Formal Analysis of SDNsec: Attacks and Corrections for Payload, Route Integrity and Accountability

Ayoub Ben Hassen
ayoub.benhassen@supcom.tn
École Supérieure des Communications de Tunis
Tunis, Tunisia

Pascal Lafourcade
pascal.lafourcade@uca.fr
Université Clermont Auvergne, CNRS, Clermont
Auvergne INP, Mines Saint-Etienne, LIMOS
63000 Clermont-Ferrand, France

Dhekra Mahmoud
dhekra.mahmoud@uca.fr
Université Clermont Auvergne, CNRS, Clermont
Auvergne INP, Mines Saint-Etienne, LIMOS
63000 Clermont-Ferrand, France

Maxime Puys
maxime.puys@uca.fr
Université Clermont Auvergne, CNRS, Clermont
Auvergne INP, Mines Saint-Etienne, LIMOS
63000 Clermont-Ferrand, France

Abstract

Modern networks increasingly rely on Software-Defined Networking (SDN) to achieve flexibility and efficiency, which makes it more critical to ensure their security. Particularly, it is crucial to ensure the integrity of packets routing within the Network.

In this paper, we address key security challenges within SDN architectures with a particular focus on the data plane. We explore these challenges through the lens of a security solution called SDNsec. We identify three fundamental security properties, namely payload integrity, route integrity and accountability. We define these security properties in the formal framework of the applied Π-Calculus. Likewise, we suggest two levels of route integrity: strong and weak, depending on the protocol's requirements. We use ProVerif, a cryptographic protocol verifier, to conduct a formal analysis of the security of SDNsec.

Thanks to our models, we discover several flaws on all the aforementioned properties. In response, we propose corrective measures to enhance SDNsec's security. Moreover, to ensure that such attacks are feasible and that our improvements are effective, we implement and test SDNsec and our corrected solution using the RYU controller and Mininet network emulator.

CCS Concepts

• **Security and privacy** → **Formal security models**; **Security protocols**; • **Networks** → **Network protocols**.

Keywords

Security, Protocols, Routing, SDN, Formal Methods, ProVerif.

ASIA CCS '25, Hanoi, Vietnam

ACM ISBN 979-8-4007-1410-8/25/08
https://doi.org/10.1145/3708821.3710828

ACM Reference Format:
Ayoub Ben Hassen, Pascal Lafourcade, Dhekra Mahmoud, and Maxime Puys. 2025. Formal Analysis of SDNsec: Attacks and Corrections for Payload, Route Integrity and Accountability. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '25), August 25–29, 2025, Hanoi, Vietnam.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3708821.3710828

1 Introduction

Software-Defined Networking has emerged as a trans formative paradigm in network management and architecture, decoupling the control plane from the data plane to enable centralized control and programmability. However, alongside the benefits, SDN presents significant security challenges that need to be addressed.

In order to enhance various security aspects within SDN, researchers have proposed a range of solutions targeting different SDN planes. For instance, FLOWGUARD [25] is a robust framework developed to strengthen firewall security in SDN, particularly those utilizing OpenFlow [36]. As network states and configurations undergo frequent modifications, FLOWGUARD employs techniques such as flow path tracking, header space analysis, and dependency-breaking methods to identify and resolve firewall policy violations.

Solutions also include adapting traditional intrusion detection systems (IDS) like Snort to SDN environments (e.g., SnortFlow [47]) and employing Moving Target Defense (MTD) techniques [28], such as randomizing network paths between source and target [26], to make it harder for attackers to exploit specific vulnerabilities. Middleboxes, such as FlowVisor [43], act as intermediaries between the control plane and data plane, adding an extra layer of security. Techniques like multi-path routing are employed to obfuscate communication paths and prevent eavesdropping [19], while traffic isolation through network slicing [41] ensures that sensitive data remains confidential.

As our work focuses on SDN data plane security, we concentrate on security solutions tailored to this layer, with a particular emphasis on SDNsec [40]. The data plane in SDN is vulnerable to attacks from compromised switches, which can redirect traffic for eavesdropping, bypass security measures, or disrupt service by dropping packets. This layer often lacks accountability mechanisms to ensure that network policies are correctly applied, which leads to undetected policy violations. To address these challenges,

SDNsec has been developed as a security extension designed to enhance traceability and verification of packet routing rules within the data plane, adding cryptographic features such as Message Authentication Codes (MAC).

Contributions. We present a formal verification of SDNsec, focusing on three key security properties: payload integrity, route integrity, and accountability. Our work involves a comprehensive approach that includes: (i) the formal definitions of these security properties, (ii) a formal modeling of SDNsec, and (iii) analysis of these properties using ProVerif [6], a cryptographic protocol verifier based on the applied Π-Calculus [1]. More precisely, our contributions include the following:

- We give a clear description of SDNsec protocol. Then we propose a formal generic model in applied Π-Calculus for SDN protocols. For this, we consider a network topology where all switches are connected to one another. This provides the intruder with the possibility of establishing any connection between any entities, modeling the most powerful intruders in order to have the strongest security.
- We define relevant security properties to SDN protocols. We consider: payload integrity ensuring that data remain unaltered during transmission, route integrity ensuring that packets adhere to the designated network path, and accountability ensuring the ability of the controller to blame who would be responsible of a dishonest misbehavior.
- To thoroughly assess these properties, we develop models to formally analyze payload integrity, route integrity and accountability for SDN-based protocols. We formally define route integrity and propose two distinct versions of this security property: Weak Route Integrity, and Strong Route Integrity (Strong route integrity being the combination of two sub-properties: local route integrity and transitional route integrity). We also define accountability mainly following the definitions of Bruni et al [8].
- We perform a formal analysis of SDNsec using ProVerif for our models of payload integrity, route integrity, and accountability properties. We identify attacks and develop corresponding corrective measures.
- Additionally, we implement the enhanced solution using the SDN controller RYU [46], and the network emulator Mininet [24]. This practical implementation allows us to demonstrate that our attacks are possible and to test our corrections in realistic virtual network environments, demonstrating their effectiveness in a controlled setting.

All our ProVerif files and RYU implementation codes are available in our anonymous artifacts [4].

Outline. We start by presenting how SDNsec works in details. We explain how cryptographic primitives and its architecture should enhance security. In Section 4, we propose a generic formal model for SDN protocol in the applied Π-Calculus. In Section 5, we define our three security properties: payload integrity, route integrity, and accountability for SDN protocols. In Section 6, we apply our models for SDN to SDNsec to analyze these three security properties. Thanks to ProVerif, we discover flaws regarding each property. We also propose corrective measures to refine the SDNsec solution based on our findings and re-validates these adjustments using ProVerif to confirm whether the attacks have been addressed. Finally, in Section 7, we detail the implementation using the RYU controller and Mininet. We discuss ethics in Section 8 conclude and discuss future perspectives in Section 9.

2 Related Work

Related work can be grouped into two categories. The various approaches proposed in the literature to address the vulnerabilities in the data plane of SDN architectures is presented first. An overview of the automated verification of cryptographic protocols and their application to SDN protocols follows.

Securing the data plane of SDN architectures. Real-time verification of network state and protection against malicious activities are some of the most common mechanisms that address the vulnerabilities within the data plan of SDN architectures. Numerous related works are introduced by Charfadine [39].

VeriFlow is introduced by Khurshid et al. [29] as a real-time verification tool designed to identify malicious flow rules inserted in OpenFlow switches. It intercepts rule changes from the SDN controller before they are applied to network devices, verifying their impact on the network in real-time. By dividing the network into equivalence classes (ECs) that represent sets of packets with similar forwarding behaviors, VeriFlow focuses on the relevant portions of the network affected by each rule. It constructs forwarding graphs for these ECs to model packet traversal through the network checks these graphs against network invariants.

Avant-Guard is proposed by Shin et al. [45] aiming at safeguarding the connection between the data plane and control plane in SDN architectures from flooding attacks. It achieves this by augmenting OpenFlow switches with an additional hardware component, specifically a TCP proxy, to counteract TCP-based attacks. This proxy works by issuing a SYN-ACK response and then forwarding the SYN packet to confirm the presence of both the source and destination. Only when both endpoints are validated is the connection deemed legitimate.

Yegneswaran et al. [49] develop FortNox, a platform designed to help a NOX controller verify rule inconsistencies in real time. Before OpenFlow applications can modify these rules, FortNox uses digital signatures and security constraints to authorize or deny the changes. This platform, through a software extension on the NOX controller, dynamically analyzes rule conflicts using a dedicated algorithm. Consequently, when an authenticated security application inserts a flow rule, FortNox prevents other applications from adding conflicting rules on the same SDN network.

Black and Scott-Hayward [5] provide a comprehensive survey on the verification of adversarial data planes in software-defined networks. In 2020, Zhang et al. propose FlOw Counter Equation System (FOCES) [51], a method to detect misrouting and malicious switches in SDN by applying a network-wide flow conservation principle. It models the expected forwarding behavior using a Flow Counter Matrix (FCM) that captures the relationship between all flows and rules in the network. FOCES periodically collects flow counters from switches and checks if these counters satisfy the constraints defined by the FCM. Inconsistencies in the counters

Solution	Cryptography	Misrouting Detection	Payload Integrity Verification
VeriFlow [29]	✗	✗	✗
Avant-Guard [45]	✗	✗	✗
FortNox [49]	✗	✗	✗
Sphinx [21]	✗	✗	✗
FlowMon [27]	✗	✗	✗
WedgeTail [42]	✗	✓	✗
FOCES [51]	✗	✓	✗
WhiteRabbit [44]	✗	✓	✗
REV [50]	✓	✓	✗
SDNsec [40]	✓	✓	✗

✓: **Property satisfied** ✗: **Property violated or cryptography not in use**

Table 1: Comparison of SDN security solutions

indicate forwarding anomalies, such as path deviations, switch bypasses, or early packet drops, revealing the presence of misrouting or malicious activity.

Shimizu et al. present WhiteRabbit [44], a scalable SDN Data-Plane verification method through time scheduling. WhiteRabbit ensures the consistency of forwarding states in SDN by performing a byte consistency check by comparing the byte count of packets passing through consecutive switches along the expected path. Any significant discrepancies indicate potential misrouting or malicious activities. WhiteRabbit constructs a flow graph based on controller-issued FLOW_MOD messages, and validates actual packet paths against this graph to identify deviations, ensuring the detection of compromised switches altering routes.

Dhawan et al. introduce SPHINX [21], a framework that uses flow graphs to represent and validate network operations in real-time. This allows for the detection of both known and unknown attacks on network topology and data plane forwarding. SPHINX incrementally verifies network updates using these flow graphs and raises alerts for suspicious changes. It monitors packet statistics and verifies them against expected values using flow conservation principles. By continuously checking for consistency in these statistics across the network, Sphinx identifies discrepancies that indicate malicious activity.

Shaghaghi et al. propose WedgeTail [42], a trajectory-based sampling mechanism to prioritize the inspection of forwarding devices efficiently. It calculates expected packet trajectories using a virtual network replica and compares them with actual trajectories to detect malicious actions such as packet drops, misrouting, and generation. WedgeTail can autonomously identify and localize these malicious devices without relying on predefined rules, making it adaptable to various SDN setups. Evaluations in simulated environments demonstrate its effectiveness in detecting and responding to malicious forwarding devices.

Additionally, Kamisinski et al. present FlowMon [27], two anomaly detection algorithms identifying compromised switches based on their behaviors. The first detects packet droppers by comparing the number of packets received and transmitted on each switch port. The second identifies packet swappers by analyzing the discrepancies between expected and actual forwarding paths. FlowMon effectively detects these malicious behaviors in real-time by leveraging the OpenFlow protocol for collecting switch statistics. Simulation results demonstrate that FlowMon can accurately detect both packet droppers and swappers with minimal false positives.

Furthermore, Zhang et al. introduce Rule Enforcement Verification (REV) [50] to ensure correct rule enforcement in SDN. Unlike previous methods such as SDNsec, which rely on Message Authentication Codes (MAC) and result in high communication overhead, REV utilizes compressive MACs to significantly reduce switch-to-controller traffic by 97% and increase verification throughput by 8 times. Additionally, it detects misrouting by tagging packets at their entry point with a cryptographic hash, which is updated by each switch along the expected path using a secret key shared with the controller. The destination switch reports the final tag to the controller, which verifies it against the expected value. If a packet bypasses any intended switches or deviates from the expected path, the final tag does not match the expected tag, indicating a malicious activity.

Finally, SDNsec [40] is proposed by Sasaki et al. It is a security extension designed to ensure traceability of packet routing rules within the data plane of SDN. In other words, it verifies that flow rules are correctly applied at the infrastructure layer. Authors proposed a mechanism that ensures switches forward packets according to the controller's instructions and validates the packet paths. This allows the controller to re-actively verify that the data plane is following the specified rules. This solution guarantees consistent updates to flow rules, ensuring that the data plane behaves as expected during re-configurations.

Table 1 summarizes these information. As far as we know, none of the listed solutions have been analyzed using formal modeling tools. Most analyzes have been conducted using network simulators, emulators, testbeds, or prototype implementations.

Automated verification for cryptographic protocols. After the initial work of Lowe in 1995 [34] based on [22], several tools have been developed during the last decades among [3, 6, 10–13, 17, 23, 37]. We chose ProVerif, one of the most efficient tools [18, 33].

For security definitions, many works have proposed definitions for classical security properties like secrecy, authentication, or privacy [14, 16, 20, 35], but, up to our knowledge, only few one work study routing protocols [2, 9, 15, 48]. In [9], the authors propose a simple formal model for dynamic source routing protocol, however it does not provide any verification with a tool, hence this work remains theoretical and focus on a different security property for routing protocols. In [9], the authors are performing computational simulation based analysis to discover attacks on two secure ad hoc routing protocols, SRP and Ariadne but they do not analyze SDNsec. In contrast to our approach, they do a manual analysis and do not use an automatic verification tool. In [2], they studied wireless communication and proposed a new intruder model to catch the wireless communication in applied Π-Calculus, meaning that the intruder can only attack nodes in his range. We consider wire communications and such attacker is not adapted to the analysis of SDNsec. For accountability definition, after the pioneer work of Küsters et *al.* [32] several works have studied this property [8, 31, 38]. We use the approach proposed by [8] to formalize accountability adapted to SDN protocols.

3 Overview of SDNsec Solutions

SDNsec is a security extension for the SDN data plane proposed in [40]. It enhances network security by ensuring data packets follow authorized paths. It incorporates the use of Message Authentication Codes (MACs) to maintain the integrity of forwarding rules through path enforcement and includes a path validation mechanism for additional security. The SDNsec data plane incorporates both edge and core switches.

Edge switches. These switches are located at the network's periphery. They are equipped with flow tables, which they use to direct traffic either entering or exiting the network. The edge switches interact differently based on their role:

- *Ingress switches* receive packets from source hosts and determine the appropriate forwarding actions. If required information is not found, they contact the controller for instructions.
- *Egress switches* receive packets from core switches and route them to their final destinations.

Core switches. These switches are situated centrally within the network and focus on efficiently forwarding packets according to embedded forwarding information. They perform minimal processing, simply verifying the integrity of each packet's forwarding details and routing them accordingly. If discrepancies are found during verification, the packets are discarded, and the controller is alerted. Additionally, core switches maintain a record of alternative routing paths to be utilized in the event of link failures, ensuring continuity of service and facilitating ongoing flow monitoring. In Figure 1, we illustrate the comprehensive architecture of SDNsec includes a controller, two endpoints, $H1$ and $H2$, an ingress switch S_0, an egress switch S_n, and intermediate core switches S_1 to S_{n-1}. The controller distributes secret symmetric keys, represented as $K_0 \ldots K_n$ to each switch in the network.

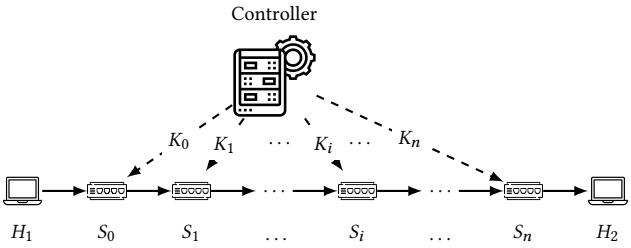

Figure 1: The SDNsec Network Architecture.

Thus, whenever ingress switch S_0 receives a packet from endpoint H_1, it checks its flow table according to packet's header details. If a rule is found containing the forwarding information installed by the controller, S_0 assigns a sequence number, calculates security fields later described, and forwards the packet to a core switch through one of its outbound interface, later referenced as $egr(S_0)$, as specified by the rule. If no rule is found, S_0 requests forwarding information from the controller. The controller computes the required information and sends it to S_0, which then assigns a sequence number, calculates the security fields, and forwards the packet through $egr(S_0)$.

As the packet arrives at S_1, S_1 extracts the security fields from the packet and checks them according to its own secret key K_1. If the MAC is valid, then S_1 increments the sequence number, and forwards the packet to S_2 via $egr(S_1)$ and so on. If the MAC is invalid, then S_1 drops the packet and notifies the controller.

When the packet reaches the egress edge switch S_n, the same process is repeated. If the MAC is valid, then S_2 forwards the packet to its destination H_2 and sends its PVF to the controller who will check it re-actively. If the controller receives the expected PVF, path enforcement is confirmed, otherwise, the controller requests the PVF from previous switches to detect where the path was altered.

3.1 SDNsec Security Fields

SDNsec involves two components adding security fields to messages: *Forwarding Entries* (FE) and *Path Validation Fields* (PVF).

3.1.1 Forwarding Entries. Path computation involves the controller deciding the route that the packet should take and computing the forwarding information for the decided path. This involves calculating the egress interface for each packet along the path, embedding a sequence of these interfaces in the packet, along with an expiration time and flow identifier for each path. As part of SDNsec, cryptographic mechanisms ensure each packet's FE, employing Message Authentication Codes (MACs) calculated with a shared secret key. Each $FE(S_i)$ for a switch S_i includes a MAC covering egress interface, expiration time, and flow identifier, along with forwarding entry of the previous switch, computed as follows:

$$B = FlowID \parallel ExpTime$$
$$FE(S_i) = egr(S_i) \parallel MAC(S_i)$$
$$MAC(S_i) = MAC_{K_i}(egr(S_i) \parallel FE(S_{i-1}) \parallel B)$$

Where:

- *FlowID (Flow Identifier):* is a 3-byte integer, generated by the controller, that uniquely identifies a flow and indexes its information. This allows both the controller and switches to efficiently search for flow information.
- *ExpTime (Expiration Time):* is a four-byte timestamp generated by the controller, indicating when the flow becomes invalid, prompting switches to discard the flow information.
- *FE (Forwarding Entry):* is calculated by the controller and contains one byte for the egress interface through which the switch must forward the packet to the next switch, along with a 7-byte MAC that ensures the integrity of the path.
- *egr(S_i):* represents the egress interface of the switch S_i through which the packet is forwarded to the next hop.

The concatenation symbol $\|$ represents the process of joining two or more sequences of data together end-to-end. Additionally, since the communications between switches and the controller is assumed secure by SDNsec, $FE(S_0)$ is not utilized by the first-hop switch and is computed as: $FE(S_0) = B$. Finally $FE(S_0)$ is then used in the computation of forwarding entry for switch S_1.

3.1.2 Path Validation Field. Path verification is operated re-actively to verify the path taken by packets using a *Path Validation Field* (PVF) embedded in each packet, which contains a MAC computed over the previous switch's validation field and mutable per-packet information, including the flow identifier and a sequence number. This MAC validation allows the controller to confirm the authenticity of the path traveled by the packet and it is computed as follows:

$$C = FlowID \,\|\, SeqNo$$
$$PVF(S_0) = MAC_{K0}(C)$$
$$PVF(S_i) = MAC_{K_i}(PVF(S_{i-1}) \,\|\, C)$$

Where:

- *SeqNo (Sequence Number):* is a 24-bit number maintained by the ingress switch, choosing a unique SeqNo for each flow entry. As it forwards packets, each switch updates the SeqNo. This SeqNo helps randomize the PVF and detect replay attacks, where a malicious switch might reuse valid PVFs to approve a rogue path.
- *Path Validation Field (PVF):* is an 8-byte cryptographic marker used by the controller to validate the path. Each switch calculates it based on equation given above. Upon request, the switch sends the packet header and its PVF to the controller.

As S_0 is the first switch of the path, no previous PVF can be included in the computation of $PVF(S_0)$ leading $PVF(S_0) = MAC_{K0}(C)$.

3.2 SDNsec Packet Headers

The packet header used in SDNsec protocol is depicted in Figure 2. It including several fields for forwarding, path validation, and link-failure recovery. We note that the third byte consists of three fields: the packet type (1 bit), a "do not detour" flag (1 bit), and a link failure counter (6 bits):

- *PktType (Packet Type):* A single bit flag indicating if the packet is multicast/broadcast or unicast.
- *FE Ptr (Forwarding Entry Pointer):* A one-byte pointer to the forwarding entry (FE) that each switch on the path must examine. Each switch updates and encrypts this value accordingly. The forwarding entry pointer ensures that the next-hop switch checks the correct FE.
- *LFC (Link Failure Counter):* A 6-bit counter indicating the number of failed links the packet encounters on its path to the destination.
- *EgressID (Egress Switch ID):* A 2-byte ID for identifying the egress switch when a core switch suffers a link failure and needs to determine an alternate path to the egress switch.

We choose to not address issues related to link failures, and our models do not account for any information related.

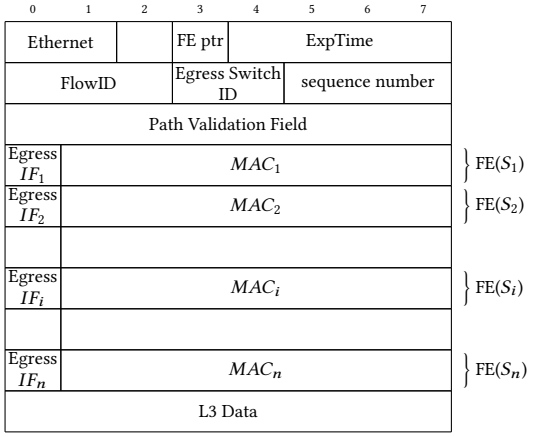

Figure 2: SDNsec Packet Header Layout.

4 Modeling SDN Protocols

We first introduce the applied Π-Calculus [1]. We then give a formal model for SDN protocols in this framework.

4.1 The Applied Π-Calculus

The applied Π-Calculus is an extension of the process calculus the Π-Calculus. It is among the most used languages for modeling security protocols in formal verification of cryptographic protocols. It is also the input language of ProVerif. In the following, we do not go into the full details of the calculus syntax and semantics. Instead, we focus on the grammar of the calculus that is relevant for modeling our security properties [1].

In the applied Π-Calculus, systems are described as processes. Given two processes P and Q, $P \mid Q$ denotes the parallel composition of P and Q. Given a name n, for example a secret key sk, the process $vn \cdot P$ makes a new private name then runs P. Processes can also be extended with active substitutions. We write $P\sigma_{sk}$ when the free variable x of P is substituted by sk. An extended process is called *close*, when all its free variables are replaced by active substitutions.

To reason about the security properties, a syntactical extension of the applied Π-Calculus, used by many tools, consists of annotating processes using events. Therefore, the reachability of the events and/or the correspondence between them are often used to formalize trace based security properties.

4.2 Model of SDN Protocols

An SDN protocol involves four main parties: a controller C, ingress switches S_i, egress switches S_e and core switches S_c. In the following, we give a formal specification of SDN protocols in the applied Π-Calculus. However, we should first make some assumptions about the topology of the network we are considering. Since, we consider the Dolev-Yao's attacker (that controls the entire network), we make the assumption that all entities can communicate with one another. This enables the attacker to establish any connection between any node in the network. The topology of the network model is thus a complete graph between all entities. In addition to that, in our model, we let the users choose the length of the route of their choice. We first give a general definition in applied Π-Calculus for any SDN protocol with one controller and some switches.

Definition 4.1 (SDN Protocol). An *SDN Protocol* is a tuple $(C, S_i, S_e, S_c, \tilde{n}_p)$ where C is the process executed by the controller, S_i is the process executed by the ingress switches, S_e is the process executed by the egress switches, S_c is a process executed by the core switches and $\tilde{n}_p$ is a set of private channel names to allow communication between these entities.

We note that different processes can be run by the same entity, depending on the protocol and the roles played by the switches within it. Moreover, we do not impose any restrictions on the set of private channel names, which may be empty when the SDN protocol does not require private channels.

Definition 4.2 (SDN instance). An *SDN instance* is a closed process $SP = \nu\tilde{n}\cdot(S_i\sigma_{id_{i_1}}\sigma_{skS_{i_1}} | \ldots | S_i\sigma_{id_{i_j}}\sigma_{skS_{i_j}} | S_e\sigma_{id_{e_1}}\sigma_{skS_{e_1}} | \ldots | S_e\sigma_{id_{e_k}}\sigma_{skS_{e_k}} | S_c\sigma_{id_{c_1}}\sigma_{skS_{c_1}} | \ldots | S_c\sigma_{id_{c_l}}\sigma_{skS_{c_l}} | C)$ where $\tilde{n}$ is the set of all restricted names, which includes the set of the protocol's private channels; $S_i\sigma_{id_{i_m}}\sigma_{skS_{i_m}}$ are the processes run by the ingress switches with the substitutions specifying the identity and the private name of the m^{th} switch; $S_e\sigma_{id_{e_p}}\sigma_{skS_{e_p}}$ are the processes run by the egress switches with the substitutions specifying the identity and the private name of the p^{th} switch; $S_c\sigma_{id_{c_n}}\sigma_{skS_{c_n}}$ are the processes run by core switches with the substitutions specifying the identity and the private name of the n^{th} switch and C is the process executed by the controller.

Some SDN protocols do not involve cryptographic primitives at all, so the switches do not involve private names. In that case, we can simply write $S\sigma_{id}$ instead of $S\sigma_{id}\sigma_{sk}$ *i.e.* the latter substitution does not alter the semantics. Once again, it should be noted that the preceding definitions are sufficiently broad to be instantiated with any SDN protocol up to the processes' specifications.

5 Modeling Security Properties

We model the payload integrity, Route integrity and accountability for any SDN protocols using *events* and first-order logic formulas. Events, whose parameters refer to the information contained in the exchanged messages, indicate steps followed during the execution of the protocol. They are simply annotations that do not alter processes' behaviors yet are inserted at specific locations to allow reasoning about the protocol's execution. The idea is to raise events at the right place in the processes and thanks to these events, it is possible to model traced based security properties. We describe how to place such events in a SDN protocol and give the formal definition of our three security properties based on them. In the next section, we apply our modelings to SDNsec.

5.1 Payload Integrity

This property ensures that data remain unaltered during transmission. Attacks on this property would thus imply any core-switches of the route modifying the payload of the packet during transport. We model payload integrity with a trace property. We define two events where p is the payload of the packet. Note that in our work, we assume that the payload is immutable and we consider it to be the user data, and we aim to secure its integrity. We define two events to be thrown in the protocol model:

- $begin(p)$: is the event inserted when the payload p begins its journey through the network. Typically, this event would be triggered at the source node, where the data packet is first received.
- $end(p)$: is the event added at the final node and is triggered when the packet leaves this last node.

The payload integrity property ensures that the payload has successfully completed its journey without being altered, ensuring that its integrity is preserved.

Definition 5.1 (Payload Integrity). An SDN protocol ensures Payload Integrity, if for every SDN process, each occurrence of the event *end(p)* is preceded by an occurrence of the event *begin(p)*.

$$\text{event}(end(p)) \Rightarrow \text{event}(begin(p))$$

5.2 Route integrity

This property ensures that packet will properly follow the route designated by the controller. Attacks on this property would thus imply re-routing the packet to core-switches that were not part of the designated route[1]. Let us assume, without loss of generality, the following route to be chosen by the controller: $R = (S_0, S_1, \ldots, S_n)$ such that S_0 is an ingress switch, S_n is an egress switch and $S_{j,1\leq j\leq n-1}$ are core switches. Let event($beginS_k$) be the event inserted into the process of the switch S_k at the location where it forwards the packet. Likewise, let event($endS_k$) be the event inserted into the process of the switch S_{k+1} when the latter receives and accepts the incoming packet. We define four properties: (i) Local route integrity, (ii) Transitional route integrity, (iii) Weak route integrity, and (iv) Strong route integrity.

The first property ensures a local integrity of the route. It means that one switch locally starts and finishes correctly its execution.

Definition 5.2 (Local Route Integrity). An SDN protocol ensures *Local Route Integrity* with regard to the switch S_k belonging to the route $R\backslash\{S_0\}$, if for every SDN process, each occurrence of the event $beginS_k$ is preceded by an occurrence of the event $endS_{k-1}$.

$$\text{event}(beginS_k) \Rightarrow \text{event}(endS_{k-1})$$

[1]An attack example could for instance be a payload that should not transit through switches of some country for national security reasons and that could be re-routed to switches from this country as part of the attack.

The second property ensures the integrity of transitions between local routes. We call it *transitional route integrity*, since it models the fact that the switches sent the packets according to the route.

Definition 5.3 (Transitional Route Integrity). An SDN protocol ensures *Transitional Route Integrity* with regard to the switches S_k and S_{k+1} belonging to the route R, if for every SDN process, each occurrence of the event $endS_k$ is preceded by an occurrence of the event $beginS_k$.

$$\text{event}(\text{endS}_k) \Rightarrow \text{event}(beginS_k)$$

The third property ensures that whenever a route ends, all switches within the route have at least participated in the protocol.

Definition 5.4 (Weak Route Integrity). An SDN protocol ensures *Weak Route Integrity* with regard to the route R, if for every SDN process, each occurrence of the event $endS_n$ is preceded by the occurrence of all events $beginS_k$ for all $k \in \{0 \dots n\}$.

$$\text{event}(endS_n) \Rightarrow \bigwedge_{k=0}^{n} \text{event}(beginS_k)$$

The last property is the strongest one. It ensures that local route integrity and transitional route integrity are satisfied.

Definition 5.5 (Strong Route Integrity). An SDN protocol ensures *Strong Route Integrity* with regard to the route R, if for every SDN process we have:

$$\bigwedge_{k=0}^{n} (\text{event}(endS_k) \Rightarrow \text{event}(beginS_k)) \wedge$$
$$\bigwedge_{k=0}^{n-1} (\text{event}(beginS_{k+1}) \Rightarrow \text{event}(endS_k))$$

In the following theorem, we demonstrate that strong route integrity implies weak route integrity. The corresponding proof is given in Appendix A.

Theorem 5.6. *Let R be a route, then if a protocol ensures* Strong Route Integrity *it implies that it ensures* Weak Route Integrity.

Such result is very important. First, it allows us to only verify Strong Route Integrity property for secure protocols. For insecure protocols, if we find an attack against Weak Route Integrity, we do not have to test the other property. Note that Strong Route Integrity is a more complex property, then it might be more difficult to verify by the tools. It is therefore interesting to provide a weaker version for early attack detection. Second, one may find Strong Route Integrity to be a very strong property, especially for protocols that do not involve cryptographic mechanisms. The weak version may appear to be sufficient in this case. Since we wanted to propose general definitions without sticking to one particular SDNprotocol, we proposed both definitions.

5.3 Accountability

Developing security protocols requires trusting some participants. Trustworthiness can be bolstered by holding participants accountable for their actions, as this drives them to steer clear malicious actions. We formally define *accountability* for SDN protocols following mainly the exposition of Bruni et al [8]. We suppose that the switches can misbehave and thereof can be indicted and that the controller is the only trusted party participating in the protocol. In order to establish accountability for misbehavior, we must first define what misbehavior means.

Definition 5.7 (honest S_i). A switch S_i is considered to be *honest* if it is a legitimate participant in the protocol which will not deviate from the defined protocol.

We therefore define misbehavior as any deviation from the protocol. If the switch does not depart from the protocol specifications, then it is considered honest. It is noteworthy that the latter definition does not exclude a collusion between honest switches and the attacker, as switches still have the ability to transmit confidential information to the intruder. We believe that the latter scenario alone is not pertinent to our definition of accountability, provided that it does not result in a deviation from the protocol. Let $\mathcal{T}$ be a target or a goal which should be verified within the protocol. Let $V_{\mathcal{T}}$ and $A_{\mathcal{T}}$ be two processes with only two possible outcomes 0 or 1 and let event(V_{ok}) and event(V_{ko}) be the events inserted into the process $V_{\mathcal{T}}$ at the location where it outputs 1 and 0 respectively. Similarly, we place the events event(A_{ok}) and event(A_{ko}) into the process $A_{\mathcal{T}}$. In order to account for a misbehaving switch in relation to the target $\mathcal{T}$, it is important that the protocol provides means to verify whether the target has been reached or not. Otherwise, we cannot hold a party accountable when we are unable to verify at least the attainability of the goal.

Definition 5.8 (Verifiability). An SDN protocol is verifiable with regard to $\mathcal{T}$ if for every SDN process we have:

(1) $\text{event}(\text{V}_{ok}) \Rightarrow \mathcal{T}$
(2) $\mathcal{T} \Rightarrow \text{event}(\text{V}_{ok})$

The first condition guarantees that the process $V_{\mathcal{T}}$ returns 1 only if $\mathcal{T}$ holds when running the protocol. The second condition implies that the process cannot return 0 if the switches do not deviate from the protocol in a way that falsifies $\mathcal{T}$.

Definition 5.9 (Accountability). An SDN protocol guarantees accountability of the switch S_i, with regard to $\mathcal{T}$, if for every SDN process we have:

(1) The SDN protocol is verifiable with regard to $\mathcal{T}$
(2) (soundness) $\text{event}(\text{A}_{ko}) \Rightarrow S_i$ is honest
(3) (completeness) S_i is honest $\Rightarrow \text{event}(\text{A}_{ko})$

Soundness guarantees that the process $A_{\mathcal{T}}$ returns 0 only if the indicted switch is honest. Completeness states that the $A_{\mathcal{T}}$ cannot return 1 if the indicted switch is honest.

6 Application to SDNsec

For sake of simplicity, we present our model for the simplest topology to find an attack, it means with only three switches. To be general enough, we provide a Python script that given a length n of a route generates the corresponding models. As explain in Section 5, we focus on three relevant aspects of the SDNsec protocol (payload integrity, route integrity, and accountability). We implement all properties on two versions of SDNsec: (i) SDNsec modeled directly from [40] and (ii) a corrected version of SDNsec where we propose fixing attacks found on the original version of

	Payload Integrity	Route Integrity				Accountability	
		Local RI	Trans. RI	Weak RI	Strong RI	Soundness	Completeness
SDNsec [40]	UNSAFE	SAFE	UNSAFE	UNSAFE	UNSAFE	SAFE	UNSAFE
SDNsec⋆	SAFE	SAFE	SAFE	SAFE	SAFE	SAFE	SAFE

Table 2: Results of the analysis of the SDNsec protocol and the proposed corrected version SDNsec⋆ with ProVerif.

the protocol. We model both channels for both data exchange between switches alongside control frames between switches and the controller. As it was stated in [40] that communications between switches and the controller are considered secure[2], we model these channels as private channels in ProVerif (referred as `c_s0` and `c_s2` in the models), oppositely to the public channel `c` used between switches and accessible to the attacker. The attacker is unable to access private channel yet has complete control on the public one (can inject, intercept, block and send messages).

Table 2 summarizes results found by ProVerif regarding payload integrity, route integrity and accountability properties for SDNsec. In Section 6.2, we explain results of SDNsec regarding payload integrity. In Section 6.1, we present our ProVerif models. In Section 6.3, we detail results for all flavors of route integrity. In Section 6.4, we explain the results found for accountability. Finally, in Section 6.5, we showcase our proposal correction for SDNsec and display results for the properties. All ProVerif models can be found in artifacts [4].

6.1 Automated Verification with ProVerif

In the previous section, we gave formal definitions of SDN protocols in the applied *Pi*-Calculus and security properties using first-order logic formulas over events. Most of the existing symbolic tools supports the specification of trace properties (reachability and correspondence of events) via first-order logic such as ProVerif [6], SAPIC$^+$ [11], Verifpal [30] and Tamarin [37]. Thereof, our security properties can be easily encoded within several automated verifiers. As we used ProVerif in this work, we provide an overview of this tool and how we use it to model processes and our security properties.

```
let ProcessS0(sk0: skey) =
  in(c, n_s0: nat);
  new p: bitstring;
  let p_encrypted = senc(p, sk0) in
  out(c_s0, (p_encrypted, n_s0));
  in(c_s0, y: bitstring);
  let(F: bitstring, t: bitstring, [...]) =
  sdec(y, sk0) in    [...]
  event begins0(egr0);
  out(c, [...]
```

Figure 3: Process of Ingress switch in ProVerif.

In symbolic models, messages are represented as terms which can be either atomic (fresh values such as keys or random coins) or constructed by applying function symbols. For instance the term $MAC(sk, m)$ represents a message authentication code computed with a secret key sk and a message m. A simplified extract of the process modeling the ingress switch role is shown in Figure 3. When defining a process in ProVerif, we begin by specifying its private names. The private name for the ingress switch corresponds to its secret key $sk0$. Then, the ingress switch receives a sequence number generated by the attacker. We made the latter choice to give the attacker more power over the messages. Nonetheless, the only condition that we apply to the sequence number is uniqueness. We do this by means of restrictions provided by ProVerif. The switch creates a new payload **new** `p`, encrypts it using its secret key and sends the ciphertext `p_encrypted` along with the sequence number to the controller through the private channel `c_s0`. The event **event begins0**`(egr0)` is positioned, as defined in Section 5.2, in the beginning of the route (when the ingress switch sends the packet to the following switch).

Additionally, we need to explain how the different roles are run in parallel. The main process modeled in ProVerif in depicted is Figure 4. We model the role of the controller alongside three roles for switches: ProcessS0 (ingress switch), ProcessS1 (core switch), and ProcessS2 (egress switch). We define a *session* of the protocol, a set of these four processes. All sessions are instantiated with a secret symmetric key per switch, shared with the controller of the session. The symbol **!** in front of the processes instantiates unbounded number of sessions.

```
process
  !(new sk0: skey;new sk1: skey;
   new sk2: skey;
   (! Controller(sk0,sk1,sk2)) | (! ProcessS0(sk0))
   (! ProcessS1(sk1)) | (! ProcessS2(sk2)))
```

Figure 4: Main process for 3 switches in ProVerif.

Properties on traces can be expressed by means of correspondence or reachability queries. Reachability queries over an event means that there exists an execution or a trace of the protocol where this event is reached. For example, to check the property of *Local Route Integrity* with regard to the switch s_1 the event we write the query depicted in Figure 5 in the ProVerif file.

```
query egr0: bitstring; event(ends0) ⇒ event(begins0(egr0)
```

Figure 5: Local Route Integrity for Switch s_1.

Either the tool confirms the security property by displaying **true**, or it finds an attack trace that violates the property and outputs

[2]Section II-B: "*The communication channel between the controller and benign switches is secure (e.g., TLS can be used, as in OpenFlow).*"

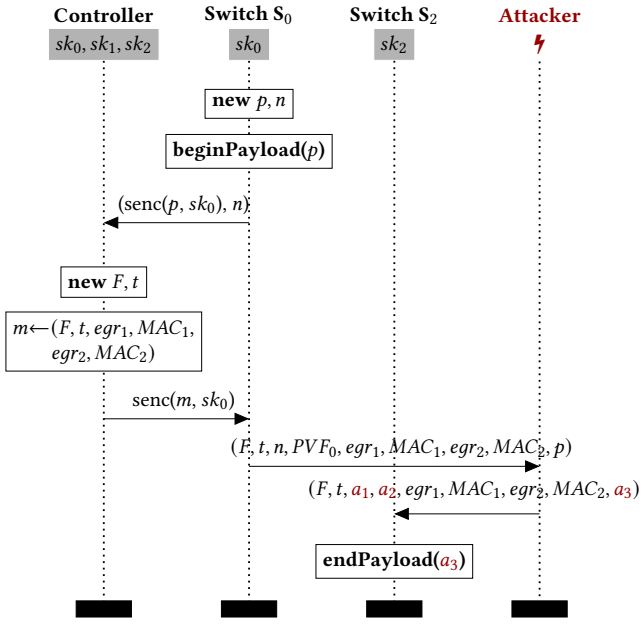

Figure 6: Attack against Payload Integrity.

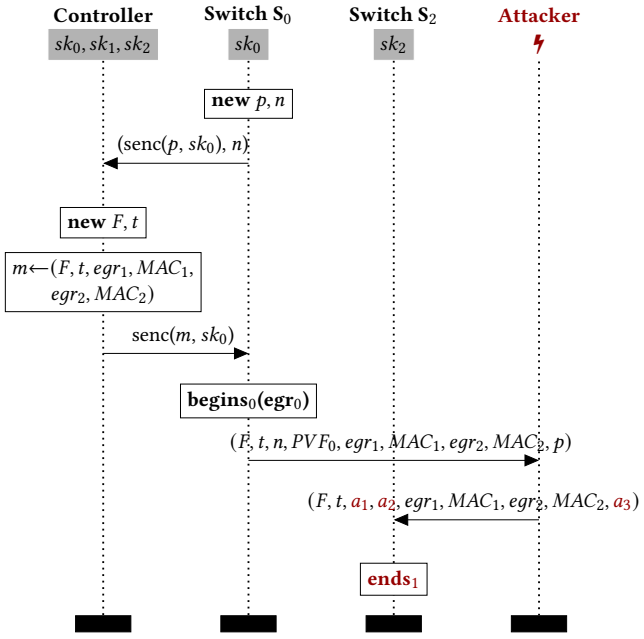

Figure 7: Attack against Weak Route Integrity.

false, or displays *cannot be proved* which means that the tool was unable to conclude[3]. The full processes model in case of three switches are given in Appendix B.

6.2 Verification of Payload Integrity

ProVerif finds an attack which is displayed in Figure 6. As in [40], we do not model hosts and consider ingress and egress switches as source and destination[4]. Thus the trace starts with S_0 creating the payload p and sending it to the controller alongside the sequence number n. The controller computes all FE information and sends them to S_0. The message that S_0 would send to S_1 is intercepted by the attacker who later sends a message directly to S_2, completely bypassing S_1. In this message, the attacker ignores the sequence number which they replace by a value $a1$ chosen by the attacker and the PVF1 which they replace by another value $a2$ chosen by the attacker. More importantly, they also replace the payload p by another payload called $a3$. S_2 triggers the event endPayload(a3), for which no event beginPayload(a3) exists.

Admittedly, no PVF verification has been performed which can seem odd. This is explained since [40] mention that PVF verification is only performed *re-actively* and is not required for S_2 to send the payload to its destination. However, we can confirm that the controller will detect a PVF mismatch (this can easily be modeled if we move the event endPayload from S_2 to C). To circumvent this attack against payload integrity, a hash of the payload needs to be added to *Forwarding Entry* as later explained in Section 6.5.

[3]It is noteworthy to mention that, although we did not encounter those cases in this work, ProVerif is actually sound but not complete [7], which means that the tool may find false attacks. In this case, the attack trace should be manually inspected. ProVerif also may not terminate

[4]Section V-B: "However, an important difference is that we consider the ingress and egress switch – not the hosts – as the source and destination, respectively."

6.3 Verification of Route Integrity

ProVerif finds an attack on transitional route integrity which is displayed in Figure 7. As strong route integrity requires transitional route integrity, strong route integrity is invalid as well. We can remark that weak route integrity is however guaranteed, illustrating that this property is much more loose.

The trace starts with S_0 creating the payload p and sending it to the controller alongside the sequence number n. The controller computes all FE information and sends them to S_0. The message that S_0 would send to S_1 is intercepted by the attacker who later sends a message directly to S_2, completely bypassing S_1. In this message, the attacker ignores the sequence number which they replace by a value $a1$ chosen by the attacker and the PVF1 which they replace by another value $a2$ chosen by the attacker. More importantly, they also replace the payload p by another payload called $a3$. S_2 triggers the event endS1(egr1, egr2), for which no event beginS1(egr1, egr2) exists. Indeed, S_1 have been completely skipped by the attacker. To the best of our understanding, this kind of attack is referenced as path shortcut in [40][5]. Our model confirms that SDNsec does not prevents path shortcut attacks. To circumvent this attack against route integrity, a hash of the previous PVF needs to be added to *Forwarding Entry* as later explained in Section 6.5.

6.4 Verification of Accountability

ProVerif finds an attack on the completeness property. According to Definition 5.9, it means that if all switches are honest, the verification of the final PVF should never fail. Otherwise, the controller would be likely to accuse an honest participant. In SDNsec, it is possible for the computation of the final PVF to fail even though all

[5]Section II-A: Path shortcut: A switch redirects a packet and skips other switches on the path; the packet is forwarded only by a subset of the intended switches.

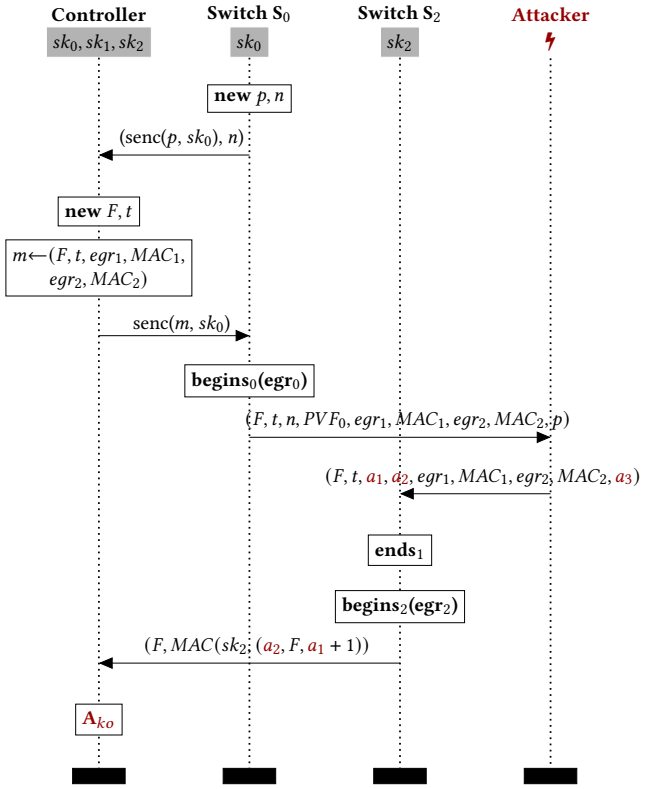

Figure 8: Attack against Completeness of Accountability.

participants are honest (due to attacks found on payload integrity and route integrity).

The trace starts with S_0 creating the payload p and sending it to the controller alongside the sequence number n. The controller computes all FE information and sends them to S_0. The message that S_0 would send to S_1 is intercepted by the attacker who later sends a message directly to S_2, completely bypassing S_1. In this message, the attacker ignores the sequence number which they replace by a value $a1$ chosen by the attacker and the PVF1 which they replace by another value $a2$ chosen by the attacker. They also replace the payload p by another payload called $a3$. As accountability aims at verifying if the controller can re-actively discriminate which switch is responsible of an invalid PVF, the trace continues with S_2 sending its erroneous PVF2 to the controller, where $PVF1$ has been replaced by $a2$, and SeqNo by $a1$, both breaking the chaining of PVF values. The computation of the final PVF requires both the honest value of the sequence number generated by the ingress switch, and the correct chain of PVF values. ProVerif shows that an attacker is able to alter both terms of the packets when intercepting them without being detected by honest switches. Thus, the final PVF computed by the egress switch would not correspond to the final PVF computed by the controller. To circumvent this attack against route integrity, a hash of the previous PVF and previous sequence number need to be added to *Forwarding Entry* as later explained in Section 6.5.

Accountability soundness, on the other hand, checks if dishonest switches (who's keys have been leaked to the attacker) can indeed forge invalid PVF, which explains why soundness always shows an attack in Table 2.

6.5 Proposed Correction

As we showed in Table 2, the version of SDNsec we modeled from [40] is vulnerable to attacks against payload integrity, transitional route integrity, strong route integrity and accountability. In this section, we propose simple corrective measures to circumvent these attacks. They involve simple modifications to the protocol from [40] in the computation of *Forwarding Entries*. The modified computation of *Forwarding Entry* equation is as follows:

$$B = FlowID \parallel ExpTime$$
$$FE(S_i) = egr(S_i) \parallel MAC(S_i)$$
$$MAC(S_i) = MAC_{K_i}(egr(S_i) \parallel FE(S_{i-1}) \parallel$$
$$B \parallel \mathbf{H(p \parallel PVF(S_{i-1}) \parallel SeqNo_{i-1}))}$$

As mentioned in attack descriptions, we proposed a correction for each attack involving one or two terms to add to *Forwarding Entries*. The following modifications in the computation of FE merges each of the proposed fixes:

- $H(p)$ added for payload integrity;
- $H(PVF_{i-1})$ added for transitional route integrity (and thus strong route integrity);
- $H(PVF_{i-1} \parallel SeqNo_{i-1})$ added for accountability completeness.

We applied the latter modifications to the SDNsec protocol and re-analyzed it again with ProVerif. The results of the analysis of the modified version of the protocol, which we refer to as SDNsec★, are depicted in Table 2. ProVerif confirms that all the tested properties are verified. We also analyzed SDNsec★ while varying the number of switches within the protocol. The results of the analysis are depicted in Figure 9. Once again, the security properties are verified, taking into account up to 20 switches within the protocol architecture.

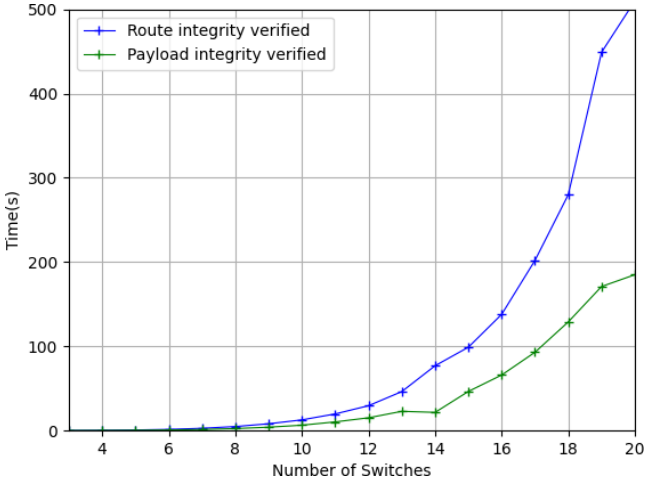

Figure 9: Execution time of ProVerif files depending on the number of switches

7 Implementation with RYU SDN Controller

To confirm the feasibility of our attacks and correction, we carried out an implementation of SDNsec with and without our correction. This implementation was realized in Python relying on the RYU [46] SDN controller [24] and the Mininet network emulator. Tests are run on an Ubuntu 24.04 machine with Linux kernel 5.4.0-189-generic, hosted on a system with an Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz (featuring SSE4.2), and equipped with 15,861 MB of physical memory. The specific versions of the software used in our setup include RYU controller 4.43, Python 3.8.5, Mininet 2.3.0, Wireshark 3.2.3, and ProVerif 2.05.

We created a network topology representing Figure 1 using Mininet. The topology consists of two hosts, H_1 and H_2, and four switches, S_0, S_1, S_2, and S_{12}, with S_0, S_1, and S_2 being honest switches and S_{12} being a malicious switch, representing the role of the attacker. The controller uses secret keys to calculate and verify Message Authentication Codes (MACs) for packets as required by the protocol. We implemented the SDNsec solution as a custom RYU application, which manages the packet forwarding and verification using Hashed Message Authentication Code (HMAC). The application was designed to handle packets with a specific custom header.

Our goal was to replicate the attacks found in Figures 6, 7, and 8. All three attacks involve the attacker intercepting the message sent by S_0 and sending it directly to S_2, bypassing S_1. Obviously, as RYU is a SDN controller simulator, it was not engineered to include attackers. We thus chose to replicate the attack by forcing the traffic between S_0 and S_2 through a malicious switch called S_12 (ignoring S_1), while computing all values of SDNsec (FE, PVF) with the route $S_0 \rightarrow S_1 \rightarrow S_2$. Thus, SDNsec will impose the route $S_0 \rightarrow S_1 \rightarrow S_2$ while RYU will force a different route. The goal is to check if the implementation of SDNsec will detect the path shortcut attack that was found by ProVerif. Obviously, we also implemented the correct route in RYU making sure that the protocol executes correctly.

In case of normal topology (without path shortcut) with SDNsec from [40], RYU correctly implements the protocol, verifying all cryptographic values. When switching to the topology short-cutting S_1, RYU is unable to detect the PVF modification as found by ProVerif, confirming the attack. When implementing our proposed correction as in Section 6.5, RYU is now able to detect the attack. Finally, if we switch back the topology to the correct one, while keeping our correction, RYU is able to complete the protocol correctly, confirming our results. As these tests were done on a Python simulator using optimized cryptographic libraries, impacts of cryptographic operations appeared negligible[6]. All RYU implementation code are available in artifacts [4].

8 Ethics and Responsible Disclosure

While the attacks found by ProVerif in Section 6 completely violate the properties that were supposed to be guaranteed by [40], it is important to note that their work is only a proposal resulting from an academic work. To the best of our knowledge, there is currently no off-the-shelf product implementing the original protocol from [40]. Thus, no real responsible disclosure procedure such as creating a CVE seemed important after our findings. We however notified the authors from [40] to warn them from about the result of our analysis. The absence of a real product also explains why the implementation we proposed Section 7 is done using a simulated environment. We hope that whenever an industrial version of a secure SDN protocol is proposed by vendors, they will take our results into consideration.

[6]More realistic experiments will be carried-on as part of future work.

9 Conclusion and Future Perspectives

We define a formal framework for the analysis of secure SDN protocols. We show how to model SDN protocols in the applied Π-Calculus and define three relevant security properties namely payload integrity, route integrity and accountability. Our primary goal is to ensure the integrity of the payload within the Network while also reinforcing route integrity. Therefore, we distinguish two types of route integrity, namely weak and strong, as the latter may appear to be exceedingly restrictive depending on the requirements of the protocol.

Using ProVerif, we rigorously analyzed the security of SDNsec. Our analysis shows that it satisfies none of the three security properties and it highlights the importance of the security aspects in SDN's data plane, especially concerning misrouting and unauthorized modifications of packet forwarding rules. In SDNsec, switches send the path validation function (PVF) only upon the controller's request. We directed our research towards securing the protocol even in scenarios where the controller does not explicitly request PVF values. We proposed to enhance the cryptographic mechanisms by incorporating a hash of the payload, the sequence number and the previous switch's PVF value. This approach ensures that path enforcement and validation are preserved under all conditions, including cases where no requests are initiated by the controller.

We also implement the SDNsec solution using RYU and Mininet to replicate the attacks. Based on our enhancements, our tests successfully detected attacks related to both payload integrity and route integrity, even in the absence of PVF verification. We have yet to explore the impact of our cryptographic enhancements on network performance metrics, such as bandwidth, latency, or resource consumption, although we believe that they do not affect the performances. Expanding our tests to more complex architectures will allow us to evaluate the broader applicability of our solution across various SDNsec deployments. These challenges remain crucial for ensuring that our approach is resilient and effective in a wider range of scenarios.

Another challenging continuation of our work is to try to adapt to SDN security analysis the result given in [15], where the authors reduce the number of topology to perform formal analysis in routing protocol. Therefore, we will no longer need to vary the number of switches in the protocol in order to check its security.

Acknowledgments

Authors thank the anonymous reviewers, whose comments have helped us to improve the presentation of the paper. This work was partially supported by ANR project SEVERITAS (ANR-20-CE39-0009), and ANR Project PRIVA-SIQ (ANR-23-CE39-0008).

References

[1] Martín Abadi, Bruno Blanchet, and Cédric Fournet. 2017. The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication. *J. ACM* 65, 1, Article 1 (oct 2017), 41 pages. doi:10.1145/3127586

[2] Mathilde Arnaud, Véronique Cortier, and Stéphanie Delaune. 2014. Modeling and verifying ad hoc routing protocols. *Inf. Comput.* 238 (2014), 30–67. doi:10.1016/J.IC.2014.07.004

[3] Manuel Barbosa, Gilles Barthe, Karthik Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. 2021. SoK: Computer-Aided Cryptography. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 777–795. doi:10.1109/SP40001.2021.00008

[4] Ayoub Ben Hassen, Pascal Lafourcade, Dhekra Mahmoud, and Maxime Puys. 2025. *Experiment artifacts for ASIACCS submission #175.* doi:10.5281/zenodo.14629053

[5] Conor Black and Sandra Scott-Hayward. 2021. A Survey on the Verification of Adversarial Data Planes in Software-Defined Networks. *Proceedings of the ACM International Workshop on Software Defined Networks & Network Function Virtualization Security, SDN-NFV Sec 2021.* doi:10.1145/3445968.3452092

[6] Bruno Blanchet. 2001. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proceedings. 14th IEEE Computer Security Foundations Workshop, 2001. (CSFW '01)*. IEEE, IEEE Computer Society, 82–96. doi:10.1109/CSFW.2001.930138

[7] Bruno Blanchet. 2012. Security Protocol Verification: Symbolic and Computational Models. In *Principles of Security and Trust*, Pierpaolo Degano and Joshua D. Guttman (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 3–29.

[8] Alessandro Bruni, Rosario Giustolisi, and Carsten Schuermann. 2017. Automated Analysis of Accountability. 417–434. doi:10.1007/978-3-319-69659-1_23

[9] Levente Buttyán and István Vajda. 2004. Towards provable security for ad hoc routing protocols. In *Proceedings of the 2nd ACM Workshop on Security of Ad Hoc and Sensor Networks* (Washington DC, USA) *(SASN '04)*. Association for Computing Machinery, New York, NY, USA, 94–105. doi:10.1145/1029102.1029119

[10] Vincent Cheval. 2014. APTE: An Algorithm for Proving Trace Equivalence. In *Tools and Algorithms for the Construction and Analysis of Systems*, Erika Ábrahám and Klaus Havelund (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 587–592.

[11] Vincent Cheval, Charlie Jacomme, Steve Kremer, and Robert Künnemann. 2022. SAPIC+: protocol verifiers of the world, unite!. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA. https://www.usenix.org/conference/usenixsecurity22/presentation/cheval

[12] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. 2018. DEEPSEC: Deciding Equivalence Properties in Security Protocols Theory and Practice. In *2018 IEEE Symposium on Security and Privacy (SP)*. 529–546. doi:10.1109/SP.2018.00033

[13] Véronique Cortier, Antoine Dallon, and Stéphanie Delaune. 2017. SAT-Equiv: An Efficient Tool for Equivalence Properties. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. 481–494. doi:10.1109/CSF.2017.15

[14] Véronique Cortier, Alexandre Debant, and Florian Moser. 2024. Code Voting: When Simplicity Meets Security. In *Computer Security - ESORICS 2024 - 29th European Symposium on Research in Computer Security, Bydgoszcz, Poland, September 16-20, 2024, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 14983)*, Joaquin García-Alfaro, Rafal Kozik, Michal Choras, and Sokratis K. Katsikas (Eds.). Springer, 410–429. doi:10.1007/978-3-031-70890-9_21

[15] Véronique Cortier, Jan Degrieck, and Stéphanie Delaune. 2012. Analysing Routing Protocols: Four Nodes Topologies Are Sufficient. In *Principles of Security and Trust - First International Conference, POST 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012, Proceedings (Lecture Notes in Computer Science, Vol. 7215)*, Pierpaolo Degano and Joshua D. Guttman (Eds.). Springer, 30–50. doi:10.1007/978-3-642-28641-4_3

[16] Véronique Cortier and Joseph Lallemand. 2018. Voting: You Can't Have Privacy without Individual Verifiability. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM, 53–66. doi:10.1145/3243734.3243762

[17] Cas J. F. Cremers. 2008. The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols. In *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, NJ, USA, July 7-14, 2008, Proceedings (Lecture Notes in Computer Science, Vol. 5123)*, Aarti Gupta and Sharad Malik (Eds.). Springer, 414–418. doi:10.1007/978-3-540-70545-1_38

[18] Cas J. F. Cremers, Pascal Lafourcade, and Philippe Nadeau. 2009. Comparing State Spaces in Automatic Security Protocol Analysis. In *Formal to Practical Security - Papers Issued from the 2005-2008 French-Japanese Collaboration (Lecture Notes in Computer Science, Vol. 5458)*, Véronique Cortier, Claude Kirchner, Mitsuhiro Okada, and Hideki Sakurada (Eds.). Springer, 70–94. doi:10.1007/978-3-642-02002-5_5

[19] Eduardo Germano da Silva, Luis Augusto Dias Knob, Juliano Araujo Wickboldt, Luciano Paschoal Gaspary, Lisandro Zambenedetti Granville, and Alberto Schaeffer-Filho. 2015. Capitalizing on SDN-based SCADA systems: An anti-eavesdropping case-study. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 165–173.

[20] Stéphanie Delaune, Steve Kremer, and Mark Ryan. 2009. Verifying privacy-type properties of electronic voting protocols. *J. Comput. Secur.* 17, 4 (2009), 435–487. doi:10.3233/JCS-2009-0340

[21] Mohan Dhawan, Rishabh Poddar, Kshiteej Mahajan, and Vijay Mann. 2015. SPHINX: Detecting Security Attacks in Software-Defined Networks.

[22] D. Dolev and A. Yao. 1983. On the security of public key protocols. *IEEE Transactions on Information Theory* 29, 2 (1983), 198–208. doi:10.1109/TIT.1983.1056650

[23] Santiago Escobar, Catherine Meadows, and José Meseguer. 2009. *Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1–50. doi:10.1007/978-3-642-03829-7_1

[24] Marco Longhi Gelati, Giovanni Neri, Pierantonio Natali, Richard C. S. Morling, Gerald D. Cain, and Eugenio Faldella. 1984. MININET: A Local Area Network for Real-Time Instrumentation Applications. *Comput. Networks* 8 (1984), 107–131.

[25] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao. 2014. FLOWGUARD: Building Robust Firewalls for Software-Defined Networks. *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, HotSDN'14*, 97–102. doi:10.1145/2620728.2620749

[26] Jafar Haadi Jafarian, Ehab Al-Shaer, and Qi Duan. 2012. Openflow random host mutation: transparent moving target defense using software defined networking. In *Proceedings of the first workshop on Hot topics in software defined networks*. 127–132.

[27] Andrzej Kamisinski and Carol Fung. 2015. FlowMon: Detecting Malicious Switches in Software-Defined Networks. In *Proceedings of the 2015 ACM Workshop on Automated Decision Making for Active Cyber Defense*. ACM.

[28] Panos Kampanakis, Harry Perros, and Tsegereda Beyene. 2014. SDN-based solutions for moving target defense network protection. In *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*. IEEE, 1–6.

[29] Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. 2012. Veriflow: Verifying Network-wide Invariants in Real Time. *ACM SIGCOMM Computer Communication Review* 42 (2012), 467–472.

[30] Nadim Kobeissi, Georgio Nicolas, and Mukesh Tiwari. 2019. Verifpal: Cryptographic Protocol Analysis for the Real World. Cryptology ePrint Archive, Paper 2019/971. https://eprint.iacr.org/2019/971

[31] Robert Künnemann, Ilkan Esiyok, and Michael Backes. 2019. Automated Verification of Accountability in Security Protocols. In *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*. IEEE, 397–413. doi:10.1109/CSF.2019.00034

[32] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. 2010. Accountability: definition and relationship to verifiability. In *Proceedings of the 17th ACM Conference on Computer and Communications Security* (Chicago, Illinois, USA) *(CCS '10)*. Association for Computing Machinery, New York, NY, USA, 526–535. doi:10.1145/1866307.1866366

[33] Pascal Lafourcade and Maxime Puys. 2015. Performance evaluations of cryptographic protocols verification tools dealing with algebraic properties. In *International Symposium on Foundations and Practice of Security*. Springer, 137–155.

[34] Gavin Lowe. 1995. An Attack on the Needham-Schroeder Public-Key Authentication Protocol. *Inform. Process. Lett.* 56, 3 (1995), 131–133.

[35] Gavin Lowe. 1997. A Hierarchy of Authentication Specifications. In *Proceedings of the 10th IEEE Workshop on Computer Security Foundations (CSFW '97)*. IEEE Computer Society, USA, 31.

[36] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: Enabling Innovation in Campus Networks. *ACM CCR* (2008).

[37] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. 2013. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In *Computer Aided Verification, 25th International Conference, CAV 2013, Princeton, USA, Proc. (Lecture Notes in Computer Science, Vol. 8044)*, Natasha Sharygina and Helmut Veith (Eds.). Springer, 696–701. doi:10.1007/978-3-642-39799-8_48

[38] Kevin Morio and Robert Künnemann. 2021. Verifying Accountability for Unbounded Sets of Participants. In *34th IEEE Computer Security Foundations Symposium, CSF 2021, Dubrovnik, Croatia, June 21-25, 2021*. IEEE, 1–16. doi:10.1109/CSF51468.2021.00032

[39] Mahamat Charfadine Salim. 2019. *Gestion dynamique et évolutive de règles de sécurité pour l'Internet des Objets*. Ph. D. Dissertation. Université de Reims Champagne-Ardenne. Thèse de doctorat.

[40] Takayuki Sasaki, Christos Pappas, Taeho Lee, Torsten Hoefler, and Adrian Perrig. 2016. SDNsec: Forwarding Accountability for the SDN Data Plane. *NEC Corporation, Japan and ETH Zurich, Switzerland* (2016). Available: NEC Corporation and ETH Zurich.

[41] Cole Schlesinger, Alec Story, Stephen Gutz, Nate Foster, and David Walker. 2012. Splendid isolation: Language-based security for software-defined networks. In *Proc. of Workshop on Hot Topics in Software Defined Networking*.

[42] Arash Shaghaghi, Mohamed Ali Kaafar, and Sanjay Jha. 2017. WedgeTail: An Intrusion Prevention System for the Data Plane of Software Defined Networks. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM.
[43] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, and Guru Parulkar. 2009. Flowvisor: A network virtualization layer. *OpenFlow Switch Consortium, Tech. Rep* 1 (2009), 132.
[44] Takahiro Shimizu, Naoya Kitagawa, Kohta Ohshima, and Nariyoshi Yamai. 2019. WhiteRabbit: Scalable Software-Defined Network Data-Plane Verification Method Through Time Scheduling. (2019). doi:10.1109/ACCESS.2019.2929958
[45] P. P. S. Shin, V. Yegneswaran, and G. Gu. 2013. Avant-Guard: Scalable and Vigilant Switch Flow Management in Software-Defined Networks. *CProceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013), 413–424.
[46] RYU team. 2014. *RYU SDN Framework - English Edition.* RYU project team. https://books.google.fr/books?id=JC3rAgAAQBAJ
[47] T. Xing, D. Huang, C.-J. Chung L. Xu, and P. Khatkar. 2013. SnortFlow: an OpenFlow-based intrusion prevention system in cloud environment. *Second GENI Research and Educational Experiment Workshop (GREE)*, 89–92. doi:10.1109/GREE.2013.25
[48] Naoto Yanai. 2016. Towards Provable Security of Dynamic Source Routing Protocol and Its Applications. In *Advances in Conceptual Modeling - ER 2016 Workshops, AHA, MoBiD, MORE-BI, MReBA, QMMQ, SCME, and WM2SP, Gifu, Japan, November 14-17, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 9975)*, Sebastian Link and Juan Trujillo (Eds.). 231–239. doi:10.1007/978-3-319-47717-6_20
[49] V. Yegneswaran, M. Fong, M. Thottan, P. Porras, S. Shin, and G. Gu. 2012. A security enforcement kernel for OpenFlow networks. ACM.
[50] Peng Zhang, Hui Wu, Dan Zhang, and Qi Li. 2020. Verifying Rule Enforcement in Software Defined Networks With REV. *IEEE/ACM Transactions on Networking* (2020).
[51] Peng Zhang, Shimin Xu, Zuoru Yang, Hao Li, Qi Li, Huanzhao Wang, and Chengchen Hu. 2020. FOCES: Detecting Forwarding Anomalies in Software Defined Networks. (2020).

A Proof of Theorem 5.6

In the following, we prove the Theorem 5.6 stating if a protocol ensures *Strong Route Integrity* with regard to a route R then it ensures *Weak Route Integrity*.

Proof. For ease of notation, we refer to the events event($endS_i$) and event($beginS_j$) by e_i and b_j respectively. Moreover, in addition to the logical disjunction and conjunction symbols, we use the multiplicative and additive symbols for the same logical purposes respectively; and let $\{0, 1\}$ be our Boolean domain. We prove the theorem by induction.

- For $n = 1$, we suppose that our *Strong Route Integrity* property is verified. By definition, we have $(e_1 \Rightarrow b_1)(e_0 \Rightarrow b_0)(b_1 \Rightarrow e_0)$ which can be rewrite as follows $(\bar{e}_1 + b_1)(\bar{e}_0 + b_0)(\bar{b}_1 + e_0)$.
  By definition of the property of *Weak Route Integrity*, in the case of $n = 1$, $(e_1 \Rightarrow b_0 b_1)$ which is equivalent to $\bar{e}_1 + b_0 b_1$.
  We want to prove :
  $$(\bar{e}_1 + b_1)(\bar{e}_0 + b_0)(\bar{b}_1 + e_0) \Rightarrow \bar{e}_1 + b_0 b_1$$
  which is equivalent :
  $$(e_1\bar{b}_1) + (e_0\bar{b}_0) + (b_1\bar{e}_0) + \bar{e}_1 + b_0 b_1$$
  We use the equality $a + \bar{a}b = a + b$ to simplify the formula and prove that it is a tautology. We apply it successively with $\bar{e}_1$, $\bar{b}_1$, $\bar{e}_0$ and $\bar{b}_0$.
  $$\bar{b}_1 + (e_0\bar{b}_0) + (b_1\bar{e}_0) + \bar{e}_1 + b_0 b_1$$
  $$\Leftrightarrow$$
  $$\bar{b}_1 + (e_0\bar{b}_0) + \bar{e}_0 + \bar{e}_1 + b_0 b_1$$
  $$\Leftrightarrow$$
  $$\bar{b}_1 + \bar{b}_0 + \bar{e}_0 + \bar{e}_1 + b_0 b_1$$
  $$\Leftrightarrow$$
  $$\bar{b}_1 + \bar{b}_0 + \bar{e}_0 + \bar{e}_1 + b_1$$
  Which is always true since we have $\bar{b}_1 + b_1$. It concludes the case $n = 1$.
- We assume that the theorem holds for some arbitrary natural number n, that is:
  $$\left(\prod_{k=0}^{n}(e_k \Rightarrow b_k)\prod_{k=0}^{n-1}(b_{k+1} \Rightarrow e_k)\right) \Rightarrow \left(e_n \Rightarrow \prod_{k=0}^{n} b_k\right)$$
  We want to prove it is true for $n + 1$:
  $$\left(\prod_{k=0}^{n+1}(e_k \Rightarrow b_k)\prod_{k=0}^{n}(b_{k+1} \Rightarrow e_k)\right) \Rightarrow \left(e_{n+1} \Rightarrow \prod_{k=0}^{n+1} b_k\right)$$
  We notice that:
  $$\prod_{k=0}^{n+1}(e_k \Rightarrow b_k)\prod_{k=0}^{n}(b_{k+1} \Rightarrow e_k)$$
  $$\Leftrightarrow$$
  $$\left(\prod_{k=0}^{n}(e_k \Rightarrow b_k)\prod_{k=0}^{n-1}(b_{k+1} \Rightarrow e_k)\right)(\bar{e}_{n+1} + b_{n+1})(\bar{b}_{n+1} + e_n)$$
  $$\Leftrightarrow$$
  $$\alpha(\bar{e}_{n+1} + b_{n+1})(\bar{b}_{n+1} + e_n)$$
  where
  $$\alpha = \prod_{k=0}^{n}(e_k \Rightarrow b_k)\prod_{k=0}^{n-1}(b_{k+1} \Rightarrow e_k)$$
  We also denote
  $$\theta = \prod_{k=0}^{n} b_k$$
  We recall the following logical equivalencies: $ab \Rightarrow c \Leftrightarrow (a \Rightarrow c) + (b \Rightarrow c)$. Hence, we have:
  $$\begin{aligned} e_{n+1} \Rightarrow \prod_{k=0}^{n+1} b_k \quad &\Leftrightarrow \quad e_{n+1} \Rightarrow \left(b_{n+1}\prod_{k=0}^{n} b_k\right) \\ &\Leftrightarrow \quad \left(e_{n+1} \Rightarrow \prod_{k=0}^{n} b_k\right)(e_{n+1} \Rightarrow b_{n+1}) \\ &\Leftrightarrow \quad (e_{n+1} \Rightarrow \theta)\,(e_{n+1} \Rightarrow b_{n+1}) \\ &\Leftrightarrow \quad (\bar{e}_{n+1} + \theta)\,(\bar{e}_{n+1} + b_{n+1}) \end{aligned}$$
  We recall the following logical equivalencies: $a \Rightarrow bc \Leftrightarrow (a \Rightarrow b)(a \Rightarrow c)$. Hence, we need to prove:
  $$\alpha(\bar{e}_{n+1} + b_{n+1})(\bar{b}_{n+1} + e_n) \Rightarrow (\bar{e}_{n+1} + \theta)\,(\bar{e}_{n+1} + b_{n+1})$$
  $$\Leftrightarrow$$
  $$\begin{aligned}&[\big(\alpha(\bar{e}_{n+1} + b_{n+1})(\bar{b}_{n+1} + e_n) \Rightarrow (\bar{e}_{n+1} + \theta)\big) \\ &\big(\alpha(\bar{e}_{n+1} + b_{n+1})(\bar{b}_{n+1} + e_n) \Rightarrow (\bar{e}_{n+1} + b_{n+1})\big)]\end{aligned}$$
  $$\Leftrightarrow$$
  $$\begin{aligned}&[(\alpha \Rightarrow (\bar{e}_{n+1} + \theta)) + \big((\bar{e}_{n+1} + b_{n+1})(\bar{b}_{n+1} + e_n) \Rightarrow (\bar{e}_{n+1} + \theta)\big) \\ &(\alpha \Rightarrow (\bar{e}_{n+1} + b_{n+1})) + \big((\bar{e}_{n+1} + b_{n+1})(\bar{b}_{n+1} + e_n) \Rightarrow (\bar{e}_{n+1} + b_{n+1})\big)]\end{aligned}$$
  To prove the first part we have:
  $$(\alpha \Rightarrow (\bar{e}_{n+1} + \theta)) + \big((\bar{e}_{n+1} + b_{n+1})(\bar{b}_{n+1} + e_n) \Rightarrow (\bar{e}_{n+1} + \theta)\big)$$

$$\Leftrightarrow$$

$$\bar{\alpha} + \bar{e}_{n+1} + \theta + e_{n+1}\bar{b}_{n+1} + b_{n+1}\bar{e}_n + \bar{e}_{n+1} + \theta$$

This is true due to the induction hypothesis: $\alpha \Rightarrow (e_n \Rightarrow \theta) \Leftrightarrow \bar{\alpha} + \bar{e}_n + \theta$, which is a tautology. We prove the second part as follows:

$$(\alpha \Rightarrow (\bar{e}_{n+1} + b_{n+1})) + \big((\bar{e}_{n+1} + b_{n+1})(\bar{b}_{n+1} + e_n) \Rightarrow (\bar{e}_{n+1} + b_{n+1})\big)$$

$$\Leftrightarrow$$

$$\bar{\alpha} + \bar{e}_{n+1} + b_{n+1} + e_{n+1}\bar{b}_{n+1} + b_{n+1}\bar{e}_n + \bar{e}_{n+1} + b_{n+1}$$

$$\Leftrightarrow$$

$$\bar{\alpha} + \bar{e}_{n+1} + b_{n+1} + e_{n+1} + b_{n+1}\bar{e}_n + \bar{e}_{n+1} + b_{n+1}$$

It is a tautology since we have $e_{n+1} + \bar{e}_{n+1}$. This concludes the proof.

□

B SDNsec Processes Model in ProVerif

The full processes of the controller, the ingress switch, the core switch and the egress switch, as modeled in ProVerif, are depicted in Figures 13, 10, 11 and 12 respectively. It should be noted that the variables and names' types have been omitted for the sake of simplicity of presentation.

```
let ProcessS0(sk0) =
  in(c, n_s0);
  event seqUniqueness(st, n_s0);
  new p;
  let p_encrypted = senc(p, sk0) in
  out(c_s0, (p_encrypted, n_s0));
  in(c_s0, y);
  let(F, t, xegr1, MAC1, xegr2, MAC2) =
  sdec(y, sk0) in
  let B = (F, t) in
  let MAC0 = MAC(sk0, B) in
  let C = (F, n_s0) in
  let PVF0 = MAC(sk0, C) in
  event begins0(egr0);
  out(c, (F, t, n_s0, PVF0, xegr1, MAC1,
  xegr2, MAC2, p)).
```

Figure 10: Process of Ingress switch S_0 in ProVerif.

```
let ProcessS1(sk1) =
  in(c, (F, t, n_s0, PVF0, xegr1, MAC1,
  xegr2, MAC2, p));
  let B = (F, t) in
  if (MAC1 = MAC(sk1, (xegr1, B, B))) then
  event ends0;
  let n_s1 = n_s0 + 1 in
  let PVF1 = MAC(sk1, (PVF0, F, n_s1)) in
  event begins1(xegr1);
  out(c, (F, t, n_s1, PVF1, xegr1, MAC1,
  xegr2, MAC2, p)).
```

Figure 11: Process of Core switch S_1 in ProVerif.

```
let ProcessS2(sk2) =
  in(c, (F, t, n_s1, PVF1, xegr1, MAC1,
  xegr2, MAC2, p));
  let B = (F, t) in
  let FE1 = (xegr1, MAC1) in
  if (MAC2 = MAC(sk2, (xegr2, FE1, B))) then
  event ends1;
  let n_s2 = n_s1 + 1 in
  let PVF2 = MAC(sk2, (PVF1, F, n_s2)) in
  event begins2(xegr2);
  out(c_s2, (F, PVF2)).
```

Figure 12: Process of Egress switch S_2 in ProVerif.

```
let ProcessController(sk0, sk1, sk2) =
  in(c, (xegr1, xegr2));
  in(c_s0, (p_encrypted, n_s0));
  let p = sdec(p_encrypted, sk0) in
  new F; new t;
  let B = (F, t) in
  let FE0 = B in
  let MAC1 = MAC(sk1, (xegr1, FE0, B)) in
  let FE1 = (xegr1, MAC1) in
  let MAC2 = MAC(sk2, (xegr2, FE1, B)) in
  let FE2 = (xegr2, MAC2) in
  let C = (F, n_s0) in
  let PVF0 = MAC(sk0, C) in
  let PVF1 = MAC(sk1, (PVF0, F, n_s0 + 1)) in
  let PVF2 = MAC(sk2, (PVF1, F, n_s0 + 2)) in
  let y = senc((F, t, xegr1, MAC1, xegr2,
  MAC2), sk0) in
  out(c_s0, y);
  in(c_s2, (=F, xPV2));
  if xPV2 = PVF2 then event OK else event KO.
```

Figure 13: Process of Controller in ProVerif.