigate this problem, we selected appropriate cryptographic primitives (SHA512), libraries (NaCl), and key generation mechanisms (iDVV) to ensure the security of control plane communications maintaining high performance. By logically centralizing the fundamental aspects of these mechanisms in the ANCHOR, the performance overhead introduced in forwarding devices and controllers is limited, as they require only minimal functionality to 'hook' to the ANCHOR instructions.

Complexity vs robustness? Traditional implementations of SSL/TLS, such as OpenSSL, have a large, complex code base, that recurrently leads to vulnerabilities being discovered. Similar problems are observed in PKI subsystems. It is known that an effective means to achieve robustness is by reducing complexity. Hence our choice for the NaCl and iDVV mechanisms to help fill the gap, since they are respectively lightweight (small code base), efficient, yet secure alternatives to OpenSSL-like implementations. As such, they are a robust solution to provide authentication and authorization material for the secure communications protocols we propose. They are also amenable to verification mechanisms aimed to assure correctness, which are much harder to employ in very large code bases. Again, the centralization of the non-functional mechanisms introduced in our solution is the key to reduce complexity of networking devices, improving their robustness.

Global security policies? We have argued that controllers and network devices often employ suboptimal network authentication and secure communication mechanisms, despite recommendations from ONF and other such organizations for the opposite. This problem is very similar in nature to the state of affairs in networking before SDN. In traditional networks, enforcing relatively "simple" policies such as access control rules [Cas+07] or traffic engineering mechanisms [Jai+13] was either very hard or even impossible in practice. Given the current undesirable state of affairs, we believe the same to be true for non-functional properties, with security as a prominent example. Our logically centralized ANCHOR architecture addresses this gap by providing a means for making centralized policy rules and the necessary mechanisms to enforce them, permeating the SDN architecture in a global and coherent way.

Resilient roots-of-trust? We debated that there is a (probably reduced) number of functions which should not be left to ad-hoc implementations, due to their criticality on system correctness. The list is not closed, but we hope to have shown that strong sources of entropy and resilient indistinguishable from random number generators are clear examples of difficult-to-get-right mechanisms that benefit from such logically centralized approach. ANCHOR addresses this issue, by providing the motivation to design and verify careful and resilient once-and-for-all implementations of such root-of-trust mechanisms, which can then be reinstantiated in different SDN deployments.

5.4.2 Devil's advocate analysis

Doesn't the logical centralization of non-functional properties create a single point of failure? The results of this thesis already go a long way to improving robustness of a single root-of-trust, compared to the state of the art: lowering failure probability; mitigating and recovering from the consequences of failure. The logical next step would be to try and prevent failures in the first place. However, the failure of a simplex system of reasonable complexity cannot be prevented.

Nevertheless, note that logical centralization is not necessarily physical centralization. And indeed, our plan for future work (and the way we drafted our architecture paved the way) is to leverage state-of-the-art security and dependability mechanisms using replication. For instance, to achieve tolerance of Byzantine faults, we can readily enhance ANCHOR by replication, taking advantage of state machine replication libraries such as BFT-SMaRt [BSA14], replicating and diversifying components to prevent failure of this logically central point, with the desired confidence. These concepts have been applied to root-of-trust like configurations similar to ANCHOR [ZSV02]; [CS04]; [Kre+14]. Furthermore, systems designed with state machine replication in mind can also handle different types of threats, such as DoS, without compromising the operation of the service [Kre+16].

Won't the natural hardware evolution be by itself enough to reduce the penalty imposed by cryptographic primitives? The recent reality seems to contradict this assertion – hardware evolution does not seem enough, for several reasons. First, new hardware architectures can benefit from different existing software-based solutions. For instance, both NaCl and OpenSSL take advantage of hardware-based AES accelerators. Second, and as is well known, the fixed price of advancements in hardware seems to be coming to an end [IEE15]. This is made clear by most of the major IT companies, such as Google and Microsoft, redesigning existing software as a response to cope with this problem [Liv+15].

Aren't traditional PKI and TLS implementations enough? Following what is becoming recurrently advocated by many in the industry and in the security community, we have tried to argue that the simplicity and size of software and IT infrastructure matters [Cis14]; [Ver15]. Higher complexity has been shown to lead inevitably to an increased likelihood of bugs and security incidents in software. Indeed, different implementations of PKI and TLS have been recently used as powerful "weapons" for cyber-attacks and cyber-espionage [PwC14]; [BOC15], leading to concerns about their robustness. Contrary to what this argument may suggest, that does not mean PKI and TLS are "broken". We believe they remain fundamental to various IT infrastructures. However, as the main challenges of securing SDN are usually relatively constrained to within a network domain, we have come to understand that simpler, domain-specific solutions seem to be preferable in this environment when compared to complex infrastructures such as the PKI, and large code bases such as OpenSSL.

Wouldn't the use of out-of-band control channels solve most problems? Out-of-band channels may be useful in some contexts, but they are not "intrinsically" secure. It is a recurrent mistake to consider physical isolation, *per se*, as a form of security. Several studies have indeed argued the opposite: that out-of-band channels worsen the problem, by making control plane management more complex and less flexible, endangering control plane communications [Edw14]; [ME15]. We do not take a stance in this discussion, but the fact is that real incidents, such as National Security Agency (NSA) sniffing of Google's cables between data centers [Sch15], seem clear examples that out-of-band channels are not, *per se*, enough.

5.4.3 Other use cases of ANCHOR

Using ANCHOR *beyond control plane communications.* As already alluded to in Section 3.1, ANCHOR can be extended to support other use cases. For instance, one application running on top of the SDN controller could be required to provide proper credentials to identify itself. Once successfully authenticated, it should have access to a specific set of system attributes defined by the operator during registration (e.g., read, write, notify, among other system calls [Fer+13]; [ABA17]). Towards this goal, different controllers could rely on authentication and authorization attributes globally enforced by ANCHOR. Another interesting use case for ANCHOR would be to offer security support for controller clustering. This is a timely problem. To give an example, previous and current releases of OpenDaylight do not provide encryption or authentication of control messages exchanged among controller instances [Ope18b]; [Ope20]. Since each controller instance would need to be registered with ANCHOR, it would be possible to provide the same security mechanisms and services we grant to the southbound connection, to ensure security in east-west communication between controllers.

Addressing other non-functional properties of SDN. The design of ANCHOR is generic enough to accommodate non-functional properties beyond security, such as dependability or quality of service. With respect to the former, ANCHOR could help in modularizing the problem of replicated control. Specifically, ANCHOR could be responsible for coordination between controller replicas, for instance by guaranteeing a strongly consistent view of the network across all instances. Similar to our security use case, the additional modularity of such design would allow a clean separation of concerns that could simplify the design of the various components. Recent proposals [Bot+16] have indeed started following a similar design choice. ANCHOR could also provide trusted measurement services for ensuring a certain level of service even in the presence of malicious forwarding devices. For instance, once a malicious forwarding device were detected [MTG18]; [KF15], ANCHOR could automatically remove it from the list of legitimate devices, forcing the disconnection of those devices by the controllers of the network, which would be registered to receive such events. The subsequent topology updates on the controllers would trigger automatic traffic re-routing to ensure the quality of service of applications.

Final remarks

ANCHOR is the blueprint of an architectural framework addressing the problem of enforcing non-functional properties in SDN, such as security or dependability. Reiterating the successful philosophy behind the inception of SDN itself, we advocate the concept of logical centralization of SDN non-functional properties provision.

Taking ‘security’ as a proof-of-concept use case, we have shown the effectiveness of our proposal. We made a gap analysis of security in SDN, and populated the ANCHOR middleware with crucial mechanisms and services to fill those gaps and enhance the security of SDN.

We evaluated the architecture, especially focusing on the security-performance analysis tradeoff, giving proofs of the algorithms, cryptographic robustness analyses, and experimental performance evaluations. By resorting to recent primitives, lightweight albeit secure, like NaCl and iDVV, we outperform the most widely used encryption of OpenSSL by 50%, with a higher level of security. Our solution also fulfills eleven of the security requirements recommended by ONF.

The mechanisms we propose are certainly not the final answer to SDN security problems. That is not our claim. However, we believe that an architecture that logically centralizes the non-functional properties of an SDN, has the potential to address some of the most prominent unsolved problems regarding the robustness of the infrastructure.

Chapter 6

R-ANCHOR: Robust and Resilient ANCHOR

In this chapter, we review our approach to the systematic mitigation of the risk of logical centralization. As we show, from the initial ANCHOR design presenting a high level of robustness, we evolve to R-ANCHOR, providing resilience in several steps, towards a more effective solution against the single-point-of-failure syndrome of ANCHOR. Whilst the resilience approach of this thesis concerns the management aspects, further research avenues towards a fully resilient architecture beyond this thesis — management, control and date planes — are suggested in Chapter 7. First, we shortly review the status quo before and after ANCHOR in Section 6.1. Then, in Section 6.2, we review baseline robustness mechanisms included in the design of ANCHOR. The remainder of the chapter focuses on additional, more effective measures, complementing the baseline ones. In §6.2.1, we present a semi-automated approach to achieve post-compromise security, which can readily be applied to instantiations of the baseline architecture discussed in Chapter 5. Then, we evolve to a more ambitious design, based on some architectural evolutions materialized by the R-ANCHOR resilient management architecture, introduced in Section 6.3. In Section 6.4, we present the system setup and protocols. The reader will observe that the level of automation is dramatically increased, including the algorithmic steps to achieve fully automatic restart for PCS after a full compromise; and mitigation of insider threats. With regard to the latter, reports have recurrently shown that insiders are responsible for approximately 50% of all reported security incidents [IBM18]; [Obs18]; [LBL16]; [Sec19]. Recent findings unveil that nearly 90% of organizations feel vulnerable to insider threats [Cyb18]; [Sec19]. The system can be configured optionally with or without these additional measures, since they obviously incur a cost in complexity that must be justified by the criticality of the application.

6.1 Before and after ANCHOR

Before ANCHOR, as we have shown, adversaries could deploy their attacks through the multiplicity of threat vectors specific to SDN security, which we recapitulated in Section 3.3, and hit on the multiple vulnerable points in order to progressively compromise and control the system. ANCHOR modifies the scenario, as we have shown and evaluated in Chapters 4 and 5, by shrinking the threat plane. ANCHOR becomes then the target to attack par excellence, and indeed a SPoF: the failure or compromise of ANCHOR would not bring the whole system down at once, but would compromise the availability of the security services it provides to the networking devices. While the temporary unavailability of such services might not be a problem for smaller or non-critical networks, it might have a non-negligible impact on the operations of mission critical or large-scale networks. This calls for a range of mechanisms of incremental strength to handle possible failures of ANCHOR.

6.2 Baseline measures for hardening ANCHOR

The compromise of a root-of-trust is of great concern, since crucial services normally depend on it being secure and dependable. As we stated before, we have a long-term strategy towards the resilience of ANCHOR. It starts by improving the inherent reliability of its simplex (non-replicated) version, by hardening it in the face of failures. For instance, different from existing traditional security services such as Kerberos and RADIUS, we still provide some security guarantees even when ANCHOR has been compromised.

This section recapitulates the prevention and assurance mechanisms promoting robustness, already incorporated in the baseline designs of ANCHOR and KISS, as described in the previous chapters. Firstly, the design was hardened by several measures: incorporating robustness into all basic functions (e.g., sources of entropy, random number generators); using symmetric cryptography for performance and post-quantum security, systematic informal proof of correctness of all algorithms and services, and machine-assisted formal verification of the more critical ones, for increased assurance.

Second, we propose protocols to achieve two security properties guaranteeing respectively, the security of past (pre-compromise) communications, and of future (post-recovery) communications. This provides a significant improvement over existing root-of-trust infrastructures.

The first property is *perfect forward secrecy*, namely, the assurance that the compromise of all secrets in a current session does not compromise the confidentiality of the communications of the past sessions. The enforcement of PFS is systematically approached and discussed in the algorithms we presented in Chapters 4 and 5.

The second property is *post-compromise security*. While PFS considers how to protect past communications, PCS considers how to reinstate and re-establish the secure communication channels, for future communications. This security property had so far been considered only in specific scenarios such as secure messaging [YR15]; [YRC18]. The algorithms of the baseline architecture presented in Chapter 5 already include the necessary variables to prepare for and enable a form PCS, with manual recovery (initiated by system administrators), which we explain next.

6.2.1 Post-compromise recovery

As previously explained, when ANCHOR is reinstated after a compromise, it would be extremely useful to have a way to re-establish the secure communication channels between ANCHOR and all participants in the simplest and fastest way possible, thus reducing the time the system is in a compromised state and/or unavailable for correct service. In what follows, we propose a semi-automated way of reinstating the system after a compromise.

We recall that the algorithms of the baseline architecture already left in place the necessary 'recovery keys':

- ANCHOR master recovery keys $\mathrm{Ke}_{A_{rec}}$ and $\mathrm{Kh}_{A_{rec}}$
- Manager recovery keys $\mathrm{Ke}_{AM_{rec}}$ and $\mathrm{Kh}_{AM_{rec}}$
- Device recovery keys $\mathrm{Ke}_{AD_{irec}}$ and $\mathrm{Kh}_{AD_{irec}}$

The ANCHOR master recovery keys are strong random keys, the Manager and Device recovery keys are deterministically generated from the former — although indistinguishable from random — in a logical sequence $\mathrm{Kx}_{A_{rec}} \rightarrow \mathrm{Kx}_{AM_{rec}} \rightarrow \mathrm{Kx}_{AD_{irec}}$. This reduces the number of uses of the master recovery keys. All of them live essentially off-line, and exist on-line for the short periods of setup, registration, or recovery. The threat model is thus that these keys are trusted not to be compromised, unlike all the other keys in the system.

Algorithm 8 presents our solution to re-establish the secure communication channels when ANCHOR is compromised. Intuitively, since ANCHOR's master recovery keys $\mathrm{Ke}_{A_{rec}}$ and $\mathrm{Kh}_{A_{rec}}$ are stored securely offline, they are unknown to an attacker who has stolen all secrets from the ANCHOR server. To initiate recovery, the entity managing the ANCHOR server A (the system administrator) manually intervenes and gets it reinstated, installs these recovery keys from offline (line 1), with which it computes the $\mathrm{Kx}_{AM_{rec}}$ recovery key pair for each manager M (l.2). To

Algorithm 8: ANCHOR basic post-compromise recovery

```
                    {For each manager $M$ and its associated devices $\{D_i\}_{i=1}^{n}$}
1.   A          installs($Ke_{A_{rec}}$, $Kh_{A_{rec}}$);
2.              $Ke_{AM_{rec}}$=H($Ke_{A_{rec}}$ || M); $Kh_{AM_{rec}}$=H($Kh_{A_{rec}}$ || M);
3.              generates($Ke'_{AM}$, $Kh'_{AM}$, $\{Ke'_{AD_i}, Kh'_{AD_i}\}_{i=1}^{n}$);
4.              $M_k$=($Ke'_{AM}$, $Kh'_{AM}$, $\{Ke'_{AD_i}, Kh'_{AD_i}\}_{i=1}^{n}$).
5.   A → M      [Recovery, A, M, $E_{AM_{rec}}(M_k)$],$HMAC_{AM_{rec}}$
6.   A          destroys($Ke_{A_{rec}}$, $Kh_{A_{rec}}$, $Ke_{AM_{rec}}$, $Kh_{AM_{rec}}$)
7.   M          installs($Ke_{AM_{rec}}$, $Kh_{AM_{rec}}$);
8.              $Ke_{AM}$ = $Ke'_{AM}$; $Kh_{AM}$ = $Kh'_{AM}$;
                    {For each device $D_i$}
9.   M          $Ke_{AD_{i_{rec}}}$=H($Ke_{AM_{rec}}$ ||$D_i$); $Kh_{AD_{i_{rec}}}$ =H($Kh_{AM_{rec}}$||$D_i$).
10.             $M_l$=($Ke'_{AD_i}$, $Kh'_{AD_i}$).
11.  M → $D_i$  [Recovery, A, M, $D_i$, $E_{AD_{i_{rec}}}(M_l)$],$HMAC_{AD_{i_{rec}}}$.
12.  M          destroys( $Ke_{AM_{rec}}$, $Kh_{AM_{rec}}$, $Ke_{AD_{i_{rec}}}$, $Kh_{AD_{i_{rec}}}$, $Ke'_{AD_i}$, $Kh'_{AD_i}$);
13.             $Ke_{MD_i}$ = H($Ke_{MD_i}$); $Kh_{MD_i}$ = H($Kh_{MD_i}$).
14.  $D_i$      $Ke_{AD_i}$ = $Ke'_{AD_i}$; $Kh_{AD_i}$ = $Kh'_{AD_i}$;
15.             $Ke_{MD_i}$ = H($Ke_{MD_i}$); $Kh_{MD_i}$ = H($Kh_{MD_i}$).
```

continue the recovery process, ANCHOR generates new random keys to be shared with all Ms, and all D_i (l.3).

Then, ANCHOR sends to each M (line 5) a 'Recovery' message to re-share the just-generated keys (contained in M_k created in l.4) with the devices under the network administrator's control. The messages are encrypted and MAC protected with the corresponding recovery keys of each M. The new shared keys will be used to protect future communications. The recovery keys in A are immediately destroyed (l.6), i.e., as soon as they are not needed anymore.

Each M implements the recovery operation with each of the devices it manages. In preparation, M starts by installing its recovery key pair (l.7), $Ke_{AM_{rec}}$ and $Kh_{AM_{rec}}$. Those keys will be used to decrypt the 'Recovery' message from A, as well as to compute the recovery key pair for each device (l.9). The new shared secret key pair for secure communication between A and M (sent by A) replaces the possibly compromised keys at M (l.8).

The process continues with the recovery of devices subtended by this M. Using the key material recovered from the 'Recovery' message sent by A, M sends in turn to each D (line 11) a 'Recovery' message to re-share those keys (Ke'_{AD_i}, Kh'_{AD_i}), encapsulated in M_l created in line 10. These keys, for A-D communication, will be installed by each D_i (l.14), replacing the old keys. M destroys, in line 12, all the recovery keys, as well as the new A-D communication keys Ke'$_{AD_i}$, Kh'$_{AD_i}$ — for which M was a mere mediator. Finally, all keys shared between M and its subtended devices will be updated (rolled) to their hash value, on both extremities of the M-D_i channel (lines 13 and 15). As mentioned previously, this key update is used to provide perfect forward secrecy.

If one or more network administrators M get compromised, they can be recovered using a slightly modified run of Algorithm 8. For simplicity, and without loss of generality, we give just the modified lines in the algorithm below, for the Manager post-compromise recovery protocol. First, M is reinstated, and gets the recovery keys through an out-of-band channel (Ke$_{AM_{rec}}$, Kh$_{AM_{rec}}$), so that it can get the new keys sent by A under the latter. Essentially, M will re-establish and re-share its secrets with the help of ANCHOR, in a similar way as described in Algorithm 8. Now, only new M-D keys are involved, Ke'$_{MD_i}$, Kh'$_{MD_i}$, for all D_i subtended by M (generated by A in line 3). The algorithm proceeds as the original, until it is time for M to share keys with each D_i. However, now Ke'$_{MD_i}$, Kh'$_{MD_i}$ are shared, instead of Ke'$_{AD_i}$, Kh'$_{AD_i}$ (line 10). The algorithm finalizes, with the update of shared M-D communication keys, this time not a key roll-up, but the installation of new keys Ke'$_{MD_i}$, Kh'$_{MD_i}$ on both sides of the M-D channel (lines 13 and 15). The operation in line 14 is no longer needed, for obvious reasons. All the lines of the original Algorithm 8 not mentioned in the paragraph above remain the same for the modified Manager post-compromise recovery protocol.

Algorithm: Network manager M post-compromise recovery - modified steps

```
            {For each manager $M$ and its associated devices $\{D_i\}_{i=1}^{n}$}
...
3.    A     generates($\{Ke'_{MD_i}, Kh'_{MD_i}\}_{i=1}^{n}$);
...
            {For each device $D_i$}
...
10.   M     $M_l$=($Ke'_{MD_i}, Kh'_{MD_i}$).
...
13.   M     $Ke_{MD_i} = Ke'_{MD_i}$; $Kh_{MD_i} = Kh'_{MD_i}$.
14.   $D_i$   no operation;
15.         $Ke_{MD_i} = Ke'_{MD_i}$; $Kh_{MD_i} = Kh'_{MD_i}$.
```

In the following sections, we elaborate on more sophisticated mechanisms, in the way of promoting resilient management of ANCHOR, achieving fully-automatic PCS recovery and insider threat mitigation (an important threat vector as we have explained earlier [Hom+19]).

6.3 R-ANCHOR Resilient Management Architecture

R-ANCHOR builds on ANCHOR, retaining all its properties and aiming at addressing some of its limitations, namely in the management scope: the syndrome of SPoF of the management site, manual PCS recovery, vulnerability to malicious insiders (managers).

The incremental architecture shown in Figure 6.1, shows one of the fundamental changes. We re-factor ANCHOR's architecture using *architectural hybridization*. As we will detail later, this means that ANCHOR has now two parts, a major one, the 'payload' (i.e., where the main services of ANCHOR reside) and a trusted-trustworthy subsystem (the 'hybrid', a.k.a. 'wormhole' [Ver06]). The 'wormhole', and its interface to the payload, the wormhole gateway (WG), are designed and verified so as to provide very high assurance about the capacity of unconditional provision of correct albeit simple functions/services, whilst being tamperproof. These services should be provided to payload components through the wormhole gateway, even in face of the total failure of the system by accidental faults or malicious compromises [Ver06].

As we explain ahead, this architectural enhancement allows evolving the manual recovery, as previously discussed, to fully-automated post-compromise recovery, alleviating the syndrome of SPoF. It also opens avenues for more ambitious resilience mechanisms to be discussed as future work, such as fault and intrusion tolerant replicated ANCHOR server sets powered by efficient hybrid BFT protocols [Ver+13].

The other fundamental change is discussed ahead, and is concerned with modified protocols following a new trust/threat model, in order to address the problem of insider threats: malicious network managers (M).

6.3.1 System model

The system model of R-ANCHOR follows that of the baseline ANCHOR, with extensions that we describe below. Recapitulating: we are concerned with software-defined networking fabrics, where a main characteristic is the segregation of system activity between what are called the data and the control planes.

Baseline components. As explained earlier, in the context and focus of this thesis, the 'system' is the control plane part. Thus, the basic system components are: devices (D), i.e., forwarding devices (F) or controllers (C), and the ANCHOR

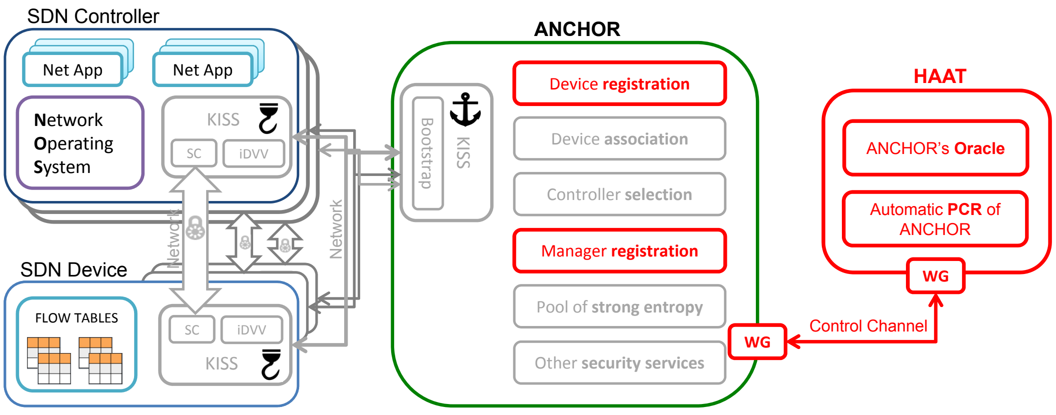

Figure 6.1: Overview of R-ANCHOR (ANCHOR's modifications in red color)

server (A), interconnected by a network that channels the control-plane activity (Figure 6.1).

Network. We assume fair First In, First Out (FIFO) reliable channels (e.g., TCP/IP) between devices, network managers and ANCHOR. Forwarding devices and controllers can communicate with all controllers and ANCHOR. The communication between any two elements can happen through different paths (or out-of-band channels) depending on the network topology, built-in redundancy, and so forth.

Architectural Hybridization. In R-ANCHOR, one additional albeit important modification in the system model is introduced: architectural hybridization [Ver06]. So the components just listed now form the 'payload' part, following a system model consistent with the expected characteristics of the environment where the system is deployed. This payload coexists with 'hybrids', or 'wormholes', which in hybridization lingo mean well-defined and self-contained subsystems following, by construction, system model and fault assumptions differentiated from the rest of the architecture, i.e., the 'normal' or 'payload' part (fault assumptions are discussed just ahead). For example, it is typical in secure and/or dependable hybrid system designs that whilst the normal part is asynchronous and subject to arbitrary faults, the 'hybrids' are trusted to be synchronous and not to fail, behaviors enforced by construction. Under such favorable conditions, these trusted-trustworthy components can reliably provide simple but effective services such as failure detection, counters, ordering, keys and signatures, random numbers, or reliable and timely execution of simple pre-defined functions. That is, they offer, for a very restricted set of services, stronger properties than would be possible in the payload part (normally subject to the security, dependability and performance impairments of systems of reasonable dimension and/or openness). The paradigm has been gaining increasing acceptance and traction in the design of secure/dependable systems [Ver06]; [Chu+07]; [Lev+09]; [Kap+12]; [Ver+13]; [BDK17].

***H**ardware-**A**ssisted **A**nchor of **T**rust.* Figure 6.1 depicts the specific wormhole used in R-ANCHOR, the HAAT, denoted T in what follows, which holds critical data (e.g., master and recovery keys), and executes simple functions (e.g., error, intrusion or failure detection, forced reset, recovery and rejuvenation), crucial to our objective of management resilience. The data and function results are made available to payload subsystems through the wormhole gateway (WG in the figure), as described next. In R-ANCHOR, using a hybrid model enables, amongst other things, a simple and effective solution to achieve automated post-compromise recovery. The detection functions it can host also contribute to improve error and failure recovery, alleviating the syndrome of SPoF. Recapitulating the baseline ANCHOR PCS process, described in §6.2.1, the recovery keys are computed through master recovery keys. However, in R-ANCHOR, instead of being stored offline and manually processed upon failure, those keys are stored in HAAT. In case of a disaster, the master recovery keys are readily used to compute the recovery keys and install them in ANCHOR, by means of a protocol executed through the wormhole gateway. The protocol provides a way to recursively share new keys between ANCHOR, network managers and devices. After the recovery, the recovery keys are "forgotten" by the payload system.

***W**ormhole **G**ateway (WG).* The payload ANCHOR server and the HAAT are connected by a special link, the WG. The link and the endpoints of the wormhole gateway interfacing between ANCHOR and HAAT are designed so as to provide very high assurance about: (i) the tamper-proofness of HAAT to ilegal accesses through ANCHOR and/or the link; (ii) the unconditional provision of the predefined HAAT services (such as forced reset), in face of faults and attacks. These include the general failure of the payload system by accidental faults or malicious compromises [Ver06]. The simplest implementation is that both ANCHOR and the HAAT should be inclosed in an isolated vault or room with strong physical access control (e.g., as recommended by Hardware Security Module (HSM) manufacturers, specialized security companies, and security experts [Saf12]; [Vau18]; [Bar19]; [ASG19]). The HAAT is only accessible by a local interface when the vault is open; the ANCHOR is accessible both locally, and remotely through secure session protocols (e.g., Secure Shell (SSH)) by authorized users [1]. Table 6.1 summarizes HAAT's API. The function `GetRecKeys()` returns the recovery keys of ANCHOR. It must be used by ANCHOR when it rejuvenates after a compromise. It is worth empha-

[1]Just to give a real feel, here is one possible implementation of this principle. The ANCHOR server is closely connected to the HAAT through a bus-level interface, allowing HAAT to follow and check ANCHOR's activity, and/or perform unconditional actions on the ANCHOR machinery, such as forced reset. The HAAT can be implemented through available COTS small computer technology (e.g., Raspberry Pi or other), capable of tight interface to the Personal Computer (PC)-level technology in the ANCHOR server implementation.

sizing that `GetRecKeys()` will only work right after a hard reset command, as described in Section 6.4.2. Failure detectors can send error, failure or compromise messages to HAAT using the `SendWarning(E|C|F)`. Assuming that HAAT receives one or more compromise (C) messages from reliable failure detectors, it sends a `HardReset` message to ANCHOR. As soon as ANCHOR's WG receives the message (`WaitForHardReset()`), it starts a hard reset of the server, rejuvenating ANCHOR.

Table 6.1: HAAT's API

Function	**Operation**
`GetRecKeys()`	Returns ANCHOR's recovery keys stored in HAAT.
`SendWarning(E|C|F)`	ANCHOR's detectors send error E or compromise C or failure F to HAAT's oracle.
`WaitForReboot()`	Waits for HAAT's reboot signal, forcing ANCHOR to recover in case of warning E or F.
`WaitForHardReset()`	Waits for HAAT's hard reset signal, forcing ANCHOR to rejuvenate in case of warning C.

Synchrony. We assume partial synchrony [DLS88], i.e., the system can behave asynchronously for some time, but it becomes synchronous with a certain probability, allowing the system to have processing and communication time bounds. Likewise, we assume that the clocks of ANCHOR and HAAT are loosely synchronized.

Social-technical components. We assume there is one logical entity performing the system administrator function (S). The system administrator is the only one entitled to: operate and manage ANCHOR, remotely or locally; manage HAAT locally. Our definition englobes both the human operator(s) and the machine(s) implementing S. In the following sections, we discuss the trust issues concerned with S. We assume there are several logical entities performing the network manager function (M). The network manager is entitled to: manage devices (C and F), remotely or locally. Our definition englobes both the human operators and the machines implementing M. We discuss the trust issues concerned with M below.

6.3.2 Trust and Threat model

We assume that the adversary does not have any access, physical or other, to the HAAT internals. We assume that the adversary does not have physical access to the ANCHOR, but may manage to remotely access it.

We assume that an unlimited number of forwarding devices and controllers in the system, or ANCHOR itself, can fail on account of accidental faults or attacks at any given time t. We consider as well that an attacker can obtain all knowledge of the victim device(s) or ANCHOR, including all stored secrets and the session status. However, the modified R-ANCHOR protocols continue to guarantee PFS. While we do not do anything directly about failed or compromised devices, which will be recovered from scratch, we provide an automated post-compromise recovery protocol for ANCHOR.

Our modified fault model for R-ANCHOR presents a framework for mitigating insider threats in the management umbrella, by restricting the actions of network managers. We assume that up to f out of n network managers entities M can be malicious. Besides the obvious configuration condition of $n > f$, in the algorithms that follow, we rely on this assumption to enforce that any critical operation in the system performed by network managers, such as registering or associating a new device, should require the intervention of at least $f + 1$ network managers entities to be deemed correct.

However, we assume that the system administrator entity S (as defined in the last section) is trusted. Consequently, in the algorithms described next, any critical operations in the system by S, are always deemed correct. Given the rareness of the intervention of S in the system operation, we can afford to substantiate the coverage of this assumption in a socio-technical manner, for example through a four-eyes policy (involving at least two human operators of S, for e.g. opening the vault, handling HAAT or the ANCHOR server, or the registration of a new network manager).

We assume an extended Dolev-Yao [DY83] threat model for the control-plane communication network, amongst ANCHOR, devices, network managers and system administrator. In consequence, an attacker may: (i) arbitrarily delay, drop, reorder, insert, or modify messages; (ii) log all messages; (iii) compromise any network device (e.g., a controller or a forwarding device) at any time. However, we assume the security of the used cryptographic primitives, including HMACs (e.g., HMAC-SHA256), hash functions (e.g., SHA-256), and symmetric encryption algorithms (e.g., AES), as well as of the strong hash functions and high entropy and randoms described earlier for ANCHOR. We also assume that an attacker can have control of the network for some time, but cannot control the entire network for the whole time. This is a reasonable assumption for this kind and scale of infrastructure.

We assume that an attacker can reach full control over the ANCHOR server and remaining payload system during a limited time (e.g., from t to t', $t < t'$). During this time, s/he has full control of the payload system and can access all locally stored secrets. However, short of a last-resort human detection of malfunctioning, we expect a much earlier and time-bounded latency in automatic error detection, made possible by the ANCHOR-HAAT interaction mechanisms through the WG which, as we have postulated (and design accordingly), are unconditional despite any compromise of ANCHOR. Namely, we are talking about host-based failure or intrusion-detection mechanisms, and automatic reset and recovery. As such, up to time t' ANCHOR's payload (server and devices) rejuvenation is automatically started, with the help of HAAT.

As hinted previously, the HAAT, besides synchronous, fails only by crashing, and its internals are completely shielded from external interaction. Substantiation of these assumptions should entail thorough verification. It should also be able of operating non-stop for some bounded time, even in case of power failures.

6.4 Setup and protocols

In this section, we discuss R-ANCHOR in detail. We start with the system roles and setup. Following, we delve into the technical and operational specifics of deploying and maintaining the correct operation of R-ANCHOR.

Before starting, for simplicity and without loss of generality, we denote $\mathrm{E}_{XY}()$ an encryption using encryption key Ke_{XY}, and we denote $[]$,HMAC_{XY}, respectively a message field inside $[]$, followed by an HMAC over the whole material within $[]$, using MAC key Kh_{XY}, where $X, Y \in \{\mathrm{T},\mathrm{S},\mathrm{M}_i,\mathrm{A},\mathrm{D}_i\}$. It is worth emphasizing that the system can generate strong keys using a suitable KDF based on the high entropy random material provided by ANCHOR's source of entropy [Kre+19]. In Table 6.2, for the comfort of the reader, we recapitulate the notations of Table 5.1 and add new ones (e.g., T, S, and M_i) used for describing the algorithms discussed in the following sections. Finally, in the hybrid architecture, communication through the control network, interfacing the payload subsystem with the wormhole HAAT through the wormhole gateway (WG), is denoted $\xrightarrow{\text{WG}}$ in the protocols.

6.4.1 System roles and setup

The system has four main types of components: forwarding devices, controllers, ANCHOR, and HAAT. While network managers are responsible for controlling the operation of network devices, system administrator are accountable for managing the operation of crucial parts of the system such as the HAAT. Each time a new network device (e.g., forwarding device, controller) is added to the network, it must be registered within ANCHOR before being able to connect to other devices.

HAAT *(T) setup.* The system administrator, in control of HAAT, generates and deploys ANCHOR's master recovery keys $\mathrm{Ke}_{A_{rec}}$ and $\mathrm{Kh}_{A_{rec}}$ in HAAT.

Table 6.2: Summary of notations

Notation	*Description*
T	Hardware-Assisted Anchor of Trust (HAAT)
S	System administrator
M_i	Network Manager i
D_i	Device i (e.g., controller C or Forwarding device F)
H	Cryptographic hash function
MAC	Message Authentication Code algorithm
X, Y	One entity belonging to {A, D_i, M_i, C, F}
Ke_{XY}	Encryption secret key. XY denotes: referring to X and Y entities sharing; or the usage the key refers to
Kh_{XY}	MAC/HMAC secret key. XY denotes: referring to X and Y entities sharing; or the usage the key refers to
E_{XY}	Encryption primitive using secret key Ke_{XY}
[],$HMAC_{XY}$	keyed-Hash MAC of message [] using secret key Kh_{XY}
KDF	Key Derivation Function

ANCHOR's payload *setup.* To start its operation, the payload system will need the recovery keys $Ke_{A_{rec}}$ and $Kh_{A_{rec}}$, both during setup, registration and for the post-compromise recovery steps described ahead. Like happened when manually handled in the basic algorithms, they live essentially off-line, now inside HAAT, and exist on-line for the short periods of setup, registration, or recovery. In R-ANCHOR, they are transacted through the secure ANCHOR to HAAT interface, through the control channel implemented by the wormwhole gateway (WG). When they are needed on-line, HAAT sends them to A through the WG.

The steps of the setup protocol are similar to the basic batch setup protocol presented in the Algorithm 5 , except for the prefix lines, which are now executed within HAAT and ANCHOR interactions, including the master recovery keys, which are installed in ANCHOR through the HAAT. The modified prefix lines are described in the algorithm below. All the remaining lines of the original Algorithm 5 remain the same. Note that [InitA, `GetRecKeys()`] causes T to create new keys.

Network manager setup. Each network manager (denoted M_i) is registered by the system administrator. Each network manager receives a pair of recovery keys computed by A.

Device setup. The first operation to be made after a device D_i is first brought to the network is the setup concerns the establishment of credentials, for secure management access by the network manager. For further details, see the device

Algorithm: Batch setup of R-ANCHOR, M, and initial $\{D_i\}_{i=1}^{n}$ - modified steps.

```
s_a.   $T \xrightarrow{WG} A$   [InitA, WaitForHardReset()].
s_b.   $A \xrightarrow{WG} T$   [InitA, GetRecKeys()].
s_c.   $T \xrightarrow{WG} A$   [InitA, $Ke_{A_{rec}}$, $Kh_{A_{rec}}$].
                      {For each manager $M_j$ and its associated devices $\{D_i\}_{i=1}^{n}$}
...                   execute Algorithm 5 from lines 1 to 11.
```

registration Algorithm 12.

Protocols. In what follows, we discuss the modified protocols required for post-compromise recovery and device registration.

6.4.2 Fully-automatic post-compromise recovery of ANCHOR

As previously explained, when ANCHOR is reinstated after a compromise, it is crucial to have a way to automatically re-establish the secure communication channels between ANCHOR and all participants. Also, one streamlined enough to reduce the window of unavailability of ANCHOR, i.e. to achieve *recoverable operation*, in terms of fault-tolerance (see Part II of [VR12]).

In particular, we consider that when ANCHOR has been compromised by an attacker (e.g., through the exploitation of software vulnerabilities), and has been reinstated by the operator (e.g., by applying software patches and rebuilding servers), the system should have a way to automatically re-establish secure communications between ANCHOR and all other participants, without having to reinstate these components (controllers and forwarding devices in this case, whose shared secrets have become compromised). Though out of the context of the current thesis, this architecture lays the ground for further automation of the image regeneration steps, and even diversification of image sequences for resilience enhancement, in the line of the work of [Sou+10].

Algorithm 11 presents our solution to automatically re-establish the secure communication channels when ANCHOR is compromised. The change lies in that the manual operations and interactions between S and the ANCHOR server are now automated, represented by the prefix lines to the main recovery protocol.

First, once the compromise has been detected, HAAT (T) immediately sends a 'Rejuvenate' command to A (line s_a) prompting it to the forced hard reset that will ensue under the control of HAAT. Rejuvenate is harder than Reboot: it implies the reinstantiation of ANCHOR from a reliable system image (secure and clean-state read-only). Since ANCHOR's master recovery keys $Ke_{A_{rec}}$ and $Kh_{A_{rec}}$ are securely stored in HAAT, they are unknown to an attacker who has stolen all secrets from

Algorithm 11: R-ANCHOR fully-automatic post-compromise recovery

```
s_a.  T $\xrightarrow{WG}$ A   [Rejuvenate, WaitForHardReset()].
s_b.  A $\xrightarrow{WG}$ T   [Rejuvenate, GetRecKeys()].
s_c.  T $\xrightarrow{WG}$ A   [Rejuvenate, $Ke_{A_{rec}}$, $Kh_{A_{rec}}$].
                  {For each manager $M$ and its associated devices $\{D_i\}_{i=1}^{n}$}
...               execute Algorithm 8 from lines 1 to 15.
```

the ANCHOR server. Once rejuvenated, A requests the master recovery keys $Ke_{A_{rec}}$ and $Kh_{A_{rec}}$ to T (line s_b). T sends those keys to ANCHOR (line s_c). Note that [Rejuvenate, `GetRecKeys()`] causes T to reuse and return the existing master keys (unlike the setup protocol where they are created anew). The following steps of the recovery process are the same as presented in lines 1 to 15 of Algorithm 8, as described in Section 6.2.1.

6.4.3 Device Registration

The device registration protocol is modified to fulfill the objective of mitigating insider threats. Having worked on the resilience of ANCHOR, we now turn our attention to the Managers. We recall that we assume that at most f Managers can be compromised at any time.

In the modified device registration protocol we present in Algorithm 12, the ANCHOR serves as a trusted third party (TTP), to vet any registration request made by any network Manager M_k, through an additional f network managers. A set of positive acks from a total of $f+1$ network managers (including the requester), is sufficient evidence that the devices to be registered are legitimate. As shown in the prefix lines, s_1 initiates the registration request from an M_k as usual. ANCHOR broadcasts the reference of the request through all network managers (line s_2). From the set of answers, if not enough positive confirmations exist, the protocol is exited in abort (lines s_3 - s_5).

Otherwise (A receives at least $f + 1$ ACKs), the device registration protocol proceeds as originally described in Algorithm 6 of Section 5.1.5. The protocol ensures resilience to insider threats perpetrated by malicious network managers: no group of up to f, even colluding, malicious managers will be able to register fake devices in the network.

Algorithm 12: R-ANCHOR device registration

```
              {Bootstrap for devices $D_1$ −, $D_n$, using a quorum from Managers $M_1 - M_m$ }
s1.  $M_k \rightarrow A$     [Reg, $M_k$, $E_{AM_k}(\{(D_i,x^i_m)\}^n_{i=1},x^a_m)$],$HMAC_{AM_k}$
                  {A as Trusted Third Party, seeks correct *acks* from $M^f_{j=1}$ }
s2.  $A \rightarrow M_j$     $\forall j \in [1,m]$ [Ack, A, $E_{AM_j}(\{(D_i)\}^n_{i=1},x^j_a)$],$HMAC_{AM_j}$.
                  {For set R of collected ack($M_j$) from Managers $M_1 - M_m$ }
s3.                if $\#R \leq f$ then
s4.  $A \rightarrow M_k$         [Reg, A, $E_{AM_k}$(Reject,$x^a_m$)],$HMAC_{AM_k}$;
s5.  A                 Abort registration and leave.
                   {(else:) Bootstrap for devices $D_1 - D_n$ (contd.)}
...                *execute Algorithm 6 from lines 2 to 19.*
```

6.5 Final remarks

In summary, even though ANCHOR as a root-of-trust, is by design a single-point-of-failure in our system, we have been mitigating the associated risks through Chapters 5 and 6, by guaranteeing:

- Robustness: the probability of component failure is reduced by basic hardening of the design of all mechanisms, functions and algorithms.
- PFS: the compromise of ANCHOR in the current session does not expose *past communications*.
- PCS: the full compromise of ANCHOR does not expose *future communications*, after the recovery of communication channels.
- PQS: ANCHOR will stand up against an attacker with quantum computers, since it only uses symmetric key cryptography.
- Resilience: insider threat mitigation by requiring a quorum of network managers for critical operations such as device registration and association; automatic recovery for fast turnaround and post-compromise security after full compromise or failure.

Chapter 7

Conclusions and Future Work

In this closing Chapter, we start by giving a global perspective of the main contributions of our work. Then we discuss some of its limitations, and point to avenues for future work.

7.1 Summary of Contributions

We recapitulate and briefly elaborate, in hindsight, the main results and contributions of our work:

1. We tested with success the innovative conjecture of using logical centralized services to enforce non-functional properties in SDN. First, by proposing an architectural blueprint of the concept, the generic ANCHOR architecture. Second, by designing, implementing and evaluating a specialization of the architecture for security, probably the hardest of the non-functional property bodies, besides dependability, safety, or QoS. What we have learned from the design process allows us to conjecture that the concept is easily re-applicable to other non-functional property categories.

2. In preparation for the 'security' target, we performed a thorough and systematic study of the main threat vectors of SDN ecosystems, providing the first input towards the existing gaps to fill by the forthcoming architecture. We are also pleased to see that this study has also been of considerable use to the community and fellow researchers.

3. We have designed, implemented and evaluated the ANCHOR architecture specialized to logically-centralized security, providing mechanisms and protocols for essential security services, such as strong entropy, resilient pseudo-random generators, secure device registration, association and recommendation.

4. We have designed, implemented and evaluated KISS, an infrastructure stimulating the generalization of highly-secure control plane communications. By introducing cryptographic mechanisms outperforming widely used alternatives, we have shown that it brings simplicity and performance to the provision of integrity, authenticity and confidentiality.

5. Through several steps, we have embedded mechanisms and protocols for the systematic and incremental mitigation of the risk of logical centralization, from baseline mechanisms — design hardening with robust functions, symmetric cryptography for post-quantum security, embedding of perfect forward secrecy measures in all algorithms, post-compromise security through manual measures — to additional more sophisticated ones in the path of resilience, materialized by the R-ANCHOR extension — mitigation of insider threats, automatic recovery for fast turnaround and post-compromise security after full compromise or failure.

We have shown that, compared to the state-of-the-art in SDN security, our solution preserves at least the same security functionality, but achieves higher levels of implementation robustness, by vulnerability reduction. Even though we prove our point with security, it is worth emphasizing that our contribution is generic enough to inspire further research concerning other non-functional properties (such as dependability or quality-of-service).

7.2 Limitations

Next, we summarize some of the limitations of our work.

1. The ANCHOR architectural concept, albeit perfectly tailored to SDN fabrics, may face some barriers to the application in traditional networks, due to the heterogeneity of the infrastructure and the vertical integration of control and data planes in forwarding devices.

2. Some device deployments may exhibit backward compatibility issues with regard to black-box cryptographic libraries, being hostile to the integration of our high-performance cryptographic libraries such as NaCl. There is however broad enough market offer of devices with white box implementations [SDx14]; [Dri20].

7.3 Future Work

This thesis opens several avenues for future work, which we exemplify below.

1. Investigate how ANCHOR could be used in SDN-based vehicular (V2X) and Internet of Things (IoT) networks.

2. Extended resilience through full Byzantine Fault Tolerance (BFT) and recovery in the entire architecture of ANCHOR. This will help to tolerate any kind of arbitrary fault, including stealth malicious attacks. It is especially important when using ANCHOR in extremely critical IT infrastructures.

3. Analyze ANCHOR's cyclomatic complexity metrics and methods as proposed in [TK14]; [EC16].

4. Even if our results are encouraging in terms of an increase in robustness — an order of magnitude reduction in the number of LOC, and thus of the implied cyclomatic complexity — and some algorithms were formally verified, one future goal should be the formal specification and verification of the whole set of protocols.

Appendix A

Correctness of algorithms of KISS

In this appendix, we introduce the correctness of the iDVV Algorithms 1 and 2 of the KISS framework for secure control plane communications.

A.1 Correctness of Algorithm 1

Theorem 1. If the initial values of seed and key are indistinguishable from random, then the resulting initial idvv (line 2) is indistinguishable from random.

Proof: The *seed* and *key* are, by assumption (Section 4.1) of the availability of robust sources of pseudo-random number generators in the central services (which generate the former), indistinguishable from random. In consequence, and assuming that H is a strong hash function, the output of H(seed || key) will thus be indistinguishable from random. □

Theorem 2. Any execution of the function `H(seed` || `key)` with the same input values seed and key, produces the same output value (`idvv` in line 2).

Proof: Proof that Algorithm 1 is deterministic follows trivially from the deterministic nature of hash functions. □

A.2 Correctness of Algorithm 2

Lemma 1. If the seed and idvv are indistinguishable from random, then the resulting new seed (line 2) is indistinguishable from random.

Proof: We start by proving the result of the run with the initial values of *seed* and *idvv*. The initial *seed* is, by assumption (Section 4.1) of the availability of robust sources of pseudo-random number generators in the central services (which generate the former), indistinguishable from random. Theorem 1 states that the initial *idvv* is indistinguishable from random. In consequence, with a similar argumentation of the proof of Theorem 1, and assuming that H is a strong hash function, the output of H(seed || idvv) will be indistinguishable from random.

Now we recurse the argumentation, to show that the proof is valid for any input values of *seed* and *idvv*. A *new seed* was just proven to be indistinguishable from random. A *new idvv* is proven below in Theorem 3 to be indistinguishable from random. Feeding these into the argumentation above, we generalise the proof $\forall$ *seed* and *idvv*. □

Theorem 3. If the seed and key are indistinguishable from random, then the resulting new idvv (line 3) is indistinguishable from random.

Proof: Lemma 1 establishes that *seed*, output by line 2 and thus used as input in line 3, is indistinguishable from random. The *key* is, by assumption of the availability of robust sources of pseudo-random number generators in the central services (which generate the former), indistinguishable from random.

We start by proving the result of the run with the initial value *idvv*. Theorem 1 states that the initial *idvv* is indistinguishable from random. In consequence, with a similar argumentation of the proof of Theorem 1, and assuming that H is a strong hash function, the output of H(seed || key) will be indistinguishable from random.

Now we recurse the argumentation, to show that the proof is valid for any values of *idvv*. Any *new idvv* was just proven to be indistinguishable from random. In some next run, it will pair with *key*, by nature indistinguishable from random, and with any new *seed*, proven by Lemma 1 to be indistinguishable from random. Feeding these into the argumentation above, we generalise the proof $\forall$ *key*, *seed* and *idvv*.

In other words, the newly generated iDVV is an indistinguishable from random value that can be safely used as an authentication or authorization code, secret key, random nonce, and so forth. □

Lemma 2. Any execution of the function `H(seed || idvv)` with the same input values seed and idvv, produces the same output value (`seed` in line 2).

Proof: Proof that the function is deterministic follows trivially from the deterministic nature of hash functions. □

Lemma 3. Any execution of the function `H(seed || key)` with the same input values seed and key produces the same output value (`idvv` in line 3).

Proof: Proof that the function is deterministic follows trivially from the deterministic nature of hash functions. □

Theorem 4. Any execution of Algorithm 2 with the same input values seed, idvv and key produces the same output value (`idvv` in line 3).

Proof: Proof that Algorithm 2 is deterministic follows trivially from Lemma 2 and 3: since the two functions are executed in a row, and the *seed* output of line 2 used as input in line 3 is deterministic (Lemma 2), it satisfies the conditions of Lemma 3 for determinism. □

Appendix B

Operations and security analysis of ANCHOR

In this appendix, we introduce the stages of ANCHOR in Section B.1 and the correctness of algorithms in Sections B.3 and B.4. Following, we also discuss the conformance of ANCHOR to security requirements of SDN, as proposed by ONF, in Section B.5.

B.1 The three stages of ANCHOR

Figure B.1 illustrates the three stages of ANCHOR, namely, setup, normal operation, and post-compromise recovery. After setup and post-compromise recovery, it goes to normal operation. The details of normal operation (e.g., device registration and association) are discussed in Section 5.1.5.The complete post-compromise recovery protocol is presented in Section 6.4.2.

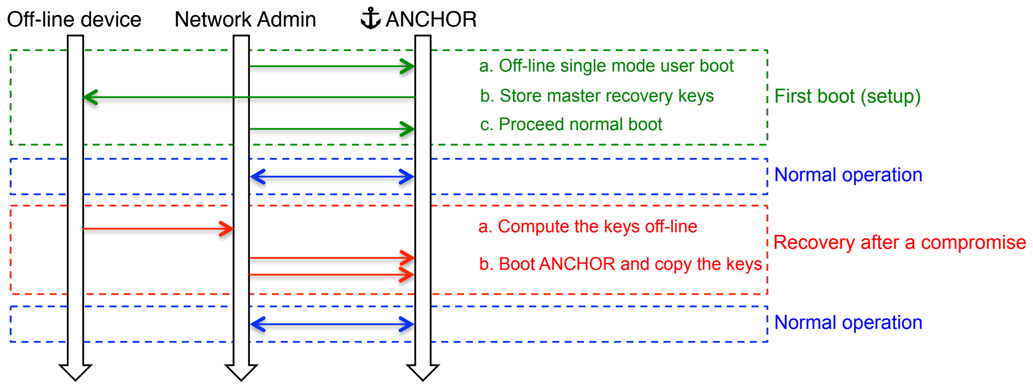
Off-line device
Network Admin
⚓ANCHOR
a. Off-line single mode user boot
b. Store master recovery keys
c. Proceed normal boot
First boot (setup)
Normal operation
a. Compute the keys off-line
b. Boot ANCHOR and copy the keys
Recovery after a compromise
Normal operation

Figure B.1: Setup, normal operation and PCR

B.1.1 Setup

During the setup, three things happen:

1. *Off-line single mode user boot.* The first boot should be off-line to generate the master recovery keys safely. These keys need to be generated a single time and stored in a safe place.

2. *Store master recovery keys.* The network admin should store the master recovery keys, for future use in case of a compromise, in an offline device (e.g., USB stick). This device should be kept as secure as possible.

3. *Normal boot.* After generating and safely storing the master recovery keys, the network admin can proceed with the normal boot of ANCHOR. This boot is going to bring up all services and functionalities of ANCHOR and put it online, ready for use.

B.1.2 Normal operation

The normal operation represents the phase in which ANCHOR should be most of the time, i.e., online and fully operational. The normal operation phase can happen after a first boot (setup phase) or after a recovery from a compromised state.

B.1.3 Recovery after a compromise

To recover ANCHOR after a compromise, the network admin has to:

1. *Compute the keys off-line* using the master recovery keys. The network admin must recursively generate the network manager recovery keys and the device recovery keys. These are special purpose keys used to automatically and safely recover communications between ANCHOR and all other entities, i.e., without needing additional procedures such as device re-registration. For more details on how it works, see Section 6.4.2.

2. *Boot* ANCHOR *and copy the keys.* After recursively computing the master recovery keys of managers and devices, the network admin should proceed with a normal boot of the system and copy these keys into ANCHOR.

B.2 Correctness of Algorithm 3

We argue about the properties of Algorithm 3, as a source of strong entropy.

Lemma 4. If the initial values of rand_bytes() and H(data) are indistinguishable from random, then the resulting initial external entropy (e_entropy - line 2) is indistinguishable from random. Then, the initial internal entropy (i_entropy - line 3) will be also indistinguishable from random.

Proof: Assuming that rand_bytes() uses one of the strongest pools of entropy of an operating system, such as /dev/urandom, the outcome of this function call will be indistinguishable from random. Assuming that H is a cryptographically strong hashing function, the output of H(data) will be indistinguishable from random for every different input data. Consequently, the XOR operation between rand_bytes() and H(data) will result in an indistinguishable-from-random initial e_entropy. Following, the XOR operation between rand_bytes() and e_entropy will result in an indistinguishable-from-random initial i_entropy. In other words, both internal and external entropy are initialized with indistinguishable-from-random values. □

Lemma 5. If P_i, P_j, and i_entropy are indistinguishable from random, then the updated external entropy (e_entropy - line 5) will be indistinguishable from random.

Proof: As discussed before, the pools of entropy P_i and P_j contain unpredictable events of external sources of entropy, such as network traffic and idleness of links. Thus, assuming that H is a cryptographically strong hashing function, then the output of H($P_i||P_j$) will be indistinguishable from random. Lemma 4 shows that the internal entropy (*i_entropy*) is indistinguishable from random. In consequence, the updated external entropy (*e_entropy* - line 5), which is the output of an XOR operation between two indistinguishable-from-random values, will be indistinguishable from random. □

Lemma 6. If the initial value of rand_bytes() is indistinguishable from random, then the resulting internal entropy (i_entropy - line 7) is indistinguishable from random.

Proof: The proof of Lemma 4 establishes that the output of *rand_bytes*() is indistinguishable from random. Additionally, *E_counter* is an internal counter not known by external entities. Therefore, assuming that H is a cryptographically strong hashing function, then *i_entropy* output by H(*rand_bytes*()||*E_counter*) will be indistinguishable from random. □

Theorem 5. If e_entropy and i_entropy are indistinguishable from random, then the resulting entropy returned by entropy_get (line 8) will be indistinguishable from random.

Proof: Lemmata 4 and 5 show that the initial and updated external entropy are indistinguishable from random. Lemma 6 has shown that the internal entropy generated in line 7 is indeed indistinguishable from random. As a consequence, *entropy*, as the output of an XOR operation between *i_entropy* and *e_entropy* (line 8) will be indistinguishable from random. This proves that Algorithm 3 satisfies the property *Strong Entropy*. □

B.3 Correctness of Algorithm 4

We argue about the properties of Algorithm 4, as a source of indistinguishable-from-random pseudo-random values.

Lemma 7. If entropy_get() returns an indistinguishable-from-random value, then the initial *seed* (line 2), *counter* (line 3) and pseudo random value (*nprd* - line 4) will be indistinguishable from random.

Proof: Theorem 5 establishes that the output of *entropy_get*() is indistinguishable from random. Thus, both the *seed* and the first *nprd* will be indistinguishable from random. Similarly, the function *long_uint*() (using as input *entropy_get*() - line 3), which, on most architectures, uses 64 bits to represent an unsigned long int, will return the value *counter*, indistinguishable from random. □

Lemma 8. If entropy_get() returns a value indistinguishable from random, then the refreshed PRG internal state (lines 6-8) will lead to indistinguishable from random values for *seed*, *counter* and *nprd*.

Proof: The proof follows the same argumentation of the proof of Lemma 7, for *seed* and *counter*. As for *nprd*, assuming that neither the seed or counter are known outside the PRG, and assuming that H is a cryptographically strong hashing function, then the output of H, having as input a concatenation of the new *seed*, current *nprd*, and new *counter*, will be indistinguishable from random. □

Theorem 6. If seed and nprd are indistinguishable-from-random values, then the next nprd returned by PRG_next (line 12) will be indistinguishable from random.

Proof: Lemmata 7 and 8 established that both the *seed* and *nprd* are always indistinguishable from random, since the initial state. Assuming that HMAC is a cryptographically strong message authentication code primitive, and that the counter is not known outside of the PRG, then the output of HMAC, keyed by *seed* and having as input a concatenation of *nprd* and *counter*, will be indistinguishable from random. This proves that Algorithm 4 satisfies property *Robust PRG*. □

B.4 Correctness of Algorithm 7

We now formalize and prove the properties of Algorithm 7.

As a result of the registration process, ANCHOR keeps lists of registered devices and controllers, and lists of the controllers each device is authorized to associate with.

Proposition 1. Any device F can only associate to a controller C authorized by the ANCHOR.

Proof: Forwarding devices will be able to associate only to controllers listed in the CList(F) provided by A (step 2 of Algorithm 7), since if F tries to associate with a non-authorized controller (for F), A will not proceed past step 4 after being contacted by that controller, aborting the association. On the other hand, a rogue controller posing to F as authorised in reply to step 3, cannot jump to step 6 and invent an association key AiD that convinces F, since it does not know x_f. This proves that Algorithm 7 satisfies property *Controller Authorization.* □

Proposition 2. Any device F can associate to some controller, only if F is authorized by the ANCHOR.

Proof: Only if a device F is legitimate, i.e. it is in the list of registered devices, will it be able to associate to some registered controller. A will not proceed past step 1 of Algorithm 7 after being contacted by a rogue device, aborting the association. On the other hand, a rogue device posing to C as legitimate and authorised in step 3, will make C proceed with step 4, indeed, but the request will be rejected by A, since $E_F()$ is not recognisable by A, corresponding to no shared key with a legitimate device. The replay of an old (but legitimate) encrypted $E_F()$ request in step 3 will also fail, since it is bound to the (current) nonces. This proves that Algorithm 7 satisfies property *Device Authorization.* □

Proposition 3. At the end of Algorithm 7 execution, the association ID (AiD) is only known to F and C.

Proof: A creates AiD in step 5, and forgets about it after sending it to C (see Section 5.1.6). AiD is sent from A to C, encrypted both by Ke_{AF} and Ke_{AC}, keys shared by A only with F and C respectively. C trusts it came from A, due to the HMAC, so the two encrypted blocks should contain the same AiD value, and sends the AiD under Ke_{AF} encryption to F. So, at the end of the execution of the algorithm, both F and C, and only them, hold AiD. This proves that Algorithm 7 satisfies property *Association ID Secrecy.* □

Proposition 4. At the end of Algorithm 7 execution, the seed ($SEED$) is only known to F and C.

Proof: C creates $SEED$ in step 7. $SEED$ is sent from C to F, encrypted by K_AiD, association key known only to C and F, as per Proposition 3. C trusts that F, and only F, has the same $SEED$ sent, when it receives back from F the XOR of $SEED$ with the current nonce x_g encrypted with AiD, since (as per Proposition 3) only F could have opened the encryption of $SEED$ with AiD in the first place, and encrypt the reply. This proves that Algorithm 7 satisfies property *Seed Secrecy.* □

B.5 Meeting ONF's security requirements

Several security requirements should be fulfilled in control plane communications. Most of these requirements are enumerated in ONF's best practice recommendations [ONF15]. In this appendix we go through the eleven (out of twenty four) such requirements that are addressed by the ANCHOR, iDVV and NaCl.

Both communicating devices should be authenticated (REQ 4.1.1). Using our ANCHOR, all devices have to be properly registered and authenticated before proceeding with any other operation.

Operations (e.g., association) of components should be authorized (REQ 4.1.2). The ANCHOR needs to explicitly authorize associations between any two devices. Each association has a unique identification.

Devices should agree upon the security (e.g., key materials) associations (REQ 4.1.3). By using the ANCHOR and its mechanisms, such as the source of strong entropy, we ensure strong key materials. The iDVV mechanism is initialized by the two communicating devices once the association has been authorized by the ANCHOR.

Integrity of packets should be ensured (REQ 4.1.4). We provide integrity and authenticity of packets through message authentication codes. By default, we generate one iDVV per packet, providing strong security.

Each device should have a unique ID and other devices should be able to verify the identity (REQ 4.2.1). Devices are uniquely identified by the ANCHOR. The unique IDs are associated to the devices as soon as they are registered within the ANCHOR.

Issues related to the lifecycle of IDs should be managed, such as generation, distribution, maintenance, and revocation (REQ 4.2.2). The ANCHOR provides the services required for managing device IDs. IDs are assigned to devices during the registration phase. Revocation can be done by network administrators at any time.

Devices should be able to verify the integrity of each message (REQ 4.4.4). Any two communicating devices are able to verify the integrity of each message through message authentication codes.

Amplification effects should be taken into account, i.e., attackers should not be able to perform reflection attacks (REQ 4.4.5). We use requests and replies of the same size between devices and the ANCHOR, which avoids reflection attacks.

Automated key/credential management should be implemented by default, allowing generation, distribution, and revocation of security credentials (REQ 4.8.3). We have in place automated mechanisms for refreshing credentials, such as refresh the iDVV's seed using the ANCHOR's source of strong entropy.

Data confidentiality, integrity, freshness and authenticity are ensured by the integrated device verification value. iDVVs are used to encrypt data and generate message authentication codes. Additionally, iDVVs can also be used as nonces, ensuring data freshness.

Availability is ensured by recommending multiple controllers to the forwarding devices. This is one of the essential tasks of the ANCHOR.

Lastly, it is also worth mentioning that whilst we do not meet all security requirements of ONF's guidelines, we do meet the fundamental ones with regard to security. For instance, requirements such as REQ 4.4.2, REQ 4.4.3, REQ 4.7.1, REQ 4.7.2, and REQ 4.7.3 [ONF15] are not yet covered by our architecture and protocols. However, most of these requirements are related to rate control of messages, additional signaling messages for dealing with future network attack types, and accountability and traceability. Such kind of requirements can be added (in the future) without impairing our conceptual architecture. In fact, some of these requirements, such as rate control of messages, are technical, rather than conceptual, which can be addressed with the right amount of engineering.

B.6 Security Analysis of ANCHOR

Disclaimer:

The content of this Section is mainly a contribution from Jiangshan Yu.

We provide formal machine-checked verification of the core security properties of ANCHOR, using the TAMARIN prover. In particular, we formalise the core protocols of ANCHOR, including device registration protocol, device association protocol, and post-compromise recovery protocol, through symbolic modeling. In addition, for each of the protocols, we verify its correctness, message confidentiality, and perfect forward secrecy (PFS). Moreover, we additionally verify the post-compromise security of ANCHOR with the post-compromise recovery protocol.

The full model contains 1712 lines of code. In total, we have proved 33 properties — 23 of them are helper lemmas for the theorem prover to understand ANCHOR better; 4 lemmas are sanity proofs which check the correctness of our protocols and their formalization; and 6 main security properties that ensure the message confidentiality, perfect forward secrecy, and post-compromise security of ANCHOR. We provide all input files and complete formal model required to understand and reproduce our security analysis at [Yu18].

B.6.1 Security properties

ANCHOR achieves both classical security properties and novel security properties. In a classical sense, the confidentiality of communications between any two devices is guaranteed. In particular, ANCHOR also provides perfect forward secrecy, namely if a device is compromised, then all communications of this device in the past are still secure.

For the novel security guarantee, as mentioned before, rather than assuming the trusted party cannot be compromised, such as CAs in X.509 PKI or the KDC in Kerberos, we also consider that ANCHOR might be compromised. In this case, we assume that there are means to detect that the compromise has happened, and then the system can be recovered through our post-compromise recovery protocol, which also guarantees perfect forward security, when ANCHOR is compromised and recovered.

B.6.2 Formal analysis

We analyze the security properties of the protocol using Tamarin [Mei+13]. The Tamarin prover is a symbolic analysis tool that can prove properties of security protocols for an unbounded number of instances and supports reasoning about protocols with mutable global state, which makes it suitable for our protocols. Protocols are specified using multi-set rewriting rules, and properties are expressed in a guarded fragment of first order logic that allows quantification over timepoints.

Tamarin is capable of automatic verification in many cases, and it also supports interactive verification by manual traversal of the proof tree. If the tool terminates without finding a proof, it returns a counter-example. Counter-examples are given as so-called dependency graphs, which are partially ordered sets of rule instances that represent a set of executions that violate the property. Counter-examples can then be used to refine the model, and give feedback to the implementer and designer.

B.6.3 Modeling aspects

As explained, we consider four protocol roles in ANCHOR, namely A (ANCHOR), M (network Manager), F (Forwarding device), and C (Controller). To simplify our model, we consider an additional role D (Device) to represent any kind of network device, when it is irrelevant to distinguish its type (i.e., F or C).

We model the above protocol roles by a set of rewrite rules. Our modeling of the roles follows the typical Tamarin models, and directly corresponds to the algorithm descriptions in the previous sections. Specifically, each rewrite rule typically models receiving a message, taking an appropriate action, and sending a response message. TAMARIN provides built-in support for a Dolev-Yao style network attacker, i.e., one who is in full control of the network. We also specify rules that enable the attacker to compromise ANCHOR and/or any device in the network, and learn all of their session keys.

B.6.4 Proof goals

We state several proof goals as specified in TAMARIN's syntax. Since TAMARIN's property specification language is a fragment of first-order logic, it contains logical connectives (`|`, `&`, `==>`, `not`, ...) and quantifiers (`All`, `Ex`). In TAMARIN, proof goals are marked as `lemma`. The `#`-prefix is used to denote timepoints, and "`E @ #i`" expresses that the event E occurs at timepoint i. Due to the space limitation, we only present a set of examples selected from our full model, to explain the core ideas. We refer the reader to the full model and detailed proof results available at [Yu18].

The first example goal is a check for executability that ensures that our model allows for the successful transmission of a message. The following example, which is a correctness lemma in the device registration protocol, shows how it is encoded in our proof.

```
lemma protocol_correctness [use_induction]:
 exists-trace
  "Ex A D Did k keAD #i1.
     SendSec(A, D, Did,k, keAD) @ i1"
```

The property holds if the TAMARIN model exhibits a behaviour in which a device D of any type with unique identity Did can successfully exchange with ANCHOR A a message k encrypted by using a secret keAD shared between D and A. This property mainly serves as a sanity check on the model. If it does not hold, it would mean our model does not model the normal message flow, which could indicate a flaw in the model. TAMARIN automatically proves this property and generates the expected trace in the form of a graphical representation of the rule instantiations and the message flow. We additionally proved several other sanity-checking properties to minimize the risk of modeling errors.

The second example goal is the core secrecy property with respect to a classical attacker. When a controller C is associated with a forwarding device F, then the following expresses that unless the attacker compromises either C or F, he cannot learn any messages exchanged between them. Note that `K(m)` is a special event that denotes that the attacker knows m at this time.

```
lemma message_secrecy [use_induction]:
 "All C F Did1 Did2  k seed #i.
```

```
   /* If a message k is exchanged */
      ( SendSec(C, F, Did1, Did2, k, seed) @ #i &
   /* without the adversary compromising any device */
      not (Ex #j.
        Compromise_Device(C, F, Did1, Did2, seed) @ #j)
      ) ==>
   /* then the adversary cannot know k */
     not ( Ex #j. K(k) @ #j) "
```

Tamarin also proves this property automatically. The above result implies that if a forwarding device F with identity Did1 and a controller C with identity Did2 has exchanged a message k encrypted under a shared seed, and the attacker did not compromise any device *at any time*, then the attacker will not learn k.

Similarly, the following example expresses the PFS for the communications between two devices.

```
lemma message_forward_secrecy [use_induction]:
 "All C F Did1 Did2  k seed #i.
      ( SendSec(C, F, Did1, Did2, k, seed) @ #i &
        not (Ex #j seed2.
         Compromise_Device(C, F, Did1, Did2, seed2) @ j &
        j<i)
      )
      ==>
      ( /* then the adversary cannot know k */
        not ( Ex #j. K(k) @ #j)
      )
 "
```

Tamarin proves this property automatically, and the result additionally implies that the message is secure if the attacker did not compromise any device *before the current communication session.*

The final example property encodes the post-compromise security guarantees provided by ANCHOR. In this example, if ANCHOR was compromised, and then recovered through our protocol, then the confidentiality of communications between ANCHOR and forwarding device F is guaranteed.

```
lemma message_secrecy_after_recovery [use_induction]:
 "All A M F C Did k enckey #i1 #i2 #i3.
          (Comppromised_A(A) @#i1 &
           Recovery_Done(A,M,F,C)@ #i2 & i1<i2 &
           SendSec(A, F, Did, k, enckey) @ #i3 & i2<i3)
      ==>
     /* then the adversary cannot know k */
           not ( Ex #i4. K(k) @ #i4)
 "
```

The property states that if ANCHOR was compromised at session $i1$, and the recovery action has been completed afterwards at session $i2$, then the confidentiality of message k exchanged in a later time between A and forwarding device F is guaranteed.

The above properties are all proven automatically by the TAMARIN prover on a PC[1] within 15 min.

[1]Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz, 16GB memory.

Bibliography

[AAP19] A. L. Aliyu, A. Aneiba, and M. Patwary. “Secure Communication between Network Applications and Controller in Software Defined Network”. In: *IEEE 18th International Symposium on Network Computing and Applications (NCA)*. Sept. 2019, pp. 1–8. DOI: 10.1109/NCA.2019.8935066. URL: https://ieeexplore.ieee.org/abstract/document/8935066.

[AAS14] Markku Antikainen, Tuomas Aura, and Mikko Särelä. “Spook in Your Network: Attacking an SDN with a Compromised OpenFlow Switch”. English. In: *Secure IT Systems*. Ed. by Karin Bernsmed and Simone Fischer-Hübner. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 229–244. DOI: 10.1007/978-3-319-11599-3_14. URL: http://dx.doi.org/10.1007/978-3-319-11599-3_14.

[ABA17] A. L. Aliyu, P. Bull, and A. Abdallah. “A Trust Management Framework for Network Applications within an SDN Environment”. In: *31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. 2017, pp. 93–98. DOI: 10.1109/WAINA.2017.100. URL: https://ieeexplore.ieee.org/abstract/document/7929660.

[Aca+17] Y. Acar, M. Backes, S. Fahl, S. Garfinkel, D. Kim, M. L. Mazurek, and C. Stransky. “Comparing the usability of cryptographic APIs”. In: *Proceedings of the 38th IEEE Symposium on Security and Privacy*. 2017, pp. 154–171. URL: https://ieeexplore.ieee.org/abstract/document/7958576.

[Ace+18] Giuseppe Aceto, Alessio Botta, Pietro Marchetta, Valerio Persico, and Antonio Pescapé. “A comprehensive survey on internet outages”. In: *Journal of Network and Computer Applications* 113 (2018), pp. 36–63. DOI: https://doi.org/10.1016/j.jnca.2018.03.026. URL: http://www.sciencedirect.com/science/article/pii/S1084804518301139.

[ACW16] O. I. Abdullaziz, Y. J. Chen, and L. C. Wang. “Lightweight Authentication Mechanism for Software Defined Network Using Information Hiding”. In: *IEEE Global Communications Conference (GLOBECOM)*. 2016, pp. 1–6. DOI: 10.1109/GLOCOM.2016.7841954. URL: https://ieeexplore.ieee.org/abstract/document/7841954.

[Adr+15a] David Adrian et al. “Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice”. In: *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*. CCS ’15. Denver, Colorado, USA: ACM, 2015, pp. 5–17. DOI: 10.1145/2810103.2813707. URL: http://doi.acm.org/10.1145/2810103.2813707.

[Adr+15b] David Adrian et al. “Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice”. In: *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*. CCS ’15. Denver, Colorado, USA: ACM, 2015, pp. 5–17. DOI: 10.1145/2810103.2813707. URL: http://doi.acm.org/10.1145/2810103.2813707.

[AF13] Cyril Arnaud and Pierre-Alain Fouque. “Timing Attack against Protected RSA-CRT Implementation Used in PolarSSL”. English. In: *Topics in Cryptology - CT-RSA 2013*. Ed. by Ed Dawson. Vol. 7779. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 18–33. DOI: 10.1007/978-3-642-36095-4_2. URL: http://dx.doi.org/10.1007/978-3-642-36095-4_2.

[Ahm+15] Ijaz Ahmad, Suneth Namal, Mika Ylianttila, and Andrei Gurtov. “Security in software defined networks: A survey”. In: *IEEE Communications Surveys & Tutorials* 17.4 (2015), pp. 2317–2346. DOI: 10.1109/COMST.2015.2474118. URL: https://ieeexplore.ieee.org/abstract/document/7226783.

[Akh+15] Adnan Akhunzada, Ejaz Ahmed, Abdullah Gani, Muhammad Khurram Khan, Muhammad Imran, and Sghaier Guizani. “Securing software defined networks: taxonomy, requirements, and open issues”. In: *IEEE Communications Magazine* 53.4 (2015), pp. 36–44. DOI: 10.1109/MCOM.2015.7081073. URL: https://ieeexplore.ieee.org/abstract/document/7081073.

[Akh+16] Adnan Akhunzada, Abdullah Gani, Nor Badrul Anuar, Ahmed Abdelaziz, Muhammad Khurram Khan, Amir Hayat, and Samee U. Khan. “Secure and dependable software defined networks”. In: *Journal of Network and Computer Applications* 61 (2016), pp. 199–221. DOI: https://doi.org/10.1016/j.jnca.2015.11.012. URL: http://www.sciencedirect.com/science/article/pii/S1084804515002842.

[Akh17] Haji Akhundov. *Design & development of public-key based authentication architecture for IoT devices using PUF*. Tech. rep. TUDelf, 2017. URL: http://resolver.tudelft.nl/uuid:58ad76d8-4552-461e-aa61-54299d021bd1.

[Alb+15] Martin R Albrecht, Davide Papini, Kenneth G Paterson, and Ricardo Villanueva-Polanco. *Factoring 512-bit RSA moduli for fun (and a profit of $9,000)*. 2015. URL: https://pdfs.semanticscholar.org/713c/2e84b69fa865a49ba861297d118f97fb4c7c.pdf.

[Alk+14] Hasan Alkhatib, Paolo Faraboschi, Eitan Frachtenberg, Hironori Kasahara, Danny Lange, Phil Laplante, Arif Merchant, Dejan Milojicic, and Karsten Schwan. *IEEE CS 2022 Report (DRAFT)*. Tech. rep. IEEE Computer Society, 2014. URL: https://ieeecs-media.computer.org/assets/pdf/2022Report.pdf.

[Alm+13] J. Bacelar Almeida, Manuel Barbosa, Jorge S. Pinto, and Barbara Vieira. "Formal verification of side-channel countermeasures using self-composition". In: *Science of Computer Programming* 78.7 (2013). Special section on Formal Methods for Industrial Critical Systems (FMICS 2009 + FMICS 2010) & Special section on Object-Oriented Programming and Systems (OOPS 2009), a special track at the 24th ACM Symposium on Applied Computing, pp. 796–812. DOI: http://dx.doi.org/10.1016/j.scico.2011.10.008. URL: http://www.sciencedirect.com/science/article/pii/S0167642311001857.

[Alv+17] R. Alvizu, G. Maier, N. Kukreja, A. Pattavina, R. Morro, A. Capello, and C. Cavazzoni. "Comprehensive survey on T-SDN: Software-defined networking for transport networks". In: *IEEE Communications Surveys & Tutorials* 19.4 (2017), pp. 2232–2283. DOI: 10.1109/COMST.2017.2715220. URL: https://ieeexplore.ieee.org/abstract/document/7947156.

[ALV08] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. "A Scalable, Commodity Data Center Network Architecture". In: *SIGCOMM Comput. Commun. Rev.* 38.4 (Aug. 2008), pp. 63–74. DOI: 10.1145/1402946.1402967. URL: http://doi.acm.org/10.1145/1402946.1402967.

[AM18] N. Aldaghri and H. Mahdavifar. "Fast Secret Key Generation in Static Environments Using Induced Randomness". In: *IEEE Global Communications Conference (GLOBECOM)*. Dec. 2018, pp. 1–6. DOI: 10.1109/GLOCOM.2018.8647945. URL: https://ieeexplore.ieee.org/abstract/document/8647945.

[AMA19] M. Alsaeedi, M. M. Mohamad, and A. A. Al-Roubaiey. “Toward Adaptive and Scalable OpenFlow-SDN Flow Control: A Survey”. In: *IEEE Access* 7 (2019), pp. 107346–107379. DOI: 10.1109/ACCESS.2019.2932422. URL: https://ieeexplore.ieee.org/abstract/document/8784036.

[Amb+15] M. Ambrosin, M. Conti, F. Gaspari, and R. Poovendran. “LineSwitch: Efficiently Managing Switch Flow in Software-Defined Networking While Effectively Tackling DoS Attacks”. In: *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*. ASIA CCS ’15. Singapore, Republic of Singapore: ACM, 2015, pp. 639–644. DOI: 10.1145/2714576.2714612. URL: http://doi.acm.org/10.1145/2714576.2714612.

[ANH17] Ahmad Aseeri, Nuttapong Netjinda, and Rattikorn Hewett. “Alleviating Eavesdropping Attacks in Software-defined Networking Data Plane”. In: *Proceedings of the 12th Annual Conference on Cyber and Information Security Research*. CISRC ’17. Oak Ridge, Tennessee: ACM, 2017, 1:1–1:8. DOI: 10.1145/3064814.3064832. URL: http://doi.acm.org/10.1145/3064814.3064832.

[Arb+16] R. K. Arbettu, R. Khondoker, K. Bayarou, and F. Weber. “Security analysis of OpenDaylight, ONOS, Rosemary and Ryu SDN controllers”. In: *17th International Telecommunications Network Strategy and Planning Symposium (Networks)*. 2016, pp. 37–44. DOI: 10.1109/NETWKS.2016.7751150. URL: https://ieeexplore.ieee.org/abstract/document/7751150.

[ARS18] R. Amin, M. Reisslein, and N. Shah. “Hybrid SDN Networks: A Survey of Existing Approaches”. In: *IEEE Communications Surveys Tutorials* 20.4 (Oct. 2018), pp. 3259–3306. DOI: 10.1109/COMST.2018.2837161.

[AS17] B. Agborubere and E. Sanchez-Velazquez. “OpenFlow Communications and TLS Security in Software-Defined Networks”. In: *IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. June 2017, pp. 560–566. DOI: 10.1109/iThings-GreenCom-CPSCom-SmartData.2017.88.

[ASG19] ASG Information Technologies. *Best Practices for Server Room Security*. 2019. URL: https://www.asgct.com/best-practices-for-server-room-security/.

[AvW18] A. Abdou, P. C. van Oorschot, and T. Wan. “Comparative Analysis of Control Plane Security of SDN and Conventional Networks”. In: *IEEE Communications Surveys Tutorials* 20.4 (Oct. 2018), pp. 3542–3559. DOI: 10.1109/COMST.2018.2839348. URL: https://ieeexplore.ieee.org/abstract/document/8362609.

[AW19] O. I. Abdullaziz and L. Wang. “Mitigating DoS Attacks against SDN Controller Using Information Hiding”. In: *IEEE Wireless Communications and Networking Conference (WCNC)*. Apr. 2019, pp. 1–6. DOI: 10.1109/WCNC.2019.8885764. URL: https://ieeexplore.ieee.org/abstract/document/8885764.

[AY18] M. Alhanahnah and Q. Yan. “Towards best secure coding practice for implementing SSL/TLS”. In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. Apr. 2018, pp. 1–6. DOI: 10.1109/INFCOMW.2018.8407011. URL: https://ieeexplore.ieee.org/abstract/document/8407011.

[Azi+18] N. A. Aziz, T. Mantoro, M. A. Khairudin, and A. F. b. A. Murshid. “Software Defined Networking (SDN) and its Security Issues”. In: *International Conference on Computing, Engineering, and Design (ICCED)*. Sept. 2018, pp. 40–45. DOI: 10.1109/ICCED.2018.00018. URL: https://ieeexplore.ieee.org/abstract/document/8691107.

[BAM10] Theophilus Benson, Aditya Akella, and David A. Maltz. “Network Traffic Characteristics of Data Centers in the Wild”. In: *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*. IMC ’10. Melbourne, Australia: ACM, 2010, pp. 267–280. DOI: 10.1145/1879141.1879175. URL: http://doi.acm.org/10.1145/1879141.1879175.

[Bar+17] R. Barrett, A. Facey, W. Nxumalo, J. Rogers, P. Vatcher, and M. St-Hilaire. “Dynamic Traffic Diversion in SDN: testbed vs Mininet”. In: *International Conference on Computing, Networking and Communications (ICNC)*. Jan. 2017, pp. 167–171. DOI: 10.1109/ICCNC.2017.7876121. URL: https://ieeexplore.ieee.org/abstract/document/7876121.

[Bar19] Susan Baranowski. *How Secure are the Root DNS Servers?* 2019. URL: https://www.sans.org/reading-room/whitepapers/dns/secure-root-dns-servers-991.

[Bas+10] Lawrence E. Bassham III et al. *SP 800-22 Rev. 1a. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications.* Tech. rep. Gaithersburg, MD, United States, 2010. URL: https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final.

[Bas+13] Paul Baskett, Yi Shang, Wenjun Zeng, and Brandon Guttersohn. "SDNAN: Software-defined networking in ad hoc networks of smartphones". In: *IEEE Consumer Communications and Networking Conference (CCNC).* IEEE. 2013, pp. 861–862. DOI: 10.1109/CCNC.2013.6488568. URL: https://ieeexplore.ieee.org/abstract/document/6488568.

[BCS13] Kevin Benton, L. Jean Camp, and Chris Small. "OpenFlow vulnerability assessment". In: *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking.* HotSDN '13. Hong Kong, China: ACM, 2013, pp. 151–152. DOI: 10.1145/2491185.2491222. URL: http://doi.acm.org/10.1145/2491185.2491222.

[BDK17] Johannes Behl, Tobias Distler, and Rüdiger Kapitza. "Hybrids on Steroids: SGX-Based High Performance BFT". In: 2017, pp. 222–237. DOI: 10.1145/3064176.3064213. URL: http://doi.acm.org/10.1145/3064176.3064213.

[Bed16] Ann Bednarz. *Top reasons for network downtime.* 2016. URL: https://www.networkworld.com/article/3142838/infrastructure/top-reasons-for-network-downtime.html.

[BEI19] Jaouad Benabbou, Khalid Elbaamrani, and Noureddine Idboufker. "Security in OpenFlow-based SDN, opportunities and challenges". In: *Photonic Network Communications* 37.1 (Feb. 2019), pp. 1–23. DOI: 10.1007/s11107-018-0803-7. URL: https://doi.org/10.1007/s11107-018-0803-7.

[Ben+10] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. "Understanding Data Center Traffic Characteristics". In: *SIGCOMM Comput. Commun. Rev.* 40.1 (Jan. 2010), pp. 92–99. DOI: 10.1145/1672308.1672325. URL: http://doi.acm.org/10.1145/1672308.1672325.

[Ber+14] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar. "ONOS: Towards an Open, Distributed SDN OS". In: *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking.* HotSDN '14. Chicago, Illinois, USA: Association for Computing

Machinery, 2014, pp. 1–6. DOI: 10.1145/2620728.2620744. URL: https://doi.org/10.1145/2620728.2620744.

[Ber+15] DanielJ. Bernstein, Bernard van Gastel, Wesley Janssen, Tanja Lange, Peter Schwabe, and Sjaak Smetsers. "TweetNaCl: A Crypto Library in 100 Tweets". English. In: *Progress in Cryptology - LATINCRYPT 2014*. Ed. by Diego F. Aranha and Alfred Menezes. Vol. 8895. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 64–83. DOI: 10.1007/978-3-319-16295-9_4. URL: http://dx.doi.org/10.1007/978-3-319-16295-9_4.

[Ber09] Daniel J. Bernstein. "Introduction to post-quantum cryptography". In: *Post-Quantum Cryptography*. Ed. by Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–14. DOI: 10.1007/978-3-540-88702-7_1. URL: https://doi.org/10.1007/978-3-540-88702-7_1.

[Bet19] BetterCloud. *State of Insider Threats in the Digital Workplace*. Tech. rep. BetterCloud.com, 2019. URL: https://www.bettercloud.com/monitor/wp-content/uploads/sites/3/2019/03/BetterCloud-State-of-Insider-Threats-2019-FINAL.pdf.

[Beu+15] Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. "A messy state of the union: Taming the composite state machines of TLS". In: *IEEE Symposium on Security and Privacy*. IEEE. 2015, pp. 535–552. DOI: 10.1109/SP.2015.39. URL: https://ieeexplore.ieee.org/abstract/document/7163046.

[BF18] O. I. Bentstuen and J. Flathagen. "On Bootstrapping In-Band Control Channels in Software Defined Networks". In: *IEEE International Conference on Communications Workshops (ICC Workshops)*. 2018, pp. 1–6. DOI: 10.1109/ICCW.2018.8403796. URL: https://ieeexplore.ieee.org/document/8403796.

[Bha+13] Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, and Pierre-Yves Strub. "Implementing TLS with verified cryptographic security". In: *IEEE Symposium on Security and Privacy (SP)*. IEEE. 2013, pp. 445–459. DOI: 10.1109/SP.2013.37. URL: https://ieeexplore.ieee.org/abstract/document/6547126.

[Bha+17] Karthikeyan Bhargavan et al. "Everest: Towards a Verified, Drop-in Replacement of HTTPS". In: *2nd Summit on Advances in Programming Languages (SNAPL 2017)*. Ed. by Benjamin S. Lerner, Rastislav Bodík, and Shriram Krishnamurthi. Vol. 71. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, 1:1–1:12. DOI: 10.4230/LIPIcs.SNAPL.2017.1. URL: http://drops.dagstuhl.de/opus/volltexte/2017/7119.

[Big20] Big Switch Networks, Inc. *Open Network Linux*. 2020. URL: https://opennetlinux.org/.

[BLN16] Daniel J. Bernstein, Tanja Lange, and Ruben Niederhagen. "Dual EC: A Standardized Back Door". In: *The New Codebreakers: Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*. Ed. by Peter Y. A. Ryan, David Naccache, and Jean-Jacques Quisquater. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 256–281. DOI: 10.1007/978-3-662-49301-4_17. URL: https://doi.org/10.1007/978-3-662-49301-4_17.

[BLS12] DanielJ. Bernstein, Tanja Lange, and Peter Schwabe. "The Security Impact of a New Cryptographic Library". English. In: *Progress in Cryptology - LATINCRYPT 2012*. Ed. by Alejandro Hevia and Gregory Neven. Vol. 7533. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 159–176. DOI: 10.1007/978-3-642-33481-8_9. URL: http://dx.doi.org/10.1007/978-3-642-33481-8_9.

[BMV17] Samaresh Bera, Sudip Misra, and Athanasios V Vasilakos. "Software-Defined Networking for Internet of Things: A Survey". In: *IEEE Internet of Things Journal* 4.6 (2017), pp. 1994–2008. DOI: 10.1109/JIOT.2017.2746186. URL: https://ieeexplore.ieee.org/abstract/document/8017556.

[BOC15] KEVIN BOCEK. *Infographic: How an Attack by a Cyber-espionage Operator Bypassed Security Controls*. 2015. URL: https://www.venafi.com/blog/post/infographic-cyber-espionage-operator-bypassed-security-controls/.

[Bot+16] Fábio Botelho, Tulio A Ribeiro, Paulo Ferreira, Fernando MV Ramos, and Alysson Bessani. "Design and Implementation of a Consistent Data Store for a Distributed SDN Control Plane". In: *12th European Dependable Computing Conference (EDCC)*. IEEE. 2016, pp. 169–180. DOI: 10.1109/EDCC.2016.12. URL: https://ieeexplore.ieee.org/abstract/document/7780356.

[BP10] Olivier Benoit and Thomas Peyrin. "Side-Channel Analysis of Six SHA-3 Candidates". English. In: *Cryptographic Hardware and Embedded Systems, CHES 2010*. Ed. by Stefan Mangard and François-Xavier Standaert. Vol. 6225. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 140–157. DOI: 10.1007/978-3-642-15031-9_10. URL: http://dx.doi.org/10.1007/978-3-642-15031-9_10.

[BSA14] A. Bessani, J. Sousa, and E. E. P. Alchieri. "State Machine Replication for the Masses with BFT-SMART". In: *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 2014, pp. 355–362. DOI: 10.1109/DSN.2014.43. URL: https://ieeexplore.ieee.org/abstract/document/6903593.

[BSM18] F. Bannour, S. Souihi, and A. Mellouk. "Distributed SDN Control: Survey, Taxonomy, and Challenges". In: *IEEE Communications Surveys Tutorials* 20.1 (Jan. 2018), pp. 333–354. DOI: 10.1109/COMST.2017.2782482. URL: https://ieeexplore.ieee.org/abstract/document/8187644.

[BT11] BillyBob Brumley and Nicola Tuveri. "Remote Timing Attacks Are Still Practical". English. In: *Computer Security - ESORICS 2011*. Ed. by Vijay Atluri and Claudia Diaz. Vol. 6879. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 355–371. DOI: 10.1007/978-3-642-23822-2_20. URL: http://dx.doi.org/10.1007/978-3-642-23822-2_20.

[Buh+15] D. Buhov, M. Huber, G. Merzdovnik, E. Weippl, and V. Dimitrova. "Network Security Challenges in Android Applications". In: *10th International Conference on Availability, Reliability and Security*. 2015, pp. 327–332. DOI: 10.1109/ARES.2015.59. URL: https://ieeexplore.ieee.org/abstract/document/7299933.

[Cal+07] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. *OpenPGP Message Format*. RFC 4880 (Proposed Standard). Updated by RFC 5581. Internet Engineering Task Force, Nov. 2007. URL: http://www.ietf.org/rfc/rfc4880.txt.

[Can+18] M. Canini, I. Salem, L. Schiff, E. M. Schiller, and S. Schmid. "Renaissance: A Self-Stabilizing Distributed SDN Control Plane". In: *IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. July 2018, pp. 233–243. DOI: 10.1109/ICDCS.2018.00032. URL: https://ieeexplore.ieee.org/abstract/document/8416295.

[Cao+19] Jiahao Cao, Kun Sun, Qi Li, Mingwei Xu, Zijie Yang, Kyung Joon Kwak, and Jason Li. “Covert Channels in SDN: Leaking Out Information from Controllers to End Hosts”. In: *Security and Privacy in Communication Networks*. Ed. by Songqing Chen, Kim-Kwang Raymond Choo, Xinwen Fu, Wenjing Lou, and Aziz Mohaisen. Cham: Springer International Publishing, 2019, pp. 429–449. DOI: https://doi.org/10.1007/978-3-030-37228-6_21. URL: https://link.springer.com/chapter/10.1007/978-3-030-37228-6_21.

[Car17] K. Carter. “Francois Raynaud on DevSecOps”. In: *IEEE Software* 34.5 (2017), pp. 93–96. DOI: 10.1109/MS.2017.3571578. URL: https://ieeexplore.ieee.org/abstract/document/8048652.

[Cas+07] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. “Ethane: Taking Control of the Enterprise”. In: *SIGCOMM Comput. Commun. Rev.* 37.4 (Aug. 2007), pp. 1–12. DOI: 10.1145/1282427.1282382. URL: https://doi.org/10.1145/1282427.1282382.

[Cas+18] Valentina Casola, Alessandra [De Benedictis], Massimiliano Rak, and Umberto Villano. “Security-by-design in multi-cloud applications: An optimization approach”. In: *Information Sciences* 454-455 (2018), pp. 344–362. DOI: https://doi.org/10.1016/j.ins.2018.04.081. URL: http://www.sciencedirect.com/science/article/pii/S0020025518303517.

[CCG16] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. “On post-compromise security”. In: *IEEE 29th Computer Security Foundations Symposium (CSF)*. IEEE. 2016, pp. 164–178. DOI: 10.1109/CSF.2016.19. URL: https://ieeexplore.ieee.org/abstract/document/7536374.

[CDM17] Mauro Conti, Fabio De Gaspari, and Luigi V. Mancini. “Know Your Enemy: Stealth Configuration-Information Gathering in SDN”. In: *Green, Pervasive, and Cloud Computing*. Ed. by M. H. A. Au, A. Castiglione, K. K. R. Choo, F. Palmieri, and K. C. Li. Cham: Springer International Publishing, 2017, pp. 386–401. DOI: https://doi.org/10.1007/978-3-319-57186-7_29. URL: https://link.springer.com/chapter/10.1007/978-3-319-57186-7_29.

[Cer01] Iliano Cervesato. “The Dolev-Yao Intruder is the Most Powerful Attacker”. In: *Proceedings of the Sixteenth Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press. Short, 2001, pp. 16–19. URL: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.21.2903.

[Cha+17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. "Post-Quantum Zero-Knowledge and Signatures from Symmetric Key Primitives". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Dallas, Texas, USA: ACM, 2017, pp. 1825–1842. DOI: 10.1145/3133956.3133997. URL: http://doi.acm.org/10.1145/3133956.3133997.

[Che+16] Min Chen, Yongfeng Qian, Shiwen Mao, Wan Tang, and Ximin Yang. "Software-Defined Mobile Networks Security". In: *Mobile Networks and Applications* 21.5 (Oct. 2016), pp. 729–743. DOI: 10.1007/s11036-015-0665-5. URL: https://doi.org/10.1007/s11036-015-0665-5.

[Che+17] Manuel Cheminod, Luca Durante, Lucia Seno, Fulvio Valenza, Adriano Valenzano, and Claudio Zunino. "Leveraging SDN to improve security in industrial networks". In: *IEEE 13th International Workshop on Factory Communication Systems (WFCS)*. IEEE. 2017, pp. 1–7. DOI: 10.1109/WFCS.2017.7991960. URL: https://ieeexplore.ieee.org/abstract/document/7991960.

[CHH16] Golriz Chehrazi, Irina Heimbach, and Oliver Hinz. "The impact of security by design on the success of open source software". In: *European Conference on Information Systems (ECIS)*. 2016. URL: https://aisel.aisnet.org/ecis2016_rp/179.

[Chi+15] Po-Wen Chi, Chien-Ting Kuo, Jing-Wei Guo, and Chin-Laung Lei. "How to detect a compromised SDN switch". In: *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. 2015, pp. 1–6. DOI: 10.1109/NETSOFT.2015.7116184. URL: https://ieeexplore.ieee.org/abstract/document/7116184.

[Chi+16] Marco Chiesa, Ilya Nikolaevskiy, Slobodan Mitrović, Aurojit Panda, Andrei Gurtov, Aleksander Maidry, Michael Schapira, and Scott Shenker. "The quest for resilient (static) forwarding tables". In: *The 35th Annual IEEE International Conference on Computer Communications (INFOCOM)*. IEEE. 2016, pp. 1–9. DOI: 10.1109/INFOCOM.2016.7524552. URL: https://ieeexplore.ieee.org/abstract/document/7524552/.

[Cho+17a] Tom Chothia, Flavio D Garcia, Chris Heppel, and Chris McMahon Stone. “Why Banker Bob (still) Can’t Get TLS Right: A Security Analysis of TLS in Leading UK Banking Apps”. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2017, pp. 579–597. DOI: https://doi.org/10.1007/978-3-319-70972-7_33. URL: https://link.springer.com/chapter/10.1007/978-3-319-70972-7_33.

[Cho+17b] Ankur Chowdhary, Sandeep Pisharody, Adel Alshamrani, and Dijiang Huang. “Dynamic Game Based Security Framework in SDN-enabled Cloud Networking Environments”. In: *Proceedings of the ACM International Workshop on Security in Software Defined Networks and Network Function Virtualization*. SDN-NFVSec ’17. Scottsdale, Arizona, USA: ACM, 2017, pp. 53–58. DOI: 10.1145/3040992.3040998. URL: http://doi.acm.org/10.1145/3040992.3040998.

[Cho+19] A. Chowdhary, D. Huang, A. Alshamrani, M. Kang, A. Kim, and A. Velazquez. “TRUFL: Distributed Trust Management Framework in SDN”. In: *IEEE International Conference on Communications (ICC)*. May 2019, pp. 1–6. DOI: 10.1109/ICC.2019.8761661. URL: https://ieeexplore.ieee.org/abstract/document/8761661/.

[Chu+07] Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. “Attested append-only memory: making adversaries stick to their word”. In: 2007, pp. 189–204. DOI: 10.1145/1294261.1294280. URL: http://doi.acm.org/10.1145/1294261.1294280.

[Cis14] Cisco. *Annual Security Report*. 2014. URL: https://www.cisco.com/c/dam/assets/global/UK/pdfs/executive_security/sc-01_casr2014_cte_liq_en.pdf.

[CKM16] V. Catros, R. Kerherve, and A. Munyandekwe. *Access Control Based on DHCP*. 2016. URL: https://wiki.onosproject.org/display/ONOS/Access+Control+Based+on+DHCP.

[CL17] Yen-Chun Chiu and Po-Ching Lin. “Rapid detection of disobedient forwarding on compromised OpenFlow switches”. In: *International Conference on Computing, Networking and Communications (ICNC)*. IEEE. 2017, pp. 672–677. DOI: 10.1109/ICCNC.2017.7876210. URL: https://ieeexplore.ieee.org/abstract/document/7876210.

[Cob20] Sarah Coble. *Cost of Insider Threats Rises 31%*. 2020. URL: https://www.infosecurity-magazine.com/news/cost-of-insider-threats-rises-31/.

[CPP15] Roy Liang Chua, Andrew Keith Pearce, and Matthew Palmer. *Authentication for software defined networks*. US Patent 9,038,151. 2015. URL: https://patents.google.com/patent/US9038151B1.

[Cro17] Bob Cromwell. *Massive Failures of Internet PKI*. http://cromwell-intl.com/cybersecurity/pki-failures.html. 2017. URL: http://cromwell-intl.com/cybersecurity/pki-failures.html.

[CS04] C. Cachin and A. Samar. "Secure distributed DNS". In: *International Conference on Dependable Systems and Networks*. 2004, pp. 423–432. DOI: 10.1109/DSN.2004.1311912. URL: https://ieeexplore.ieee.org/abstract/document/1311912.

[Cyb18] Cybersecurity Insiders. *Insider Threat*. Tech. rep. Crowd Research Partners, 2018. URL: https://crowdresearchpartners.com/wp-content/uploads/2017/07/Insider-Threat-Report-2018.pdf.

[Dac+17] Marc C Dacier, Hartmut König, Radoslaw Cwalinski, Frank Kargl, and S. Dietrich. "Security Challenges and Opportunities of Software-Defined Networking". In: *IEEE Security & Privacy* 15.2 (2017), pp. 96–100. DOI: 10.1109/MSP.2017.46. URL: https://ieeexplore.ieee.org/abstract/document/7891523.

[Dar+17] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, and M. Conti. "A Survey on the Security of Stateful SDN Data Planes". In: *IEEE Communications Surveys Tutorials* 19.3 (Sept. 2017), pp. 1701–1725. DOI: 10.1109/COMST.2017.2689819. URL: https://ieeexplore.ieee.org/abstract/document/7890396.

[De +14] Rogério Leão Santos De Oliveira, Christiane Marie Schweitzer, Ailton Akira Shinoda, and Ligia Rodrigues Prete. "Using mininet for emulation and prototyping software-defined networks". In: *IEEE Colombian Conference on Communications and Computing (COLCOM)*. IEEE. 2014, pp. 1–6. DOI: 10.1109/ColComCon.2014.6860404. URL: https://ieeexplore.ieee.org/abstract/document/6860404.

[Dha+15] Mohan Dhawan, Rishabh Poddar, Kshiteej Mahajan, and V. Mann. "SPHINX: Detecting Security Attacks in Software-Defined Networks." In: *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2015. DOI: 10.14722/ndss.2015.23064. URL: https://www.ndss-symposium.org/ndss2015/ndss-2015-programme/sphinx-detecting-security-attacks-software-defined-networks/.

[Dig17] DigiCert Inc. *Enabling Perfect Forward Secrecy.* 2017. URL: https://www.digicert.com/ssl-support/ssl-enabling-perfect-forward-secrecy.htm.

[Dix+18] Vaibhav Hemant Dixit, Adam Doupé, Yan Shoshitaishvili, Ziming Zhao, and Gail-Joon Ahn. "AIM-SDN: Attacking Information Mismanagement in SDN-Datastores". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security.* CCS '18. Toronto, Canada: Association for Computing Machinery, 2018, pp. 664–676. DOI: 10.1145/3243734.3243799. URL: https://doi.org/10.1145/3243734.3243799.

[DK16] Felix Dörre and Vladimir Klebanov. "Practical Detection of Entropy Loss in Pseudo-Random Number Generators". In: *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security.* CCS '16. Vienna, Austria: Association for Computing Machinery, 2016, pp. 678–689. DOI: 10.1145/2976749.2978369. URL: https://doi.org/10.1145/2976749.2978369.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. "Consensus in the Presence of Partial Synchrony". In: *J. ACM* 35.2 (Apr. 1988), pp. 288–323. DOI: 10.1145/42282.42283. URL: https://doi.org/10.1145/42282.42283.

[Dod+13] Yevgeniy Dodis, David Pointcheval, Sylvain Ruhault, Damien Vergniaud, and Daniel Wichs. "Security Analysis of Pseudo-random Number Generators with Input: /Dev/Random is Not Robust". In: *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security.* CCS '13. Berlin, Germany: ACM, 2013, pp. 647–658. DOI: 10.1145/2508859.2516653. URL: http://doi.acm.org/10.1145/2508859.2516653.

[Dov13] Jeremy M Dover. *A denial of service attack against the Open Floodlight SDN controller.* 2013. URL: https://docplayer.net/7087999-A-denial-of-service-attack-against-the-open-floodlight-sdn-controller.html.

[Dov17] Jeremy M. Dover. *A switch table vulnerability in the Open Floodlight SDN controller.* 2017. URL: http://docplayer.net/4847712-A-switch-table-vulnerability-in-the-open-floodlight-sdn-controller.html.

[Dri20] DriveNets. *White Box Architecture.* 2020. URL: https://drivenets.com/products/white-box-architecture/.

[DSZ16] Benjamin Dowling, Douglas Stebila, and Greg Zaverucha. “Authenticated Network Time Synchronization”. In: *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, 2016, pp. 823–840. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/dowling.

[DY83] Danny Dolev and Andrew Yao. “On the security of public key protocols”. In: *IEEE Transactions on Information Theory* 29.2 (1983), pp. 198–208. DOI: 10.1109/TIT.1983.1056650. URL: https://ieeexplore.ieee.org/abstract/document/1056650.

[EC16] C. Ebert and J. Cain. “Cyclomatic Complexity”. In: *IEEE Software* 33.6 (2016), pp. 27–29. DOI: 10.1109/MS.2016.147. URL: https://ieeexplore.ieee.org/abstract/document/7725232.

[Edw14] Chris Edwards. “Researchers probe security through obscurity”. In: *Communications of the ACM* 57.8 (2014), pp. 11–13. DOI: 10.1145/2632038. URL: https://dl.acm.org/doi/fullHtml/10.1145/2632038.

[Ege+13] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. “An Empirical Study of Cryptographic Misuse in Android Applications”. In: *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. CCS ’13. Berlin, Germany: ACM, 2013, pp. 73–84. DOI: 10.1145/2508859.2516693. URL: http://doi.acm.org/10.1145/2508859.2516693.

[Ekr19] Ekran System. *Insider Threat Statistics for 2019: Facts and Figures*. 2019. URL: https://www.ekransystem.com/en/blog/insider-threat-statistics-facts-and-figures.

[Eng12] Theresa Enghardt. “Authentication, Authorization and Mobility in Openflow-enabled Enterprise Wireless Networks”. MA thesis. Berlin: Technische Universität Berlin, 2012. URL: https://www.net.t-labs.tu-berlin.de/papers/E-AAMOE-11.pdf.

[Fah+12] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. “Why eve and mallory love android: an analysis of android SSL (in)security”. In: *Proceedings of the 2012 ACM conference on Computer and communications security*. CCS ’12. Raleigh, North Carolina, USA: ACM, 2012, pp. 50–61. DOI: 10.1145/2382196.2382205. URL: http://doi.acm.org/10.1145/2382196.2382205.

[Fan+19] L. Fang, Y. Li, X. Yun, Z. Wen, S. Ji, W. Meng, Z. Cao, and M.Tanveer. “THP: A Novel Authentication Scheme to Prevent Multiple Attacks in SDN-based IoT Network”. In: *IEEE Internet of Things Journal* (2019), pp. 1–1. DOI: 10.1109/JIOT.2019.2944301. URL: https://ieeexplore.ieee.org/abstract/document/8851192.

[Far+19] I. Farris, T. Taleb, Y. Khettab, and J. Song. “A Survey on Emerging SDN and NFV Security Mechanisms for IoT Systems”. In: *IEEE Communications Surveys Tutorials* 21.1 (Jan. 2019), pp. 812–837. DOI: 10.1109/COMST.2018.2862350. URL: https://ieeexplore.ieee.org/abstract/document/8424018.

[Fer+13] Andrew D. Ferguson, Arjun Guha, Chen Liang, Rodrigo Fonseca, and Shriram Krishnamurthi. “Participatory networking: an API for application control of SDNs”. In: *Proceedings of the ACM SIGCOMM conference on SIGCOMM*. SIGCOMM ’13. Hong Kong, China: ACM, 2013, pp. 327–338. DOI: 10.1145/2486001.2486003. URL: http://doi.acm.org/10.1145/2486001.2486003.

[FM19] J. Fiaidhi and S. Mohammed. “Security and Vulnerability of Extreme Automation Systems: The IoMT and IoA Case Studies”. In: *IT Professional* 21.4 (July 2019), pp. 48–55. DOI: 10.1109/MITP.2019.2906442. URL: https://ieeexplore.ieee.org/abstract/document/8764076.

[Fon+15] Ramon R Fontes, Samira Afzal, Samuel HB Brito, Mateus AS Santos, and Christian Esteve Rothenberg. “Mininet-WiFi: Emulating software-defined wireless networks”. In: *11th International Conference on Network and Service Management (CNSM)*. IEEE. 2015, pp. 384–389. DOI: 10.1109/CNSM.2015.7367387. URL: https://ieeexplore.ieee.org/abstract/document/7367387.

[Fre+19] Lars-undefinedke Fredlund, Clara Benac Earle, Thomas Arts, and Hans Svensson. “Gaining Trust by Tracing Security Protocols”. In: *Proceedings of the 18th ACM SIGPLAN International Workshop on Erlang*. Erlang 2019. Berlin, Germany: Association for Computing Machinery, 2019, pp. 56–67. DOI: 10.1145/3331542.3342573. URL: https://doi.org/10.1145/3331542.3342573.

[FSK11] Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno. *Cryptography engineering: design principles and practical applications*. John Wiley & Sons, 2011. URL: https://www.wiley.com/en-us/Cryptography+Engineering%3A+Design+Principles+and+Practical+Applications+-p-9780470474242.

[FWC16] Shuqin Fan, Wenbo Wang, and Qingfeng Cheng. “Attacking OpenSSL Implementation of ECDSA with a Few Signatures”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’16. Vienna, Austria: Association for Computing Machinery, 2016, pp. 1505–1515. DOI: 10.1145/2976749.2978400. URL: https://doi.org/10.1145/2976749.2978400.

[Gar+19] S. Garg, K. Kaur, N. Kumar, and J. J. P. C. Rodrigues. “Hybrid Deep-Learning-Based Anomaly Detection Scheme for Suspicious Flow Detection in SDN: A Social Multimedia Perspective”. In: *IEEE Transactions on Multimedia* 21.3 (Mar. 2019), pp. 566–578. DOI: 10.1109/TMM.2019.2893549. URL: https://ieeexplore.ieee.org/abstract/document/8613868.

[Gio+20] A. Giorgetti, A. Sgambelluri, R. Casellas, R. Morro, A. Campanella, and P. Castoldi. “Control of open and disaggregated transport networks using the Open Network Operating System (ONOS) [Invited]”. In: *IEEE/OSA Journal of Optical Communications and Networking* 12.2 (2020), A171–A181. DOI: 10.1364/JOCN.12.00A171. URL: https://ieeexplore.ieee.org/abstract/document/8925429.

[GK91] Geoffrey K. Gill and Chris F. Kemerer. “Cyclomatic complexity density and software maintenance productivity”. In: *IEEE transactions on software engineering* 17.12 (1991), p. 1284. DOI: 10.1109/32.106988. URL: https://search.proquest.com/docview/195570778.

[Gog17] Marcell Gogan. *Insider Threats as the Main Security Threat in 2017*. 2017. URL: https://www.tripwire.com/state-of-security/security-data-protection/insider-threats-main-security-threat-2017/.

[Gov+16] Ramesh Govindan, Ina Minei, Mahesh Kallahalla, Bikash Koley, and Amin Vahdat. “Evolve or Die: High-Availability Design Principles Drawn from Googles Network Infrastructure”. In: *Proceedings of the ACM SIGCOMM Conference*. SIGCOMM ’16. Florianopolis, Brazil: Association for Computing Machinery, 2016, pp. 58–72. DOI: 10.1145/2934872.2934891. URL: https://doi.org/10.1145/2934872.2934891.

[GPM15] Christina Garman, Kenneth G. Paterson, and Thyla Van der Merwe. “Attacks Only Get Better: Password Recovery Attacks Against RC4 in TLS”. In: *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, 2015, pp. 113–128. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/garman.

[Gre+08] Albert Greenberg, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. “Towards a Next Generation Data Center Architecture: Scalability and Commoditization”. In: *Proceedings of the ACM Workshop on Programmable Routers for Extensible Services of Tomorrow*. PRESTO ’08. Seattle, WA, USA: ACM, 2008, pp. 57–62. DOI: 10.1145/1397718.1397732. URL: http://doi.acm.org/10.1145/1397718.1397732.

[Gre+09] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. “VL2: A Scalable and Flexible Data Center Network”. In: *SIGCOMM Comput. Commun. Rev.* 39.4 (Aug. 2009), pp. 51–62. DOI: 10.1145/1594977.1592576. URL: http://doi.acm.org/10.1145/1594977.1592576.

[Gre09] K. Greene. *MIT Tech Review 10 Breakthrough Technologies: Software-defined Networking*. http://www2.technologyreview.com/article/412194/tr10-software-defined-networking/. 2009.

[Gre12] Matthew Green. *The anatomy of a bad idea*. 2012. URL: http://blog.cryptographyengineering.com/2012/12/the-anatomy-of-bad-idea.html.

[Gre17] Matthew Green. *The Strange story of “Extended Random”*. 2017. URL: https://blog.cryptographyengineering.com.

[GT16] Shafi Goldwasser and Yael Tauman Kalai. “Cryptographic Assumptions: A Position Paper”. In: *Theory of Cryptography*. Ed. by Eyal Kushilevitz and Tal Malkin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 505–522. URL: https://link.springer.com/chapter/10.1007/978-3-662-49096-9_21.

[Guo+16] Ivy Guo, Makan Pourzandi, Sandra Scott-Hayward, Haibin Song, Clair Wangke, Frank Xialiang, Dacheng Zhang, and Xiaojun Zhuang. *Security Foundation Requirements for SDN Controllers*. Tech. rep. Open Networking Foundation, 2016. URL: https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2013/05/Security_Foundation_Requirements_for_SDN_Controllers.pdf.

[Han+19] T. Han, S. Rooh Ullah Jan, Zhiyuan Tan, Muhammad Usman, Mian Ahmad Jan, Rahim Khan, and Yongzhao Xu. “A comprehensive survey of security threats and their mitigation techniques for next-generation SDN controllers”. In: *Concurrency and Computation: Practice and Experience* (2019). e5300 cpe.5300. DOI: 10.1002/cpe.5300.

URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5300.

[Hel+10] Brandon Heller, Srini Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. “ElasticTree: saving energy in data center networks”. In: *Proceedings of the 7th USENIX conference on Networked systems design and implementation*. NSDI’10. San Jose, California: USENIX Association, 2010, pp. 17–17. URL: http://dl.acm.org/citation.cfm?id=1855711.1855728.

[Hel19] Help Net Security. *99% of misconfiguration incidents in the cloud go unnoticed*. 2019. URL: https://www.helpnetsecurity.com/2019/09/25/cloud-misconfiguration-incidents/.

[Hen+12] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. “Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices”. In: *Proceedings of the 21st USENIX Conference on Security Symposium*. Security’12. Bellevue, WA: USENIX Association, 2012, pp. 35–35. URL: http://dl.acm.org/citation.cfm?id=2362793.2362828.

[Hep+19] T. Hepp, F. Spaeh, A. Schoenhals, P. Ehret, and B. Gipp. “Exploring Potentials and Challenges of Blockchain-based Public Key Infrastructures”. In: *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. Apr. 2019, pp. 847–852. DOI: 10.1109/INFCOMW.2019.8845169. URL: https://ieeexplore.ieee.org/abstract/document/8845169.

[Her+00] A. Herzberg, Y. Mass, J. Mihaeli, D. Naor, and Y. Ravid. “Access control meets public key infrastructure, or: assigning roles to strangers”. In: *Proceeding IEEE Symposium on Security and Privacy*. 2000, pp. 2–14. DOI: 0.1109/SECPRI.2000.848442. URL: https://ieeexplore.ieee.org/abstract/document/848442.

[HFH16] Marcella Hastings, Joshua Fried, and Nadia Heninger. “Weak Keys Remain Widespread in Network Devices”. In: *Proceedings of the Internet Measurement Conference*. IMC ’16. Santa Monica, California, USA: Association for Computing Machinery, 2016, pp. 49–63. DOI: 10.1145/2987443.2987486. URL: https://doi.org/10.1145/2987443.2987486.

[Hic17] Andrew Hickey. *Application layer DDoS attacks rising*. Sept. 2017. URL: https://www.csoonline.com/article/3222824/network-security/application-layer-ddos-attacks-rising.html.

[Hil13] Brad Hill. *Failures of Trust in the Online PKI Marketplace Cannot be Fixed by "Raising the Bar" on Certificate Authority Security.* 2013. URL: http://csrc.nist.gov/groups/ST/ca-workshop-2013/cfp-submissions/hill_failures_to_trust.pdf.

[HMK12] Gernot Heiser, Toby Murray, and Gerwin Klein. "It's Time for Trustworthy Systems". In: *IEEE Security & Privacy* 10.2 (2012), pp. 67–70. DOI: 10.1109/MSP.2012.41. URL: https://ieeexplore.ieee.org/abstract/document/6173000.

[Hom+19] Ivan Homoliak, Flavio Toffalini, Juan Guarnizo, Yuval Elovici, and Martıén Ochoa. "Insight Into Insiders and IT: A Survey of Insider Threat Taxonomies, Analysis, Modeling, and Countermeasures". In: *ACM Comput. Surv.* 52.2 (Apr. 2019). DOI: 10.1145/3303771. URL: https://doi.org/10.1145/3303771.

[Hon+15] Sungmin Hong, Lei Xu, Haopei Wang, and Guofei Gu. "Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures." In: *Proceedings of the Network and Distributed System Security Symposium (NDSS).* Internet Society, 2015. URL: https://www.ndss-symposium.org/ndss2015/ndss-2015-programme/poisoning-network-visibility-software-defined-networks-new-attacks-and-countermeasures/.

[Hou+02] R. Housley, W. Polk, W. Ford, and D. Solo. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.* RFC 3280 (Proposed Standard). Obsoleted by RFC 5280, updated by RFCs 4325, 4630. Internet Engineering Task Force, Apr. 2002. URL: http://www.ietf.org/rfc/rfc3280.txt.

[HP08] Helena Handschuh and Bart Preneel. "Key-Recovery Attacks on Universal Hash Function Based MAC Algorithms". English. In: *Advances in Cryptology - CRYPTO 2008.* Ed. by David Wagner. Vol. 5157. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 144–161. DOI: 10.1007/978-3-540-85174-5_9. URL: http://dx.doi.org/10.1007/978-3-540-85174-5_9.

[Hua+14] L. S. Huang, S. Adhikarla, D. Boneh, and C. Jackson. "An Experimental Study of TLS Forward Secrecy Deployments". In: *IEEE Internet Computing* 18.6 (2014), pp. 43–51. DOI: 10.1109/MIC.2014.86. URL: https://ieeexplore.ieee.org/abstract/document/6870379.

[Hua+16] Liang-Hao Huang, Hsiang-Chun Hsu, Shan-Hsiang Shen, De-Nian Yang, and Wen-Tsuen Chen. "Multicast traffic engineering for software-defined networks". In: *The 35th Annual IEEE International Conference on Computer Communications*. IEEE. 2016, pp. 1–9. DOI: 10.1109/INFOCOM.2016.7524383. URL: https://ieeexplore.ieee.org/abstract/document/7524383.

[Hua+17] X. Huang, S. Bian, Z. Shao, and H. Xu. "Dynamic switch-controller association and control devolution for SDN systems". In: *IEEE International Conference on Communications (ICC)*. 2017, pp. 1–6. DOI: 10.1109/ICC.2017.7997427. URL: https://ieeexplore.ieee.org/abstract/document/7997427.

[IBM18] IBM Security. *2018 IBM X-Force Threat Intelligence Index*. 2018. URL: https://www.ibm.com/security/data-breach/threat-intelligence?ce=ISM0484&ct=SWG&cmp=IBMSocial&cm=h&cr=Security&ccy=US.

[IEE15] IEEE Spectrum. *Special Report: 50 years of Moore's Law*. 2015. URL: http://spectrum.ieee.org/static/special-report-50-years-of-moores-law.

[IEE18] IEEE 802.1. *802.1AR: Secure Device Identity*. 2018. URL: https://1.ieee802.org/security/802-1ar/.

[IK18] Qamar Ilyas and Rahamatullah Khondoker. "Security Analysis of FloodLight, ZeroSDN, Beacon and POX SDN Controllers". In: *SDN and NFV Security: Security Analysis of Software-Defined Networking and Network Function Virtualization*. Ed. by Rahamatullah Khondoker. Cham: Springer International Publishing, 2018, pp. 85–98. DOI: 10.1007/978-3-319-71761-6_6. URL: https://doi.org/10.1007/978-3-319-71761-6_6.

[Inf13] InfoSec. *Layer 7 DDoS attacks: detection & mitigation*. 2013. URL: http://resources.infosecinstitute.com/layer-7-ddos-attacks-detection-mitigation/.

[Isl+19] S. Islam, M. A. Islam Khan, S. Tasnim Shorno, S. Sarker, and M. A. Siddik. "Performance Evaluation of SDN Controllers in Wireless Network". In: *1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)*. 2019, pp. 1–5. DOI: 10.1109/ICASERT.2019.8934553. URL: https://ieeexplore.ieee.org/abstract/document/8934553.

[Jai+13] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, and et al. "B4: Experience with a Globally-Deployed Software Defined Wan". In: *SIGCOMM Comput. Commun. Rev.* 43.4 (Aug. 2013), pp. 3–14. DOI: 10.1145/2534169.2486019. URL: https://doi.org/10.1145/2534169.2486019.

[JCL19] Wafa Ben Jaballah, Mauro Conti, and Chhagan Lal. "A Survey on Software-Defined VANETs: Benefits, Challenges, and Future Directions". In: *CoRR* abs/1904.04577 (2019). arXiv: 1904.04577. URL: http://arxiv.org/abs/1904.04577.

[Jer+17a] S. Jero, W. Koch, R. Skowyra, H. Okhravi, C. Nita-Rotaru, and D. Bigelow. "Identifier Binding Attacks and Defenses in Software-Defined Networks". In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017, pp. 415–432. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/jero.

[Jer+17b] Samuel Jero, Xiangyu Bu, Cristina Nita-Rotaru, Hamed Okhravi, Richard Skowyra, and Sonia Fahmy. "BEADS: Automated Attack Discovery in OpenFlow-Based SDN Systems". In: *Research in Attacks, Intrusions, and Defenses*. Ed. by Marc Dacier, Michael Bailey, Michalis Polychronakis, and Manos Antonakakis. Cham: Springer International Publishing, 2017, pp. 311–333. DOI: https://doi.org/10.1007/978-3-319-66332-6_14. URL: https://link.springer.com/chapter/10.1007/978-3-319-66332-6_14.

[Jin+16] Xin Jin, Yiran Li, Da Wei, Siming Li, Jie Gao, Lei Xu, Guangzhi Li, Wei Xu, and Jennifer Rexford. "Optimizing Bulk Transfers with Software-Defined Optical WAN". In: *Proceedings of the ACM SIGCOMM Conference*. SIGCOMM '16. Florianopolis, Brazil: Association for Computing Machinery, 2016, pp. 87–100. DOI: 10.1145/2934872.2934904. URL: https://doi.org/10.1145/2934872.2934904.

[JKK19] Prathima Mabel J., Vani K.A., and Rama Mohan Babu K.N. "SDN Security: Challenges and Solutions". In: *Emerging Research in Electronics, Computer Science and Technology*. Ed. by Sridhar V., Padma M., and Rao K. Vol. 545. Lecture Notes in Electrical Engineering, Springer, Singapore, 2019. DOI: 10.1007/978-981-13-5802-9_73. URL: https://doi.org/10.1007/978-981-13-5802-9_73.

[JSV17] Anthony Journault, François-Xavier Standaert, and Kerem Varici. "Improving the security and efficiency of block ciphers based on LS-designs". In: *Designs, Codes and Cryptography* 82.1 (Jan. 2017), pp. 495–509. DOI: 10.1007/s10623-016-0193-8. URL: https://doi.org/10.1007/s10623-016-0193-8.

[Jun08] Inc. Juniper Networks. *What's Behind Network Downtime?* 2008. URL: https://www-935.ibm.com/services/au/gts/pdf/200249.pdf.

[JYR11] Lavanya Jose, Minlan Yu, and Jennifer Rexford. "Online measurement of large traffic aggregates on commodity switches". In: *Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services*. Hot-ICE'11. Boston, MA: USENIX Association, 2011, pp. 13–13. URL: https://static.usenix.org/events/hotice11/tech/full_papers/Jose.pdf.

[KAI14] Z. K. Khattak, M. Awais, and A. Iqbal. "Performance evaluation of OpenDaylight SDN controller". In: *20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*. 2014, pp. 671–676. DOI: 10.1109/PADSW.2014.7097868. URL: https://ieeexplore.ieee.org/abstract/document/7097868.

[Kam+16] A. V. Kamath, S. S, K. Kataoka, N. Vijayvergiya, G. B. Reddy, and S. Phatale. "SAFE: Software-Defined Authentication Framework". In: *Proceedings of the 12th Asian Internet Engineering Conference*. AINTEC '16. Bangkok, Thailand: Association for Computing Machinery, 2016, pp. 57–63. DOI: 10.1145/3012695.3012703. URL: https://doi.org/10.1145/3012695.3012703.

[Kap+12] Rudiger Kapitza, Johannes Behl, Christian Cachin, Tobias Distler, Simon Kuhnle, Seyed Vahid Mohammadi, W. Schroder-Preikschat, and Klaus Stengel. "CheapBFT: resource-efficient byzantine fault tolerance". In: 2012, pp. 295–308. DOI: 10.1145/2168836.2168866. URL: http://doi.acm.org/10.1145/2168836.2168866.

[KDH18] R. Khalili, Z. Despotovic, and A. Hecker. "Flow Setup Latency in SDN Networks". In: *IEEE Journal on Selected Areas in Communications* 36.12 (2018), pp. 2631–2639. DOI: 10.1109/JSAC.2018.2871291. URL: https://ieeexplore.ieee.org/abstract/document/8468231.

[Kee+05] Michelle Keeney, Eileen Kowalski, Dawn Cappelli, Andrew Moore, Timothy Shimeall, and Stephanie Rogers. *Insider threat study: Computer system sabotage in critical infrastructure sectors*. Tech. rep. National Threat Assessment CTR Washington DC, 2005. URL: https://legacy.secretservice.gov/ntac/its_report_050516_es.pdf.

[KF15] Andrzej Kamisiński and Carol Fung. “FlowMon: Detecting Malicious Switches in Software-Defined Networks”. In: *Proceedings of the 2015 Workshop on Automated Decision Making for Active Cyber Defense*. SafeConfig ’15. Denver, Colorado, USA: ACM, 2015, pp. 39–45. DOI: 10.1145/2809826.2809833. URL: http://doi.acm.org/10.1145/2809826.2809833.

[Kha+17] S. Khan, A. Gani, A. W. Abdul Wahab, M. Guizani, and M. K. Khan. “Topology Discovery in Software Defined Networks: Threats, Taxonomy, and State-of-the-Art”. In: *IEEE Communications Surveys Tutorials* 19.1 (2017), pp. 303–324. DOI: 10.1109/COMST.2016.2597193. URL: https://ieeexplore.ieee.org/abstract/document/7534866.

[KHL13] Soo Hyeon Kim, Daewan Han, and Dong Hoon Lee. “Predictability of Android OpenSSL’s Pseudo Random Number Generator”. In: *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. CCS ’13. Berlin, Germany: ACM, 2013, pp. 659–668. DOI: 10.1145/2508859.2516706. URL: http://doi.acm.org/10.1145/2508859.2516706.

[Kim+06] Jongsung Kim, Alex Biryukov, Bart Preneel, and Seokhie Hong. “On the Security of HMAC and NMAC Based on HAVAL, MD4, MD5, SHA-0 and SHA-1 (Extended Abstract)”. English. In: *Security and Cryptography for Networks*. Ed. by Roberto De Prisco and Moti Yung. Vol. 4116. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 242–256. DOI: 10.1007/11832072_17. URL: http://dx.doi.org/10.1007/11832072_17.

[Kim+19] Yeonkeun Kim, Jaehyun Nam, Taejune Park, Sandra Scott-Hayward, and Seungwon Shin. “SODA: A software-defined security framework for IoT environments”. In: *Computer Networks* 163 (2019), p. 106889. DOI: https://doi.org/10.1016/j.comnet.2019.106889. URL: http://www.sciencedirect.com/science/article/pii/S1389128619307522.

[KKS13] R. Kloti, V. Kotronis, and P. Smith. “Openflow: A security analysis”. In: *21st IEEE International Conference on Network Protocols*. IEEE.

2013, pp. 1–6. DOI: 10.1109/ICNP.2013.6733671. URL: https://ieeexplore.ieee.org/abstract/document/6733671.

[Kop+10] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. "Onix: A Distributed Control Platform for Large-Scale Production Networks". In: *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*. OSDI'10. Vancouver, BC, Canada: USENIX Association, 2010, pp. 351–364. URL: https://dl.acm.org/doi/10.5555/1924943.1924968.

[Kou+19] Michail-Alexandros Kourtis, G. Xilouris, D. Makris, A. Sarlas, T. Soenen, H. Koumaras, and A. Kourtis. "An End-to-End Carrier Ethernet MEF enabled 5G network architecture". In: *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. Apr. 2019, pp. 20–24. URL: https://ieeexplore.ieee.org/abstract/document/8717808.

[KPY15] J. W. Kang, S. H. Park, and J. You. "Mynah: Enabling lightweight data plane authentication for SDN controllers". In: *24th International Conference on Computer Communication and Networks (ICCCN)*. IEEE. 2015, pp. 1–6. DOI: 10.1109/ICCCN.2015.7288433. URL: https://ieeexplore.ieee.org/abstract/document/7288433.

[Kre+14] D. Kreutz, A. Bessani, E. Feitosa, and H. Cunha. "Towards Secure and Dependable Authentication and Authorization Infrastructures". In: *IEEE 20th Pacific Rim International Symposium on Dependable Computing*. 2014, pp. 43–52. DOI: 10.1109/PRDC.2014.14. URL: https://ieeexplore.ieee.org/abstract/document/6974750.

[Kre+15] D. Kreutz, F.M.V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig. "Software-Defined Networking: A Comprehensive Survey". In: *Proceedings of the IEEE* 103.1 (Jan. 2015), pp. 14–76. DOI: 10.1109/JPROC.2014.2371999. URL: https://ieeexplore.ieee.org/abstract/document/6994333.

[Kre+16] Diego Kreutz, Oleksandr Malichevskyy, Eduardo Feitosa, H. Cunha, Rodrigo da Rosa Righi, and Douglas D.J. de Macedo. "A cyber-resilient architecture for critical security services". In: *Journal of Network and Computer Applications* 63 (2016), pp. 173–189. DOI: http://dx.doi.org/10.1016/j.jnca.2015.09.014. URL: http://www.sciencedirect.com/science/article/pii/S1084804516000539.

[Kre+17] Diego Kreutz, Paulo Jorge Esteves Verissimo, Catia Magalhaes, and Fernando M. V. Ramos. "The KISS principle in Software-Defined Networking: An architecture for Keeping It Simple and Secure". In: *CoRR* abs/1702.04294 (2017). arXiv: 1702.04294. URL: http://arxiv.org/abs/1702.04294.

[Kre+18] D. Kreutz, J. Yu, P. Esteves-Verissimo, C. Magalhaes, and Fernando M. V. Ramos. "The KISS Principle in Software-Defined Networking: A Framework for Secure Communications". In: *IEEE Security Privacy* 16.5 (Sept. 2018), pp. 60–70. DOI: 10.1109/MSP.2018.3761717. URL: https://ieeexplore.ieee.org/abstract/document/8490167.

[Kre+19] Diego Kreutz, Jiangshan Yu, F. M. V. Ramos, and Paulo Esteves-Verissimo. "ANCHOR: Logically Centralized Security for Software-Defined Networks". In: *ACM Trans. Priv. Secur.* 22.2 (Feb. 2019), 8:1–8:36. DOI: 10.1145/3301305. URL: http://doi.acm.org/10.1145/3301305.

[KRV13] Diego Kreutz, Fernando M.V. Ramos, and Paulo Verissimo. "Towards Secure and Dependable Software-Defined Networks". In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. HotSDN '13. Hong Kong, China: Association for Computing Machinery, 2013, pp. 55–60. DOI: 10.1145/2491185.2491199. URL: https://doi.org/10.1145/2491185.2491199.

[KSG14] K. Kaur, J. Singh, and N. S. Ghumman. "Mininet as software defined networking testing platform". In: *International Conference on Communication, Computing & Systems (ICCCS)*. 2014, pp. 139–42. URL: http://www.sbsstc.ac.in/icccs2014/Papers/Paper29.pdf.

[KSM13] Timo Kiravuo, Mikko Sarela, and Jukka Manner. "A survey of ethernet LAN security". In: *IEEE Communications Surveys & Tutorials* 15.3 (2013), pp. 1477–1491. DOI: 10.1109/SURV.2012.121112.00190. URL: https://ieeexplore.ieee.org/abstract/document/6407456.

[Ku+14] Ian Ku, You Lu, Mario Gerla, Francesco Ongaro, Rafael L Gomes, and Eduardo Cerqueira. "Towards software-defined VANET: Architecture and services". In: *13th Annual Mediterranean Ad Hoc Networking Workshop (MED-HOC-NET)*. IEEE. 2014, pp. 103–110. DOI: 10.1109/MedHocNet.2014.6849111. URL: https://ieeexplore.ieee.org/abstract/document/6849111.

[Kuo+18] Chien-Ting Kuo, Po-Wen Chi, Victor Chang, and Chin-Laung Lei. "SFaaS: Keeping an eye on IoT fusion environment with security fusion as a service". In: *Future Generation Computer Systems* (2018). DOI: https://doi.org/10.1016/j.future.2017.12.069. URL: https://www.sciencedirect.com/science/article/pii/S0167739X17324834.

[Lam+16] Jun Huy Lam, Sang-Gon Lee, Hoon-Jae Lee, and Yustus Eko Oktian. "TLS Channel Implementation for ONOS's East/West-Bound Communication". In: *Electronics, Communications and Networks V*. Ed. by Amir Hussain. Singapore: Springer Singapore, 2016, pp. 397–403. DOI: https://doi.org/10.1007/978-981-10-0740-8_45. URL: https://link.springer.com/chapter/10.1007/978-981-10-0740-8_45.

[Lam+18] JunHuy Lam, Sang-Gon Lee, Hoon-Jae Lee, and Yustus Eko Oktian. "Design, implementation, and performance evaluation of identity-based cryptography in ONOS". In: *International Journal of Network Management* 28.1 (2018). e1990 nem.1990, e1990. DOI: 10.1002/nem.1990. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nem.1990. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nem.1990.

[Lan19] Bob Lantz. *Mininet: Rapid Prototyping for Software Defined Networks*. 2019. URL: https://github.com/mininet/mininet.

[LBL16] Nan (Peter) Liang, David P. Biros, and Andy Luse. "An Empirical Validation of Malicious Insider Characteristics". In: *Journal of Management Information Systems* 33.2 (2016), pp. 361–392. DOI: 10.1080/07421222.2016.1205925. eprint: https://doi.org/10.1080/07421222.2016.1205925. URL: https://doi.org/10.1080/07421222.2016.1205925.

[Lee+17] Seungsoo Lee, Changhoon Yoon, Chanhee Lee, Seungwon Shin, Vinod Yegneswaran, and Phillip Porras. "DELTA: A security assessment framework for software-defined networks". In: *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Vol. 17. Internet Society, 2017. URL: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/delta-security-assessment-framework-software-defined-networks/.

[Lee+20] S. Lee, J. Kim, S. Woo, C. Yoon, S. Scott-Hayward, V. Yegneswaran, P. Porras, and S. Shin. "A comprehensive security assessment framework for software-defined networks". In: *Computers & Security* 91 (2020), p. 101720. DOI: 10.1016/j.cose.2020.101720. URL:

http://www.sciencedirect.com/science/article/pii/S0167404820300079.

[Lev+09] Dave Levin, John R. Douceur, Jacob R. Lorch, and Thomas Moscibroda. "TrInc: Small Trusted Hardware for Large Distributed Systems". In: 2009, pp. 1–14. URL: http://www.usenix.org/events/nsdi09/tech/full_papers/levin/levin.pdf.

[LHM10] Bob Lantz, Brandon Heller, and Nick McKeown. "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks". In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. Hotnets-IX. Monterey, California: Association for Computing Machinery, 2010. DOI: 10.1145/1868447.1868466. URL: https://doi.org/10.1145/1868447.1868466.

[Liu+18] L. Liu, O. De Vel, Q. Han, J. Zhang, and Y. Xiang. "Detecting and Preventing Cyber Insider Threats: A Survey". In: *IEEE Communications Surveys Tutorials* 20.2 (Apr. 2018), pp. 1397–1417. DOI: 10.1109/COMST.2018.2800740. URL: https://ieeexplore.ieee.org/abstract/document/8278157.

[Liv+15] Benjamin Livshits, Manu Sridharan, Yannis Smaragdakis, Ondřej Lhoták, J. Nelson Amaral, Bor-Yuh Evan Chang, Samuel Z. Guyer, Uday P. Khedker, Anders Møller, and Dimitrios Vardoulakis. "In Defense of Soundiness: A Manifesto". In: *Commun. ACM* 58.2 (Jan. 2015), pp. 44–46. DOI: 10.1145/2644805. URL: https://doi.org/10.1145/2644805.

[LMK16] W. Li, W. Meng, and L. F. Kwok. "A survey on OpenFlow-based Software Defined Networks: Security challenges and countermeasures". In: *Journal of Network and Computer Applications* 68 (2016), pp. 126–139. DOI: https://doi.org/10.1016/j.jnca.2016.04.011. URL: http://www.sciencedirect.com/science/article/pii/S1084804516300613.

[LP17] Hewlett Packard Enterprise Development LP. *Aruba VAN SDN Controller 2.8 Administration Guide*. 2017. URL: http://h20628.www2.hp.com/km-ext/kmcsdirect/emr_na-a00003662en_us-1.pdf.

[LS17] Chumg-Wei Lin and Alberto Sangiovanni-Vincentelli. *Security-Aware Design for Cyber-Physical Systems*. Springer, 2017. DOI: 10.1007/978-3-319-51328-7. URL: https://www.springer.com/gp/book/9783319513270.

[Lua+19] S. Luangoudom, T. Nguyen, D. Tran, and L. G. Nguyen. “End to end message encryption using Poly1305 and XSalsa20 in Low power and Lossy Networks”. In: *11th International Conference on Knowledge and Systems Engineering (KSE)*. Oct. 2019, pp. 1–5. DOI: 10.1109/KSE.2019.8919479. URL: https://ieeexplore.ieee.org/abstract/document/8919479.

[LW18] Jaeho Lee and Dan S Wallach. “Removing Secrets from Android’s TLS”. In: *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. 2018. URL: https://www.cs.rice.edu/~jl128/papers/ndss18-removing_secrets_from_androids_tls_jaeho_dan.pdf.

[Maa+18] R. Maaloul, R. Taktak, L. Chaari, and B. Cousin. “Energy-Aware Routing in Carrier-Grade Ethernet Using SDN Approach”. In: *IEEE Transactions on Green Communications and Networking* 2.3 (Sept. 2018), pp. 844–858. DOI: 10.1109/TGCN.2018.2832658. URL: https://ieeexplore.ieee.org/abstract/document/8353829.

[Mah+19] T. Mahboob, I. Arshad, A. Batool, and M. Nawaz. “Authentication Mechanism to Secure Communication between Wireless SDN Planes”. In: *16th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*. Jan. 2019, pp. 582–588. DOI: 10.1109/IBCAST.2019.8667157. URL: https://ieeexplore.ieee.org/abstract/document/8667157.

[Mal+16] Aanchal Malhotra, Isaac E Cohen, Erik Brakke, and Sharon Goldberg. “Attacking the Network Time Protocol.” In: *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. 2016. URL: https://www.ndss-symposium.org/wp-content/uploads/2017/09/attacking-network-time-protocol.pdf.

[Mat+19] I. Mathebula, B. Isong, N. Gasela, and A. M. Abu-Mahfouz. “Analysis of SDN-Based Security Challenges and Solution Approaches for SDWSN Usage”. In: *IEEE 28th International Symposium on Industrial Electronics (ISIE)*. June 2019, pp. 1288–1293. DOI: 10.1109/ISIE.2019.8781268. URL: https://ieeexplore.ieee.org/abstract/document/8781268.

[Mau96] Ueli Maurer. “Modelling a public-key infrastructure”. In: *Computer Security — ESORICS 96*. Ed. by Elisa Bertino, Helmut Kurth, Giancarlo Martella, and Emilio Montolivo. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 325–350. DOI: https://doi.org/10.1007/3-540-61770-1_45. URL: https://link.springer.com/chapter/10.1007/3-540-61770-1_45.

[Mav11] Nikos Mavrogiannopoulos. *The price to pay for perfect-forward secrecy*. Dec. 2011. URL: http://nmav.gnutls.org/2011/12/price-to-pay-for-perfect-forward.html.

[MB16] Jonathan Margulies and Michael Berg. "That Certificate You Bought Could Get You Hacked". In: *IEEE Security & Privacy* 14.5 (2016), pp. 93–95. DOI: 10.1109/MSP.2016.106. URL: https://ieeexplore.ieee.org/abstract/document/7676181.

[MBM17] Massimiliano Maggiari, Michela Bevilacqua, and Carla Marcenaro. *Monitoring carrier ethernet networks*. US Patent 9,590,881. Mar. 2017. URL: https://patents.google.com/patent/US9590881B2/en.

[McB+13] Mike McBride, Marc Cohn, Smita Deshpande, Meenakshi Kaushik, Mat Mathews, and Shaji Nathan. *SDN Security Considerations in the Data Center*. Tech. rep. Open Networking Foundation, 2013. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/solution-briefs/sb-security-data-center.pdf.

[McK+08] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, S. Shenker, and J. Turner. "OpenFlow: Enabling Innovation in Campus Networks". In: *SIGCOMM Comput. Commun. Rev.* 38.2 (Mar. 2008), pp. 69–74. DOI: 10.1145/1355734.1355746. URL: https://doi.org/10.1145/1355734.1355746.

[MD16] Diogo Menezes Ferrazani Mattos and Otto Carlos Muniz Bandeira Duarte. "AuthFlow: authentication and access control mechanism for software defined networking". In: *Annals of Telecommunications* 71.11-12 (2016), pp. 607–615. DOI: https://doi.org/10.1007/s12243-016-0505-z. URL: https://link.springer.com/article/10.1007/s12243-016-0505-z.

[MDP15] D. Mahu, V. Dumitrel, and F. Pop. "Secure Entropy Gatherer". In: *20th International Conference on Control Systems and Computer Science*. 2015, pp. 185–190. DOI: 10.1109/CSCS.2015.74. URL: https://ieeexplore.ieee.org/abstract/document/7168427.

[ME15] Konstantinos Manousakis and Georgios Ellinas. "Attack-aware planning of transparent optical networks". In: *Optical Switching and Networking* 0 (2015). DOI: http://dx.doi.org/10.1016/j.osn.2015.03.005. URL: http://www.sciencedirect.com/science/article/pii/S1573427715000302.

[MEF20] MEF Community. *MEF*. 2020. URL: https://www.mef.net/.

[Mei+13] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. "The TAMARIN Prover for the Symbolic Analysis of Security Protocols". In: *Computer Aided Verification*. Ed. by Natasha Sharygina and Helmut Veith. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 696–701. DOI: https://doi.org/10.1007/978-3-642-39799-8_48. URL: https://link.springer.com/chapter/10.1007/978-3-642-39799-8_48.

[Men+17] A. Mendiola, J. Astorga, E. Jacob, and M. Higuero. "A Survey on the Contributions of Software-Defined Networking to Traffic Engineering". In: *IEEE Communications Surveys Tutorials* 19.2 (2017), pp. 918–953. DOI: 10.1109/COMST.2016.2633579. URL: https://ieeexplore.ieee.org/abstract/document/7762818.

[Men+18] V. B. Mendiratta, L. J. Jagadeesan, R. Hanmer, and M. R. Rahman. "How Reliable Is My Software-Defined Network? Models and Failure Impacts". In: *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. Oct. 2018, pp. 83–88. DOI: 10.1109/ISSREW.2018.00-26. URL: https://ieeexplore.ieee.org/abstract/document/8539168.

[Meu13] Nicole van der Meulen. "DigiNotar: Dissecting the First Dutch Digital Disaster". In: *Journal of Strategic Security* 6.2 (2013). DOI: http://dx.doi.org/10.5038/1944-0472.6.2.4. URL: https://www.jstor.org/stable/26466760.

[MGM19] E. M. M. Manucom, B. D. Gerardo, and R. P. Medina. "Analysis of Key Randomness in Improved One-Time Pad Cryptography". In: *IEEE 13th International Conference on Anti-counterfeiting, Security, and Identification (ASID)*. Oct. 2019, pp. 11–16. DOI: 10.1109/ICASID.2019.8925173. URL: https://ieeexplore.ieee.org/abstract/document/8925173.

[MHP14] Stephanos Matsumoto, Samuel Hitz, and Adrian Perrig. "Fleet: Defending SDNs from Malicious Administrators". In: *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*. HotSDN '14. Chicago, Illinois, USA: ACM, 2014, pp. 103–108. DOI: 10.1145/2620728.2620750. URL: http://doi.acm.org/10.1145/2620728.2620750.

[Mim16] Michael Mimoso. *GPG patches 18-year-old LibGCrypt RNG bug*. 2016. URL: https://threatpost.com/gpg-patches-18-year-old-libgcrypt-rng-bug/119984/.

[MK17] O. Michel and E. Keller. “SDN in wide-area networks: A survey”. In: *Fourth International Conference on Software Defined Systems (SDS)*. May 2017, pp. 37–42. DOI: 10.1109/SDS.2017.7939138. URL: https://ieeexplore.ieee.org/document/7939138.

[Mol+19] A. Molina Zarca, D. Garcia-Carrillo, J. Bernal Bernabe, J. Ortiz, R. Marin-Perez, and A. Skarmeta. “Enabling Virtual AAA Management in SDN-Based IoT Networks”. In: *Sensors* 19.2 (2019), p. 295. DOI: https://doi.org/10.3390/s19020295. URL: https://www.mdpi.com/1424-8220/19/2/295.

[Mor+18] R. Morro et al. “Automated End to End Carrier Ethernet Provisioning Over a Disaggregated WDM Metro Network with a Hierarchical SDN Control and Monitoring Platform”. In: *European Conference on Optical Communication (ECOC)*. Sept. 2018, pp. 1–3. DOI: 10.1109/ECOC.2018.8535422. URL: https://ieeexplore.ieee.org/document/8535422.

[MR19] Guido Maier and Martin Reisslein. “Transport SDN at the dawn of the 5G era”. In: *Optical Switching and Networking* 33 (2019), pp. 34–40. DOI: https://doi.org/10.1016/j.osn.2019.02.001. URL: http://www.sciencedirect.com/science/article/pii/S1573427719300402.

[MRS19] Salvatore Manfredi, Silvio Ranise, and Giada Sciarretta. “Lost in TLS? No More! Assisted Deployment of Secure TLS Configurations”. In: *Data and Applications Security and Privacy XXXIII*. Ed. by Simon N. Foley. Cham: Springer International Publishing, 2019, pp. 201–220. DOI: https://doi.org/10.1007/978-3-030-22479-0_11. URL: https://link.springer.com/chapter/10.1007/978-3-030-22479-0_11.

[MT19] S. Midha and K. Triptahi. “Extended TLS security and Defensive Algorithm in OpenFlow SDN”. In: *9th International Conference on Cloud Computing, Data Science Engineering (Confluence)*. Jan. 2019, pp. 141–146. DOI: 10.1109/CONFLUENCE.2019.8776607. URL: https://ieeexplore.ieee.org/abstract/document/8776607.

[MTG18] P. M. Mohan, T. Truong-Huu, and M. Gurusamy. “Towards resilient in-band control path routing with malicious switch detection in SDN”. In: *10th International Conference on Communication Systems Networks (COMSNETS)*. Jan. 2018, pp. 9–16. DOI: 10.1109/COMSNETS.2018.8328174. URL: https://ieeexplore.ieee.org/document/8328174.

[MVV96] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1996. URL: http://cacr.uwaterloo.ca/hac/.

[Nam19] Namecheap.com. *Cipher Suites Configuration (and forcing Perfect Forward Secrecy)*. 2019. URL: https://www.namecheap.com/support/knowledgebase/article.aspx/9601//cipher-suites-configuration-and-forcing-perfect-forward-secrecy.

[Nay+14] David Naylor, Alessandro Finamore, Ilias Leontiadis, Yan Grunenberger, Marco Mellia, Maurizio Munafo, Konstantina Papagiannaki, and Peter Steenkiste. "The Cost of the "S" in HTTPS". In: *Proceedings of the Tenth ACM Conference on Emerging Networking Experiments and Technologies*. CoNEXT '14. Sydney, Australia: ACM, 2014, p. 7. DOI: 10.1145/2535372.2535416. URL: http://doi.acm.org/10.1145/2535372.2535416.

[NDK16] V. Nguyen, T. Do, and Y. Kim. "SDN and virtualization-based LTE mobile network architectures: A comprehensive survey". In: *Wireless Personal Communications* 86.3 (2016), pp. 1401–1438. DOI: https://doi.org/10.1007/s11277-015-2997-7. URL: https://link.springer.com/article/10.1007/s11277-015-2997-7.

[NIS18] NIST. *NIST Statistical Test Suite*. 2018. URL: http://csrc.nist.gov/groups/ST/toolkit/rng/documentation_software.html.

[Niz19] David Nizen. *10 Reasons for Network Downtime & What to Do About It*. 2019. URL: https://www.iglass.net/blog/reasons-for-network-downtime.

[NK19] Victor Netes and Margarita Kusakina. "Reliability Challenges in Software Defined Networking". In: *Proceedings of the 24th Conference of Open Innovations Association FRUCT*. FRUCT'24. Moscow, Russia: FRUCT Oy, 2019. URL: https://dl.acm.org/doi/10.5555/3338290.3338390.

[NM16] T. Nishinaga and M. Mambo. "Implementation of μNaCl on 32-bit ARM Cortex-M0". In: *IEICE TRANSACTIONS on Information and Systems* 99.8 (2016), pp. 2056–2060. DOI: 10.1587/transinf.2015INP0013. URL: https://search.ieice.org/bin/summary.php?id=e99-d_8_2056.

[Noh+16] Jiseong Noh, Seunghyeon Lee, Jaehyun Park, Seungwon Shin, and Brent B. Kang. "Vulnerabilities of network OS and mitigation with state-based permission system". In: *Security and Communication Networks* 9.13 (2016), pp. 1971–1982. DOI: 10.1002/sec.1369. eprint:

https://onlinelibrary.wiley.com/doi/pdf/10.1002/sec.1369. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.1369.

[NOL17] MARIANNA NOLL. *When Insider Threats Come From Your Privileged User*. 2017. URL: https://itsecuritycentral.teramind.co/2017/12/29/when-insider-threats-come-from-your-privileged-user/.

[NS78] Roger M. Needham and Michael D. Schroeder. "Using Encryption for Authentication in Large Networks of Computers". In: *Commun. ACM* 21.12 (Dec. 1978), pp. 993–999. DOI: 10.1145/359657.359659. URL: https://doi.org/10.1145/359657.359659.

[NT94] B Clifford Neuman and Theodore Ts'o. "Kerberos: An authentication service for computer networks". In: *IEEE Communications magazine* 32.9 (1994), pp. 33–38. DOI: 10.1109/35.312841. URL: https://ieeexplore.ieee.org/abstract/document/312841.

[Nur+14] J. R. C. Nurse, O. Buckley, P. A. Legg, M. Goldsmith, S. Creese, G. R. T. Wright, and M. Whitty. "Understanding Insider Threat: A Framework for Characterising Attacks". In: *IEEE Security and Privacy Workshops*. 2014, pp. 214–228. DOI: 10.1109/SPW.2014.38. URL: https://ieeexplore.ieee.org/document/6957307.

[Obs18] ObserveIt. *2018 Cost of Insider Threats: Global Organizations*. 2018. URL: https://www.observeit.com/ponemon-report-cost-of-insider-threats/.

[ONF13] ONF. *OpenFlow Switch Specification (Version 1.4.0)*. 2013. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf.

[ONF14] ONF. *OpenFlow Switch Specification (Version 1.5.0)*. 2014. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf.

[ONF15] ONF. *Principles and Practices for Securing Software-Defined Networks*. Tech. rep. ONF TR-511. Open Networking Foundation, 2015. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/Principles_and_Practices_for_Securing_Software-Defined_Networks_applied_to_OFv1.3.4_V1.0.pdf.

[ONF19] ONF. *Open Networking Foundation*. 2019. URL: https://www.opennetworking.org/.

[ONO16] ONOS Project. *Configuring ONOS*. 2016. URL: https://wiki.onosproject.org/display/ONOS/Configuring+ONOS.

[OO18] F. Y. Okay and S. Ozdemir. "Routing in Fog-Enabled IoT Platforms: A Survey and an SDN-Based Solution". In: *IEEE Internet of Things Journal* 5.6 (Dec. 2018), pp. 4871–4889. DOI: 10.1109/JIOT.2018.2882781. URL: https://ieeexplore.ieee.org/document/8542693.

[Ope] OpenDaylight Project. *OpenDaylight Karaf Features*. URL: https://docs.opendaylight.org/en/stable-boron/getting-started-guide/karaf_features.html.

[Ope16] OpenSSL.org. *OpenSSL Security Advisory [10 Nov 2016]*. Nov. 2016. URL: https://www.openssl.org/news/secadv/20161110.txt.

[Ope18a] OpenDaylight Project. *OpenDaylight*. 2018. URL: https://www.opendaylight.org.

[Ope18b] OpenDaylight Project. *Security Considerations*. 2018. URL: http://docs.opendaylight.org/en/stable-nitrogen/getting-started-guide/security_considerations.html.

[Ope18c] OpenDaylight Project. *SNBI User Guide*. 2018. URL: https://test-odl-docs.readthedocs.io/en/stable-boron/user-guide/snbi-user-guide.html.

[Ope18d] OpenDaylight Project. *Unified Secure Channel*. 2018. URL: https://docs.opendaylight.org/en/stable-neon/user-guide/unified-secure-channel.html.

[Ope19a] Open Whisper Systems. *Signal*. 2019. URL: https://signal.org.

[Ope19b] OpenDaylight Project. *Defense4All Overview*. 2019. URL: https://nexus.opendaylight.org/content/sites/site/org.opendaylight.docs/master/userguide/manuals/userguide/bk-user-guide/content/_defense4all_overview.html.

[Ope20] OpenDaylight Project. *Security Considerations*. 2020. URL: https://docs.opendaylight.org/en/stable-magnesium/getting-started-guide/security_considerations.html.

[OR87] Dave Otway and Owen Rees. "Efficient and Timely Mutual Authentication". In: *SIGOPS Oper. Syst. Rev.* 21.1 (Jan. 1987), pp. 8–10. DOI: 10.1145/24592.24594. URL: https://doi.org/10.1145/24592.24594.

[Par+17] Sae Hyong Park, Tae Hong Kim, Soo Myung Park, Ji Soo Shin, and Chang Gyu Lim. *SDN CONTROLLER AND METHOD OF IDENTIFYING SWITCH THEREOF*. Mar. 2017. URL: http://www.freepatentsonline.com/y2017/0093825.html.

[Par+18] Younghee Park, Hongxin Hu, Xiaohong Yuan, and Hongda Li. "Enhancing Security Education Through Designing SDN Security Labs in CloudLab". In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. SIGCSE '18. Baltimore, Maryland, USA: Association for Computing Machinery, 2018, pp. 185–190. DOI: 10.1145/3159450.3159514. URL: https://doi.org/10.1145/3159450.3159514.

[Par+19] Taejune Park, Yeonkeun Kim, Vinod Yegneswaran, Phillip Porras, Zhaoyan Xu, KyoungSoo Park, and Seungwon Shin. "DPX: Data-Plane eXtensions for SDN Security Service Instantiation". In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Ed. by Roberto Perdisci, Clémentine Maurice, Giorgio Giacinto, and Magnus Almgren. Cham: Springer International Publishing, 2019, pp. 415–437. URL: https://link.springer.com/chapter/10.1007/978-3-030-22038-9_20.

[Par19] Paragon Initiative Enterprises. *Using Libsodium in PHP Projects*. 2019. URL: https://paragonie.com/book/pecl-libsodium.

[Pas14] Matthew Pascucci. *Evaluating the Security of Software Defined Networking*. Tech. rep. TechTarget, 2014. URL: http://cdn.ttgtmedia.com/searchNetworking/Downloads/NetSecurity+November+2014+Incentive%20(1).pdf.

[Pat19] Andy Patrizio. *The biggest risk to uptime? Your staff*. 2019. URL: https://www.networkworld.com/article/3444762/the-biggest-risk-to-uptime-your-staff.html.

[Per+02] Adrian Perrig, Robert Szewczyk, J. D. Tygar, Victor Wen, and David E. Culler. "SPINS: Security Protocols for Sensor Networks". In: *Wireless Networks* 8.5 (Sept. 2002), pp. 521–534. DOI: 10.1023/A:1016598314198. URL: https://link.springer.com/article/10.1023/A:1016598314198.

[Pet+13] W. Michael Petullo, Xu Zhang, Jon A. Solworth, Daniel J. Bernstein, and Tanja Lange. "MinimaLT: Minimal-latency Networking Through Better Security". In: *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. CCS '13. Berlin, Germany: ACM, 2013, pp. 425–438. DOI: 10.1145/2508859.2516737. URL: http://doi.acm.org/10.1145/2508859.2516737.

[Pic18] Pica8 Open Networking. *PicOS Overview*. 2018. URL: https://www.pica8.com/wp-content/uploads/pica8-whitepaper-picos-overview.pdf.

[Pic19] Pica8 Inc. *Pica8*. 2019. URL: https://www.pica8.com/.

[Pon20] Ponemon Institute. *2020 Cost of Insider Threats Global Report*. Tech. rep. Observe IT and IBM Security, 2020. URL: https://www.observeit.com/cost-of-insider-threats/.

[Por+12] Philip Porras, Seungwon Shin, Vinod Yegneswaran, Martin Fong, Mabry Tyson, and Guofei Gu. "A Security Enforcement Kernel for OpenFlow Networks". In: *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*. HotSDN '12. Helsinki, Finland: Association for Computing Machinery, 2012, pp. 121–126. DOI: 10.1145/2342441.2342466. URL: https://doi.org/10.1145/2342441.2342466.

[Por+15] Phillip A Porras, Steven Cheung, Martin W Fong, Keith Skinner, and Vinod Yegneswaran. "Securing the Software Defined Network Control Layer." In: *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. 2015. URL: http://www.csl.sri.com/papers/sefloodlight/.

[PP17] H. Polat and O. Polat. "The effects of DoS attacks on ODL and POX SDN controllers". In: *8th International Conference on Information Technology (ICIT)*. May 2017, pp. 554–558. DOI: 10.1109/ICITECH.2017.8080058. URL: https://ieeexplore.ieee.org/document/8080058.

[PPJ11] Jianli Pan, Subharthi Paul, and Raj Jain. "A survey of the research on future internet architectures". In: *IEEE Communications Magazine* 49.7 (2011). DOI: 10.1109/MCOM.2011.5936152. URL: https://ieeexplore.ieee.org/abstract/document/5936152.

[Pre15] Bart Preneel. *System security after Snowden*. Keynote Speak at the 45th Annual IEEE/IFIP DSN. 2015. URL: http://2015.dsn.org/keynote-speakers/.

[PST18] M. Paliwal, D. Shrimankar, and O. Tembhurne. "Controllers in SDN: A Review Report". In: *IEEE Access* 6 (2018), pp. 36256–36270. DOI: 10.1109/ACCESS.2018.2846236. URL: https://ieeexplore.ieee.org/document/8379403.

[PST20] Christian Paquin, Douglas Stebila, and Goutam Tamvada. "Benchmarking Post-quantum Cryptography in TLS". In: *Post-Quantum Cryptography*. Ed. by Jintai Ding and Jean-Pierre Tillich. Cham: Springer International Publishing, 2020, pp. 72–91. URL: `https://link.springer.com/chapter/10.1007/978-3-030-44223-1_5`.

[PwC14] PwC, CSO magazine and CERT/CMU. *US cybercrime: Rising risks, reduced readiness*. Tech. rep. PwC, 2014, p. 21. URL: `http://www.pwc.com/us/en/increasing-it-effectiveness/publications/assets/2014-us-state-of-cybercrime.pdf`.

[Raj+11] Himanshu Raj, David Robinson, Talha Bin Tariq, Paul England, Stefan Saroiu, and Alec Wolman. *Credo: Trusted Computing for Guest VMs with a Commodity Hypervisor*. Tech. rep. MSR-TR-2011-130. 2011. URL: `http://research.microsoft.com/apps/pubs/default.aspx?id=157213`.

[Raw15] Kristi Rawlinson. *Security Threat Landscape Still Plagued by Known Issues, says HP*. 2015. URL: `https://www8.hp.com/mx/es/hp-news/press-release.html?id=1915228`.

[Raz+17] Abbas Razaghpanah, Arian Akhavan Niaki, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Johanna Amann, and Phillipa Gill. "Studying TLS Usage in Android Apps". In: *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*. CoNEXT '17. Incheon, Republic of Korea: Association for Computing Machinery, 2017, pp. 350–362. DOI: `10.1145/3143361.3143400`. URL: `https://doi.org/10.1145/3143361.3143400`.

[Red18] Red Hat, Inc. *OpenShift SDN*. 2018. URL: `https://docs.openshift.com/container-platform/3.7/architecture/networking/sdn.html`.

[RHE16] Fabian Ruffy, Wolfgang Hommel, and Felix von Eye. "A STRIDE-based Security Architecture for Software-Defined Networking". In: *The Fifteenth International Conference on Networks*. IARIA. 2016, p. 107. URL: `http://cial.csie.ncku.edu.tw/presentation/group_pdf/A%20STRIDE-based%20Security%20Architecture%20for%20Software-Defined%20Networking.pdf`.

[RLM19] J. D. Roth, C. E. Lutton, and J. B. Michael. "Security Through Simplicity: A Case Study in Logical Segmentation Inference". In: *Computer* 52.7 (July 2019), pp. 76–79. DOI: `10.1109/MC.2019.2906443`. URL: `https://ieeexplore.ieee.org/document/8747213`.

[RME13] R.M. Ramos, M. Martinello, and C. Esteve Rothenberg. “SlickFlow: Resilient source routing in Data Center Networks unlocked by OpenFlow”. In: *IEEE 38th Conference on Local Computer Networks (LCN)*. 2013, pp. 606–613. DOI: 10.1109/LCN.2013.6761297. URL: https://ieeexplore.ieee.org/document/6761297.

[RR17] Danda B Rawat and Swetha R Reddy. “Software defined networking architecture, security and energy efficiency: A survey”. In: *IEEE Communications Surveys & Tutorials* 19.1 (2017), pp. 325–346. DOI: 10.1109/COMST.2016.2618874. URL: https://ieeexplore.ieee.org/abstract/document/7593247.

[RWW17] G. Rzepka, S. Wenke, and S. Walling. “Choose simplicity for a better digital substation design”. In: *70th Annual Conference for Protective Relay Engineers (CPRE)*. Apr. 2017, pp. 1–9. DOI: 10.1109/CPRE.2017.8090016. URL: https://ieeexplore.ieee.org/document/8090016.

[Ryu19] Ryu SDN Framework Community. *Ryu component-based software defined networking framework*. 2019. URL: https://github.com/faucetsdn/ryu.

[Saf12] SafeNet, Inc. *Security and Handling Issues - HSM Appliance*. 2012. URL: http://cloudhsm-safenet-docs-5.3.s3-website-us-east-1.amazonaws.com/007-011136-006_lunasa_5-3_webhelp_rev-c/startpage.htm#administration/physical_security_guidance.htm.

[Sah+17] Rishikesh Sahay, Gregory Blanc, Zonghua Zhang, Khalifa Toumi, and Hervé Debar. “Adaptive Policy-Driven Attack Mitigation in SDN”. In: *Proceedings of the 1st International Workshop on Security and Dependability of Multi-Domain Infrastructures*. New York, NY, USA: Association for Computing Machinery, 2017. DOI: 10.1145/3071064.3071068. URL: https://doi.org/10.1145/3071064.3071068.

[Sal+16] O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab. “SDN controllers: A comparative study”. In: *18th Mediterranean Electrotechnical Conference (MELECON)*. 2016, pp. 1–6. DOI: 10.1109/MELCON.2016.7495430. URL: https://ieeexplore.ieee.org/abstract/document/7495430.

[Sam15] D. Samociuk. “Secure communication between openflow switches and controllers”. In: *The Seventh International Conference on Advances in Future Internet (AFIN)*. 2015, p. 39. URL: https://www.thinkmind.org/download.php?articleid=afin_2015_2_30_40047.

[Sas+16] T. Sasaki, C. Pappas, T. Lee, T. Hoefler, and A. Perrig. “SDNsec: Forwarding Accountability for the SDN Data Plane”. In: *25th International Conference on Computer Communication and Networks (ICCCN)*. 2016, pp. 1–10. DOI: 10.1109/ICCCN.2016.7568569. URL: https://ieeexplore.ieee.org/document/7568569.

[Sch11] Scott Schenker. *The Future of Networking, and the Past of Protocols.* 2011. URL: http://www.youtube.com/watch?v=YHeyuD89n1Y.

[Sch12] Bruce Schneier. *Lousy Random Numbers Cause Insecure Public Keys.* 2012. URL: https://www.schneier.com/blog/archives/2012/02/lousy_random_nu.html.

[Sch15] Bruce Schneier. *Data and Goliath: The hidden battles to collect your data and control your world.* WW Norton & Company, 2015. URL: https://www.schneier.com/books/data_and_goliath/.

[Sch19] Holger Schulze. *Cloud Security Report.* 2019. URL: https://www.cybersecurity-insiders.com/portfolio/2019-cloud-security-report-isc2/.

[Sco17] Sandra Scott-Hayward. “Trailing the Snail: SDN Controller Security Evolution”. In: *CoRR* abs/1711.08406 (2017). arXiv: 1711.08406. URL: http://arxiv.org/abs/1711.08406.

[SD16] Ameya Sanzgiri and Dipankar Dasgupta. “Classification of Insider Threat Detection Techniques”. In: *Proceedings of the 11th Annual Cyber and Information Security Research Conference.* CISRC ’16. Oak Ridge, TN, USA: ACM, 2016, 25:1–25:4. DOI: 10.1145/2897795.2897799. URL: http://doi.acm.org/10.1145/2897795.2897799.

[SDH16] Drew Springall, Zakir Durumeric, and J. Alex Halderman. “Measuring the Security Harm of TLS Crypto Shortcuts”. In: *Proceedings of the Internet Measurement Conference.* IMC ’16. Santa Monica, California, USA: ACM, 2016, pp. 33–47. DOI: 10.1145/2987443.2987480. URL: http://doi.acm.org/10.1145/2987443.2987480.

[SDW17] Dominik Schürmann, Sergej Dechand, and Lars Wolf. “OpenKeychain: An Architecture for Cryptography with Smart Cards and NFC Rings on Android”. In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1.3 (2017), p. 99.

[SDx14] SDxCentral. *What is White Box Switching & White Box Switches (& are they SDN Switches)?* 2014. URL: https://www.sdxcentral.com/data-center/bare-metal/white-box/definitions/what-is-white-box-networking/.

[Sec+17] Stefano Secci, Kamel Attou, Dung Chi Phung, Sandra Scott-Hayward, Dylan Smyth, Suchitra Vemuri, and You Wang. *ONOS Security and Performance Analysis: Report No. 1*. Tech. rep. ONOS Project, 2017. URL: https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2017/07/ONOS-security-and-performance-analysis-brigade-report-no1.pdf.

[Sec+19] Stefano Secci, Alessio Diamanti, Jose Manuel Vilchez Sanchez, Mamadou Tahirou Bah, Petra Vizarreta, Carmen Mas Machuca, Sandra Scott-Hayward, and Dylan Smith. *Security and Performance Comparison of ONOS and ODL controllers*. Tech. rep. Informational Report, Open Networking Foundation, 2019. URL: https://mediatum.ub.tum.de/doc/1525888/file.pdf.

[Sec19] Securonix, Inc. *2019 Insider Threat Survey Report*. 2019. URL: https://www.securonix.com/resources/2019-insider-threat-survey-report.

[SG13] Seungwon Shin and Guofei Gu. "Attacking Software-Defined Networks: A First Feasibility Study". In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. HotSDN '13. Hong Kong, China: Association for Computing Machinery, 2013, pp. 165–166. DOI: 10.1145/2491185.2491220. URL: https://doi.org/10.1145/2491185.2491220.

[Sha+17] G. Shang, P. Zhe, X. Bin, H. Aiqun, and R. Kui. "FloodDefender: Protecting data and control plane resources under SDN-aimed DoS attacks". In: *IEEE Conference on Computer Communications (INFOCOM)*. 2017, pp. 1–9. DOI: 10.1109/INFOCOM.2017.8057009. URL: https://ieeexplore.ieee.org/document/8057009.

[Sha+20] Arash Shaghaghi, Mohamed Ali Kaafar, Rajkumar Buyya, and Sanjay Jha. "Software-Defined Network (SDN) Data Plane Security: Issues, Solutions, and Future Directions". In: *Handbook of Computer Networks and Cyber Security: Principles and Paradigms*. Ed. by Brij B. Gupta, Gregorio Martinez Perez, Dharma P. Agrawal, and Deepak Gupta. Cham: Springer International Publishing, 2020, pp. 341–387. DOI: 10.1007/978-3-030-22277-2_14. URL: https://doi.org/10.1007/978-3-030-22277-2_14.

[Sha15] Adi Shamir. *Post-Snowden Cryptography*. 2015. URL: https://wwwfr.uni.lu/snt/distinguished_lectures/post_snowden_cryptography_june_11_2015.

[SHC19] Nicolas Serrano, Hilda Hadan, and L. Jean Camp. “A Complete Study of P.K.I. (PKI’s Known Incidents)”. In: *SSRN* (2019). DOI: http://dx.doi.org/10.2139/ssrn.3425554. URL: https://ssrn.com/abstract=3425554.

[She+12] Charles Shen, Erich Nahum, Henning Schulzrinne, and Charles P Wright. “The impact of TLS on SIP server performance: measurement and modeling”. In: *IEEE/ACM Transactions on Networking (TON)* 20.4 (2012), pp. 1217–1230. DOI: 10.1109/TNET.2011.2180922. URL: https://ieeexplore.ieee.org/abstract/document/6129420.

[Shi+13a] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, and M. Tyson. “FRESCO: Modular Composable Security Services for Software-Defined Networks.” In: *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. 2013. URL: https://www.ndss-symposium.org/ndss2013/ndss-2013-programme/fresco-modular-composable-security-services-software-defined-networks/.

[Shi+13b] Seungwon Shin, Phillip Porras, Vinod Yegneswaran, and Guofei Gu. “A framework for integrating security services into software-defined networks”. In: *Proceedings of the open networking summit (Research Track poster paper)*. ONS’13. Open Networking Summit, 2013. URL: http://hdl.handle.net/10203/205988.

[Shi+13c] Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. “AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-Defined Networks”. In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. CCS ’13. Berlin, Germany: Association for Computing Machinery, 2013, pp. 413–424. DOI: 10.1145/2508859.2516684. URL: https://doi.org/10.1145/2508859.2516684.

[Shi+14] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh, and B. B. Kang. “Rosemary: A Robust, Secure, and High-Performance Network Operating System”. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’14. Scottsdale, Arizona, USA: Association for Computing Machinery, 2014, pp. 78–89. DOI: 10.1145/2660267.2660353. URL: https://doi.org/10.1145/2660267.2660353.

[SHS15] Y. Sheffer, R. Holz, and P. Saint-Andre. *Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport*

Layer Security (DTLS). RFC 7525. Internet Engineering Task Force, May 2015. URL: https://tools.ietf.org/html/rfc7525.

[SKD20] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. “Post-Quantum Authentication in TLS 1.3: A Performance Study”. In: *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. 2020, pp. 1–16. URL: https://www.ndss-symposium.org/wp-content/uploads/2020/02/24203-paper.pdf.

[SM11] Jurgen Schonwalder and Vladislav Marinov. “On the Impact of Security Protocols on the Performance of SNMP”. In: *IEEE Transactions on Network and Service Management* 8.1 (2011), pp. 52–64. DOI: 10.1109/TNSM.2011.012111.00011. URL: https://ieeexplore.ieee.org/abstract/document/5702353.

[SNS16] S. Scott-Hayward, S. Natarajan, and S. Sezer. “A Survey of Security in Software Defined Networks”. In: *IEEE Communications Surveys Tutorials* 18.1 (2016), pp. 623–654. DOI: 10.1109/COMST.2015.2453114. URL: https://ieeexplore.ieee.org/abstract/document/7150550.

[Sof09] R. C. Sofia. “A survey of advanced Ethernet forwarding approaches”. In: *IEEE Communications Surveys Tutorials* 11.1 (2009), pp. 92–115. DOI: 10.1109/SURV.2009.090108. URL: https://ieeexplore.ieee.org/document/4796929.

[Son+17] S. Song, H. Park, B. Y. Choi, T. Choi, and H. Zhu. “Control Path Management Framework for Enhancing Software-Defined Network (SDN) Reliability”. In: *IEEE Transactions on Network and Service Management* 14.2 (2017), pp. 302–316. DOI: 10.1109/TNSM.2017.2669082. URL: https://ieeexplore.ieee.org/document/7855810.

[Son18] Do Son. *OpenFlow Protocol Exposure Authentication Vulnerability*. 2018. URL: https://securityonline.info/openflow-protocol-exposure-authentication-vulnerability/.

[SOP20] SOPHOS. *Sophos 2020 Threat Report*. 2020. URL: https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/sophoslabs-uncut-2020-threat-report.pdf.

[SOS13] S. Scott-Hayward, G. O’Callaghan, and S. Sezer. “Sdn Security: A Survey”. In: *IEEE SDN for Future Networks and Services (SDN4FNS)*. 2013, pp. 1–7. DOI: 10.1109/SDN4FNS.2013.6702553. URL: https://ieeexplore.ieee.org/document/6702553.

[Sou+10] P. Sousa, A. N. Bessani, M. Correia, N. F. Neves, and P. Verissimo. “Highly Available Intrusion-Tolerant Services with Proactive-Reactive Recovery”. In: *IEEE Transactions on Parallel and Distributed Systems* 21.4 (2010), pp. 452–465. DOI: 10.1109/TPDS.2009.83. URL: https://ieeexplore.ieee.org/abstract/document/5010435.

[Sta15] Stanford University. *Stanford cybersecurity expert: Dan Boneh.* 2015. URL: https://www.youtube.com/watch?v=H-YGdcNFBJk.

[Sta17] Philip B. Stark. *Don’t Bet on your Random Number Generator.* 2017. URL: https://github.com/pbstark/pseudorandom/blob/master/prngLux17.ipynb.

[Ste15] Harlan Stenn. “Securing Network Time Protocol”. In: *Commun. ACM* 58.2 (Jan. 2015), pp. 48–51. DOI: 10.1145/2697397. URL: http://doi.acm.org/10.1145/2697397.

[Str16] Falko Strenzke. “An Analysis of OpenSSL’s Random Number Generator”. In: *Advances in Cryptology – EUROCRYPT 2016.* Ed. by Marc Fischlin and Jean-Sébastien Coron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 644–669. DOI: https://doi.org/10.1007/978-3-662-49890-3_25. URL: https://link.springer.com/chapter/10.1007/978-3-662-49890-3_25.

[Sul+19] Nasrin Sultana, Naveen Chilamkurti, Wei Peng, and Rabei Alhadad. “Survey on SDN based network intrusion detection system using machine learning approaches”. In: *Peer-to-Peer Networking and Applications* 12.2 (2019), pp. 493–501. DOI: https://doi.org/10.1007/s12083-017-0630-0. URL: https://link.springer.com/article/10.1007/s12083-017-0630-0.

[Sym16] Symantec. *Internet Security Threat Report.* 2016. URL: https://www.insight.com/content/dam/insight-web/en_US/article-images/whitepapers/partner-whitepapers/Internet%20Security%20Threat%20Report.pdf.

[TB19] N. Tuveri and B. B. Brumley. “Start Your ENGINEs: Dynamically Loadable Contemporary Crypto”. In: *IEEE Cybersecurity Development (SecDev).* Sept. 2019, pp. 4–19. DOI: 10.1109/SecDev.2019.00014. URL: https://ieeexplore.ieee.org/document/8901574.

[TFS19] Samuel Thibault, Mike Frysinger, and Andreas Schwab. *OpenDaylight – Languages.* 2019. URL: https://www.openhub.net/p/opendaylight/analyses/latest/languages_summary.

[The19a] The Haystax Team. *Insider Threat Report*. Tech. rep. Cybersecurity Insiders, 2019. URL: https://haystax.com/wp-content/uploads/2019/07/Haystax-Insider-Threat-Report-2019.pdf.

[The19b] The OpenStack project. *OpenStack*. 2019. URL: https://www.openstack.org/.

[The19c] The Sodium crypto library (libsodium). *Libsodium users*. 2019. URL: https://download.libsodium.org/doc/libsodium_users/.

[Thi+18] Kashyap Thimmaraju, Bhargava Shastry, Tobias Fiebig, Felicitas Hetzelt, Jean-Pierre Seifert, Anja Feldmann, and Stefan Schmid. "Taking Control of SDN-based Cloud Systems via the Data Plane". In: *Proceedings of the Symposium on SDN Research*. SOSR '18. Los Angeles, CA, USA: ACM, 2018, 1:1–1:15. DOI: 10.1145/3185467.3185468. URL: http://doi.acm.org/10.1145/3185467.3185468.

[Thy+16] Akhilesh S Thyagaturu, Anu Mercian, Michael P McGarry, Martin Reisslein, and Wolfgang Kellerer. "Software defined optical networks (SDONs): A comprehensive survey". In: *IEEE Communications Surveys & Tutorials* 18.4 (2016), pp. 2738–2786. DOI: 10.1109/COMST.2016.2586999. URL: https://ieeexplore.ieee.org/abstract/document/7503119.

[TK14] Umesh Tiwari and Santosh Kumar. "Cyclomatic Complexity Metric for Component Based Software". In: *SIGSOFT Softw. Eng. Notes* 39.1 (Feb. 2014), pp. 1–6. DOI: 10.1145/2557833.2557853. URL: https://doi.org/10.1145/2557833.2557853.

[TNK18] Y. Tseng, F. Nait-Abdesselam, and A. Khokhar. "SENAD: Securing Network Application Deployment in Software Defined Networks". In: *IEEE International Conference on Communications (ICC)*. May 2018, pp. 1–6. DOI: 10.1109/ICC.2018.8422405. URL: https://ieeexplore.ieee.org/document/8422405.

[Tos+14] U. Toseef, A. Zaalouk, T. Rothe, M. Broadbent, and K. Pentikousis. "C-BAS: Certificate-Based AAA for SDN Experimental Facilities". In: *Third European Workshop on Software Defined Networks*. 2014, pp. 91–96. DOI: 10.1109/EWSDN.2014.41. URL: https://ieeexplore.ieee.org/document/6984058.

[Tro+16] Celio Trois, Marcos D Del Fabro, Luis CE de Bona, and Magnos Martinello. "A survey on SDN programming languages: toward a taxonomy". In: *IEEE Communications Surveys & Tutorials* 18.4 (2016), pp. 2687–2712. DOI: 10.1109/COMST.2016.2553778. URL: https://ieeexplore.ieee.org/abstract/document/7452335.

[TSS17] K. Thimmaraju, L. Schiff, and S. Schmid. “Outsmarting Network Security with SDN Teleportation”. In: *IEEE European Symposium on Security and Privacy (EuroS P)*. IEEE. 2017, pp. 563–578. DOI: 10.1109/EuroSP.2017.21. URL: https://ieeexplore.ieee.org/abstract/document/7962003.

[TT19] A. Takahashi and M. Tibouchi. “Degenerate Fault Attacks on Elliptic Curve Parameters in OpenSSL”. In: *IEEE European Symposium on Security and Privacy (EuroS P)*. June 2019, pp. 371–386. DOI: 10.1109/EuroSP.2019.00035. URL: https://ieeexplore.ieee.org/document/8806763.

[TZN16] Yuchia Tseng, Zonghua Zhang, and Farid Naït-Abdesselam. “ControllerSEPA: A Security-Enhancing SDN Controller Plug-in for OpenFlow Applications”. In: *17th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*. IEEE. 2016, pp. 268–273. DOI: 10.1109/PDCAT.2016.064. URL: https://ieeexplore.ieee.org/abstract/document/7943369.

[VAM15] VAMPIRE. *SUPERCOP*. 2015. URL: http://bench.cr.yp.to/supercop.html.

[Vau18] Steven Vaughan-Nichols. *How to secure your server room*. 2018. URL: https://www.hpe.com/us/en/insights/articles/how-to-secure-your-server-room-1809.html.

[Ver+13] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung, and Paulo Verissimo. “Efficient Byzantine Fault-Tolerance”. In: *IEEE Trans. Computers* 62.1 (2013), pp. 16–30. DOI: 10.1109/TC.2011.221. URL: http://doi.ieeecomputersociety.org/10.1109/TC.2011.221.

[Ver06] Paulo E. Verissimo. “Travelling Through Wormholes: A New Look at Distributed Systems Models”. In: *SIGACT News* 37.1 (Mar. 2006), pp. 66–81. DOI: 10.1145/1122480.1122497. URL: http://doi.acm.org/10.1145/1122480.1122497.

[Ver15] Verizon. *2015 Data Breach Investigations Report*. Tech. rep. Verizon, 2015. URL: https://www.verizon.com/about/news/2015-data-breach-report-info.

[Ver19] Verizon. *Insider Threat Report*. Tech. rep. USA: Verizon, 2019. URL: https://enterprise.verizon.com/resources/reports/insider-threat-report/.

[VFV15] C Van Bruggen, NF Feddes, and M Vermeer. *Anonymous HD Video Streaming for Android using Tribler*. 2015. URL: https://repository.tudelft.nl/islandora/object/uuid:5bd6da9f-1464-4203-a383-321ab34d4386.

[VH14] Apostol Vassilev and Timothy A. Hall. "The Importance of Entropy to Information Security". In: *Computer* 47.2 (2014), pp. 78–81. DOI: 10.1109/MC.2014.47. URL: http://doi.ieeecomputersociety.org/10.1109/MC.2014.47.

[Viz+18] P. Vizarreta, K. Trivedi, B. Helvik, P. Heegaard, Andreas Blenk, W. Kellerer, and C. Mas Machuca. "Assessing the Maturity of SDN Controllers With Software Reliability Growth Models". In: *IEEE Transactions on Network and Service Management* 15.3 (Sept. 2018), pp. 1090–1104. DOI: 10.1109/TNSM.2018.2848105. URL: https://ieeexplore.ieee.org/document/8386840.

[Viz+19] P. Vizarreta, E. Sakic, W. Kellerer, and C. M. Machuca. "Mining Software Repositories for Predictive Modelling of Defects in SDN Controller". In: *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. Apr. 2019, pp. 80–88. URL: https://ieeexplore.ieee.org/abstract/document/8717837.

[Viz18] P. Vizarreta. "Modelling, Design and Optimization of Dependable Softwarized Networks for Industrial Applications". In: *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. Oct. 2018, pp. 170–173. DOI: 10.1109/ISSREW.2018.000-3. URL: https://ieeexplore.ieee.org/document/8539191.

[VM15] Vladimir Vujović and Mirjana Maksimović. "Raspberry Pi as a Sensor Web node for home automation". In: *Computers & Electrical Engineering* 44 (2015), pp. 153–171. DOI: https://doi.org/10.1016/j.compeleceng.2015.01.019. URL: http://www.sciencedirect.com/science/article/pii/S0045790615000257.

[VMw20] VMware, Inc. *NSX Data Center*. 2020. URL: https://www.vmware.com/products/nsx.html.

[Voe+13] Andreas Voellmy, Junchang Wang, Y Richard Yang, Bryan Ford, and Paul Hudak. "Maple: Simplifying SDN Programming Using Algorithmic Policies". In: *SIGCOMM Comput. Commun. Rev.* 43.4 (Aug. 2013), pp. 87–98. DOI: 10.1145/2534169.2486030. URL: https://doi.org/10.1145/2534169.2486030.

[VR12] Paulo Verissimo and Luis Rodrigues. *Distributed systems for system architects*. Vol. 1. Springer Science & Business Media, 2012.

[Wal08] John Walker. *ent - A Pseudorandom Number Sequence Test Program.* 2008. URL: http://www.fourmilab.ch/random/.

[Wan+13] Rui Wang, Yuchen Zhou, Shuo Chen, Shaz Qadeer, David Evans, and Yuri Gurevich. "Explicating SDKs: Uncovering Assumptions Underlying Secure Authentication and Authorization". In: *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*. Washington, D.C.: USENIX, 2013, pp. 399–314. URL: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/wang_rui.

[Wan+14] Huangxin Wang, Quan Jia, Dan Fleck, Walter Powell, Fei Li, and Angelos Stavrou. "A moving target DDoS defense mechanism". In: *Computer Communications* 46.0 (2014), pp. 10–21. DOI: http://dx.doi.org/10.1016/j.comcom.2014.03.009. URL: http://www.sciencedirect.com/science/article/pii/S0140366414000954.

[Wan+15] An Wang, Yang Guo, Fang Hao, T. V. Lakshman, and Songqing Chen. "UMON: Flexible and Fine Grained Traffic Monitoring in Open VSwitch". In: *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. CoNEXT '15. Heidelberg, Germany: Association for Computing Machinery, 2015. DOI: 10.1145/2716281.2836100. URL: https://doi.org/10.1145/2716281.2836100.

[Wan+18] P. Wang, H. Xu, L. Huang, C. Qian, S. Wang, and Y. Sun. "Minimizing Controller Response Time Through Flow Redirecting in SDNs". In: *IEEE/ACM Transactions on Networking* PP.99 (2018), pp. 1–14. DOI: 10.1109/TNET.2017.2786268. URL: https://ieeexplore.ieee.org/document/8248798.

[Wan+19a] C. Wang, Y. Zhang, X. Chen, K. Liang, and Z. Wang. "SDN-Based Handover Authentication Scheme for Mobile Edge Computing in Cyber-Physical Systems". In: *IEEE Internet of Things Journal* 6.5 (Oct. 2019), pp. 8692–8701. DOI: 10.1109/JIOT.2019.2922979. URL: https://ieeexplore.ieee.org/document/8736826.

[Wan+19b] X. Wang, K. Sun, A. Batcheller, and S. Jajodia. "Detecting "0-Day" Vulnerability: An Empirical Study of Secret Security Patch in OSS". In: *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. June 2019, pp. 485–492. DOI: 10.1109/DSN.2019.00056. URL: https://ieeexplore.ieee.org/document/8809499.

[Wan+19c] Yingjie Wang, Xing Liu, Weixuan Mao, and Wei Wang. "DCDroid: Automated Detection of SSL/TLS Certificate Verification Vulnerabilities in Android Apps". In: *Proceedings of the ACM Turing Celebration Conference - China*. ACM TURC '19. Chengdu, China: Association for Computing Machinery, 2019. DOI: 10.1145/3321408.3326665. URL: https://doi.org/10.1145/3321408.3326665.

[Wan+20] Ziyu Wang, Hui Yu, Zongyang Zhang, Jiaming Piao, and Jianwei Liu. "ECDSA weak randomness in Bitcoin". In: *Future Generation Computer Systems* 102 (2020), pp. 507–513. DOI: https://doi.org/10.1016/j.future.2019.08.034. URL: http://www.sciencedirect.com/science/article/pii/S0167739X17330030.

[Wan14] Shie-Yuan Wang. "Comparison of SDN OpenFlow network simulator and emulators: EstiNet vs. Mininet". In: *IEEE Symposium on Computers and Communication (ISCC)*. IEEE. 2014, pp. 1–6. DOI: 10.1109/ISCC.2014.6912609. URL: https://ieeexplore.ieee.org/abstract/document/6912609.

[WAv17] T. Wan, A. Abdou, and P. C. van Oorschot. "A Framework and Comparative Analysis of Control Plane Security of SDN and Conventional Networks". In: *ArXiv e-prints* (Mar. 2017). arXiv: 1703.06992 [cs.NI]. URL: https://arxiv.org/abs/1703.06992.

[WH13] M. Wasserman and S. Hartman. *Security Analysis of the Open Networking Foundation (ONF) OpenFlow Switch Specification*. Internet Engineering Task Force, 2013. URL: https://datatracker.ietf.org/doc/draft-mrw-sdnsec-openflow-analysis/.

[WK16] Dan Williams and Ricardo Koller. "Unikernel monitors: extending minimalism outside of the box". In: *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*. USENIX Association. 2016. URL: https://www.usenix.org/conference/hotcloud16/workshop-program/presentation/williams.

[Wu+98] Thomas D Wu et al. "The Secure Remote Password Protocol." In: *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Vol. 98. 1998, pp. 97–111. URL: https://www.ndss-symposium.org/ndss1998/secure-remote-password-protocol/.

[WW18] Li Wang and Dinghao Wu. "Bridging the Gap Between Security Tools and SDN Controllers". In: *EAI Endorsed Transactions on Security and Safety* 5.17 (2018). URL: https://eudl.eu/doi/10.4108/eai.10-1-2019.156242.

[Xie+19] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, and Y. Liu. "A Survey of Machine Learning Techniques Applied to Software Defined Networking (SDN): Research Issues and Challenges". In: *IEEE Communications Surveys Tutorials* 21.1 (Jan. 2019), pp. 393–430. DOI: 10.1109/COMST.2018.2866942. URL: https://ieeexplore.ieee.org/document/8444669.

[Xu+17] Lei Xu, Jeff Huang, Sungmin Hong, Jialong Zhang, and Guofei Gu. "Attacking the Brain: Races in the SDN Control Plane". In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017, pp. 451–468. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/xu-lei.

[Yan+15] Mao Yang, Yong Li, Depeng Jin, Lieguang Zeng, Xin Wu, and Athanasios V Vasilakos. "Software-defined and virtualized future mobile and wireless networks: A survey". In: *Mobile Networks and Applications* 20.1 (2015), pp. 4–18. DOI: https://doi.org/10.1007/s11036-014-0533-8. URL: https://link.springer.com/article/10.1007/s11036-014-0533-8.

[Yan+16] Q. Yan, F. R. Yu, Q. Gong, and J. Li. "Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments: A Survey, Some Research Issues, and Challenges". In: *IEEE Communications Surveys Tutorials* 18.1 (Jan. 2016), pp. 602–622. DOI: 10.1109/COMST.2015.2487361. URL: https://ieeexplore.ieee.org/document/7289347.

[YB14] Y. Yarom and N. Benger. "Recovering OpenSSL ECDSA Nonces Using the FLUSH+ RELOAD Cache Side-channel Attack." In: *IACR Cryptology ePrint Archive* 2014 (2014), p. 140. URL: https://eprint.iacr.org/2014/140.pdf.

[Yig+19] B. Yigit, G. Gur, B. Tellenbach, and F. Alagoz. "Secured Communication Channels in Software-Defined Networks". In: *IEEE Communications Magazine* 57.10 (Oct. 2019), pp. 63–69. DOI: 10.1109/MCOM.001.1900060. URL: https://ieeexplore.ieee.org/document/8875716.

[YJ15] Jiaqi Yan and Dong Jin. "VT-Mininet: Virtual-Time-Enabled Mininet for Scalable and Accurate Software-Define Network Emulation". In: *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*. SOSR '15. Santa Clara, California: Association for Computing Machinery, 2015. DOI: 10.1145/2774993.2775012. URL: https://doi.org/10.1145/2774993.2775012.

[Yoo+17a] Changhoon Yoon, Seungsoo Lee, Heedo Kang, Taejune Park, Seungwon Shin, V. Yegneswaran, Phillip Porras, and Guofei Gu. "Flow Wars: Systemizing the Attack Surface and Defenses in Software-Defined Networks". In: *IEEE/ACM Transactions on Networking* 25.6 (2017), pp. 3514–3530. DOI: 10.1109/TNET.2017.2748159. URL: https://ieeexplore.ieee.org/abstract/document/8048353.

[Yoo+17b] Changhoon Yoon, Seungwon Shin, Phillip Porras, Vinod Yegneswaran, Heedo Kang, Martin Fong, Brian O'Connor, and Thomas Vachuska. "A Security-Mode for Carrier-Grade SDN Controllers". In: *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACSAC 2017. Orlando, FL, USA: ACM, 2017, pp. 461–473. DOI: 10.1145/3134600.3134603. URL: http://doi.acm.org/10.1145/3134600.3134603.

[YR15] Jiangshan Yu and Mark D. Ryan. "Device Attacker Models: Fact and Fiction". In: *Security Protocols XXIII*. Ed. by Bruce Christianson, Petr Švenda, Vashek Matyáš, James Malcolm, Frank Stajano, and Jonathan Anderson. Cham: Springer International Publishing, 2015, pp. 158–167. DOI: https://doi.org/10.1007/978-3-319-26096-9_17. URL: https://link.springer.com/chapter/10.1007/978-3-319-26096-9_17.

[YRC18] J. Yu, M. Ryan, and C. Cremers. "DECIM: Detecting Endpoint Compromise In Messaging". In: *IEEE Transactions on Information Forensics and Security* 13.1 (2018), pp. 106–118. DOI: 10.1109/TIFS.2017.2738609. URL: https://ieeexplore.ieee.org/document/8007243.

[Yu18] Jiangshan Yu. *Tamarin models for ANCHOR*. 2018. URL: http://www.jiangshanyu.com/doc/paper/ANCHOR-proof.zip.

[YY05] FrancesF. Yao and YiqunLisa Yin. "Design and Analysis of Password-Based Key Derivation Functions". English. In: *Topics in Cryptology - CT-RSA 2005*. Ed. by Alfred Menezes. Vol. 3376. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, pp. 245–261. DOI: 10.1007/978-3-540-30574-3_17. URL: http://dx.doi.org/10.1007/978-3-540-30574-3_17.

[ZET15] KIM ZETTER. *Researchers Solve Juniper Backdoor Mystery; Signs Point to NSA*. 2015. URL: https://www.wired.com/2015/12/researchers-solve-the-juniper-mystery-and-they-say-its-partially-the-nsas-fault/.

[ZFC17] Hong Zhong, Yaming Fang, and Jie Cui. “LBBSRT: An efficient SDN load balancing scheme based on server response time”. In: *Future Generation Computer Systems* 68 (2017), pp. 183–190. DOI: https://doi.org/10.1016/j.future.2016.10.001. URL: http://www.sciencedirect.com/science/article/pii/S0167739X16303727.

[Zha+14] Jing Zhang, Zakir Durumeric, Michael Bailey, Mingyan Liu, and Manish Karir. “On the Mismanagement and Maliciousness of Networks”. In: *Symposium on Network and Distributed System Security (NDSS)*. 2014. URL: https://www.ndss-symposium.org/ndss2014/programme/mismanagement-and-maliciousness-networks/.

[ZIR15] Y. Zhao, L. Iannone, and M. Riguidel. “On the performance of SDN controllers: A reality check”. In: *IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*. 2015, pp. 79–85. DOI: 10.1109/NFV-SDN.2015.7387410. URL: https://ieeexplore.ieee.org/document/7387410.

[ZJZ17] Shengzhi Zhang, Xiaoqi Jia, and Weijuan Zhang. “Towards comprehensive protection for OpenFlow controllers”. In: *19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*. IEEE. 2017, pp. 82–87. DOI: 10.1109/APNOMS.2017.8094183. URL: https://ieeexplore.ieee.org/abstract/document/8094183.

[ZSV02] Lidong Zhou, Fred B. Schneider, and Robbert Van Renesse. “COCA: A Secure Distributed Online Certification Authority”. In: *ACM Trans. Comput. Syst.* 20.4 (Nov. 2002), pp. 329–368. DOI: 10.1145/571637.571638. URL: https://doi.org/10.1145/571637.571638.