Implementing and Testing a Probing-Based Path Verification for PolKA Source Routing Protocol

Henrique C. Layber, Roberta L. Gomes, Magnos Martinello

[1]Department of Informatics
Federal University of Espírito Santo (UFES) – Vitória, ES – Brazil

henrique.layber@edu.ufes.br,{roberta.gomes, magnos.martinello}@ufes.br

Abstract. *This work presents an implementation as proof-of-concept for testing of a path-aware routing named* ***PathSec****. It relies on a path probing and verification mechanism for the source-routing protocol PolKA. The mechanism leverages a combination of hash functions, computed by stateless core switches, enabling a trusted party to verify whether a packet traversed the network along the source-defined path. The implementation was carried out in Mininet using BMv2 switches programmed in P4. Different adversity scenarios are evaluated to assess the system's robustness and security against specific attacks. The results confirm that the probing-based approach successfully detected the various path deviation attacks or misconfigurations.*

Resumo. *Este trabalho apresenta uma implementação como prova de conceito para testar um roteamento ciente de caminho denominado* ***PathSec****. O trabalho se baseia em um mecanismo de sondagem e verificação de caminho para o protocolo de roteamento de fonte PolKA. O mecanismo utiliza uma combinação de funções hash, calculadas por switches de núcleo sem estado, permitindo que uma entidade confiável verifique se um pacote percorreu a rede conforme o caminho definido na origem. A implementação foi realizada no Mininet usando switches BMv2 programados em P4. Diferentes cenários adversos foram avaliados para testar a robustez e segurança do sistema contra ataques específicos. Os resultados confirmam que a abordagem baseada em sondagem detectou com sucesso diversos ataques de desvio de caminho ou configurações incorretas.*

Keywords – Path Aware; Network Security; Path Verification; In-networking Programming

1. Introduction

Ever since *Source Routing* (SR) was proposed, there has been a need to ensure that packets traverse the network along the paths selected by the source, not only for security reasons, but also to ensure that the network is functioning properly and correctly configured. This is particularly important in the context of *Software-Defined Networks* (SDNs), where the control plane can select paths based on a variety of criteria[Jyothi et al. 2015].

In this paper, we propose an implementation on Programming Protocol Independent Processors (*P4*) for a Probing-Based Path Verification (PBV) mechanism originally proposed in **PathSec** [Martinello et al. 2024]. This mechanism provides proofs of packet forwarding, confirming that an expected route was indeed used for a packet. This is achieved by using a composition of hash functions on stateless core switches, combined

with the switches native secret code (node_ids) and the respective route's output port, which results into a light-weight multi-signature. This signature can then be compared to a reference signature generated by a trusted party, the network controller. This work tests a proof-of-concept implementation under different adversity scenarios and results show the approach is effective as a proof-of-transit solution.

The structure of this paper is as follows: section 2 outlines the problem definition and the proposed solution. section 3 describes the implementation details, including the architecture and the tools used. section 4 presents the scenarios used to evaluate the system and the results, and section 5 concludes the paper.

All the code and test results are available online[1].

2. Problem definition

A **PathSec** network consists of a logically centralized SDN Controller that selects routing paths and configures the core and edge nodes. While ingress and egress edges encapsulates/decapsulates metadata into/from packets, core nodes execute basic packet forwarding with light signatures operations (proof-of-transit).

PathSec is based on Polka's source routing which explores the Chinese Remainder Theorem[Dominicini et al. 2020]. *PolKA* assigns a unique Route Identifier (route_id) to each path in the core network, and for each core switch it defines a secret node_id. Once a packet is associated to a specific route, each packet in this packet will carry in its header the respective route_id (embeded by the ingress edge). Then the core switches use the route_id and their respective node_ids to calculate the output port_id (it is a result of the Residue Number System (RNS)-based encoding[Martinello et al. 2024], which is out of the scope of this paper), routing the packets to the next core node in the expected path for the route_id.

Furthermore, **PathSec** designs a simplified signing mechanism using the uniqueness of the switch-port pair (node_id, port_id) for each route_id and its inherent secret component in the routing system, combined with cryptographic hashing functions. Every switch-port pair acts as a native secret code, enabling a lightweight multi-signature scheme based on Hash-based Message Authentication Code (HMAC): before routing a packet, a core node replaces the lightweight signature from the previous core node (embedded in the packet header), with a new one, hashing the previous one with the pair (node_id, port_id). This mechanism ensures an inherent property of the routing system that facilitates efficient path verification.

In the complete solution proposed by **PathSec**, once packets reach the egress edges, the final lightweight multi-signatures are extracted from the packets and sent to smart contracts deployed in a blockchain. These signatures are then compared to reference signatures previously registered by a trusted party (the network controller), allowing to verify packets' *proof-of-transit* and to register path compliance.

2.1. Problem formalization

Let i be the source node (**i**ngress node) and e be the destination node (**e**gress node). Let the path from i to e $P_{i \to e}$ be a sequence of nodes: the path is defined by $P_{i \to e} =$

[1] github.com/Henriquelay/polka-halfsiphash

$(i, s_1, s_2, ..., s_{n-1}, s_n, e)$, s_n being the n-th core node in the path. In SR protocols, packets' routes through the core network are defined in i. That is, i sets the packet header with enough information for each core node to calculate the next hop.

The main problem we are trying to solve is path verification, that is, to have a way to ensure if the packet follows the path defined. A solution must be able to identify if the packet: (i) has passed through all nodes in the Path; (ii) has passed through the correct order of nodes; (iii) has **not** passed through any node that is **not** in the path.

More formally, given a routed sequence of nodes $P_{i \to e}$, and a captured sequence of nodes actually traversed P_j, a solution must identify if $P_{i \to e} = P_j$. Notably, it does not require validation, that is, it needs only to respond if $P_{i \to e} = P_j$ and does not need to know any of their contents or check their validity as a route.

2.2. Multi-signature model for Path Verification

To balance trace effectiveness and scalability, **PathSec** proposes using packets themselves as probes[Martinello et al. 2024][Ben Basat et al. 2020], with a lightweight multi-signature, keeping the header size fixed. That is, each core node hashes the previous node's signature with a key only itself (and the Controller) can possess, and pass the hash result forward, replacing the previous, and all following core nodes will do the same until the edge node is reached.

Each node's execution plan is stateless. So, in terms of signature composition, a node n_i can be viewed as a function $f_{n_i}(x)$. A node can alter the header of the packet, which is used to push information downstream, allowing ultimately to detect if the path taken is correct.

In order to represent all nodes by the same function, we assign a distinct key value k for each node n_i, and use a bivariate function $f(k_{n_i}, x) = f_{n_i}(x)$. By using functions in two variables and enforcing one of the variables to have any value that's unique between nodes, we ensure that the function's result is unique for each switch as long as the function is collision resistant enough, that is, $f_{n_a}(x) \neq f_{n_b}(x) \iff y \neq z$.

Using function composition is appropriate, as it preserves the order-sensitive property of the path, since $f \circ g \neq g \circ f$ in a general case. In our model, each node will execute a single function of this composition, using the previous node's output as input. In this way, $(f_{n_1} \circ f_{n_2} \circ f_{n_3})(x) = f(k_{n_3}, f(k_{n_2}, f(k_{n_1}, x)))$, f being a collision-resistant function, preferably a cryptographic hash function.

3. Implementing and Testing a Probing-Based Path Verification (PBV)

PBV consists of sending special packets (probes) with path-verification capabilities. The path verification system used is describe in subsection 2.2. Some assumptions are made: (i) Each node is assumed to be secure, that is, no node will alter the packet in any way that is not expected. This is a common assumption in SDN networks, where a trusted party is the only entity that can alter the network state; (ii) Every link is assumed to be perfect, that is, no packet loss, no packet corruption, and no packet duplication; (iii) Protocol boundary is IPv4, this means that **PathSec** is only used inside this network, and only IPv4 is used outside; (iv) All paths are assumed to be valid and all information correct unless stated otherwise; (v) Hash collisions will not happen. Furthermore, this implementation

is a proof-of-concept, and there is little concerns about performance and protocol implementation: only IPv4 is supported (see item (iii)). The following subsections detail the setup and implementation.

3.1. Setup

P4 is a language used to program the data plane of network devices [Bosshart et al. 2014]. Behavioral Model 2 (*BMv2*) is a software switch that supports the *P4* language [Consortium 2025]. Mininet is a network emulation tool that allows the creation of virtual networks with a controlled environment [Lantz et al. 2010]. Mininet-wifi[Fontes et al. 2015] is a Mininet fork used in this work due to its compatibility with *BMv2*. Despite the name, no Wi-Fi emulating feature was used.

The Controller is responsible for the network setup and for calculating reference signatures. In our implementation, the Python Script that controls Mininet acts a controller, and itself is not a network entity in Mininet. Scapy is a Python library that allows the creation and manipulation of packets [secdev 2003]. It is used to parse packets both for Controller operation and in the experiments and automate tests. The Tester is responsible for triggering Mininet hosts to send packets, setting up Scapy for parsing them and then asserting their signatures to Controller-calculated references. The setup is shown in Figure 1a.

3.2. Implementation

This work extends previous work that implements *PolKA* in Mininet[Dominicini et al. 2021][2], by adding support to **PathSec**.

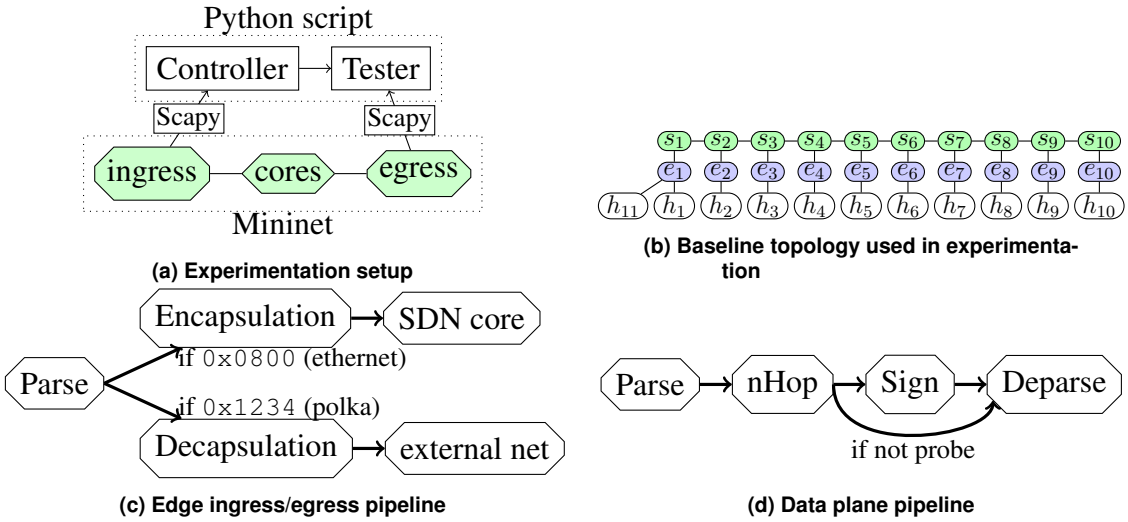
We define the header fields `timestamp` and `l_hash`, both 32-bits wide, to indicate the uniquely-generated number and the current light-weight multi-signature, respectively. Due to header incompatibility, probe packets operate using a different version number in the `version` field, so it is interoperable with *PolKA* by switching between probes and regular packets using the `version` header. *PolKA* packets use version `0x01`, and probe packets use version `0xF1`.

Figure 1b shows the used topology used in the experiments where s_i is a switch, e_i is an edge node, and h_i is a host. The topology is a linear chain with 10 switches and 11 hosts. The links are bidirectional, but only one direction is shown for simplicity.

Edge nodes can receive both *PolKA* packets or IPv4 packets. If the IPv4 protocol `ethertype` header field is detected (`0x0800`), it must be a packet from outside the network, so it proceeds with the role of an ingress edge and *PolKA* headers need to be added. Let this process be called *encapsulation*. If a *PolKA* protocol `ethertype` field is detected (`0x1234`), it must be a packet from the network core, so it proceeds with the role of an egress edge and the original IPv4 packet must be unwrapped. Let this process be called *decapsulation*. This flow is illustrated in Figure 1c.

PolKA headers consists of the route polynomial (`route_id`), along with `version`, `ttl` and `proto` (stores the original `ethertype`). During encapsulation, the packet can be made into a probe packet by setting the `version` header field to `0xF1`

[2] `https://github.com/nerds-ufes/polka`



and further defining the 32-bit fields `timestamp` to any unique value and `l_hash` initially equal to `timestamp`.

On the core network, the hash function used is *SipHash-2-4-32*[Aumasson and Bernstein 2012] (*HalfSipHash*). After calculating the next output port (`nhop`) and if the packet is a probe packet, `l_hash` is computed as

$$\texttt{l\_hash} \leftarrow \textit{SipHash2-4-32}(\texttt{node\_id}||\texttt{nhop}||\texttt{timestamp}||\texttt{0000000}, \texttt{l\_hash})$$

Meaning that a 64-bit string made of concatenating `node_id`, `nhop`, `timestamp` with the remaining bits set to `0` for padding is used as key for the *HalfSipHash* function hashing `l_hash`. This is illustrated on Figure 1d.

4. Testing

This section presents the results for various tests representing different adversity scenarios. Each scenario is designed to evaluate the robustness and security of the system under specific attack or misconfiguration conditions.

4.1. Adversity scenarios

The scenarios include addition, detour, skipping, and out-of-order packet delivery. We define any switch that is not in the position defined in Figure 1b as an *attacker*, not only an intruder device, so it includes misconfigured switches, controller malfunction, and any other kind of network-affecting anomaly that results in these specified scenarios:

i. **Addition** (Figure 2a): An attacker switch is added between s_5 and s_6. It perfectly replicates the behavior of the original switch, but it does not have the same `node_id`, as it is a secret key, and the hashes will not match when validating the path past the attacker.
ii. **Skipping** (Figure 2b): A switch is removed between s_4 and s_6. The packet will skip removed switch, and the hashes will not match when validating the path past the removed switch.

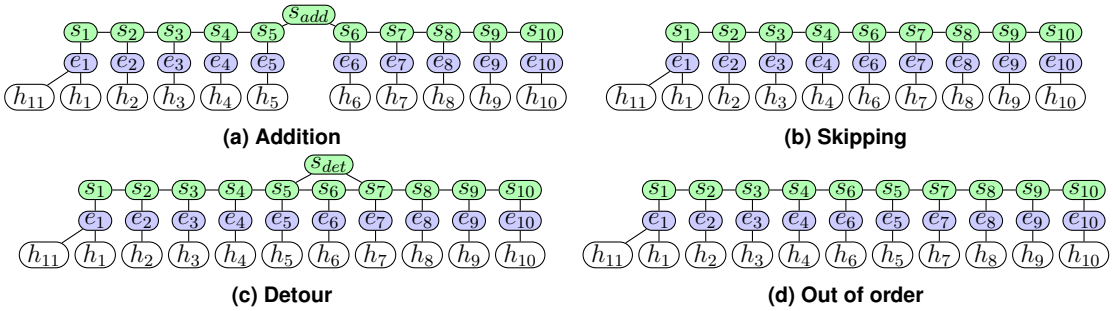


Figure 2: Topologies for tested scenarios

Reference		Addition		Skipping		Detour		Out-of-order	
s_0	0x61e8d6e7	s_0	0x61e8d6e7	s_0	0x61e8d6e7	s_0	0x61e8d6e7	s_0	0x61e8d6e7
s_1	0xd25dc935	s_1	0xd25dc935	s_1	0xd25dc935	s_1	0xd25dc935	s_1	0xd25dc935
s_2	0x245b7ac5	s_2	0x245b7ac5	s_2	0x245b7ac5	s_2	0x245b7ac5	s_2	0x245b7ac5
s_3	0xa3b38b83	s_3	0xa3b38b83	s_3	0xa3b38b83	s_3	0xa3b38b83	s_3	0xa3b38b83
s_4	0x26aee736	s_4	0x26aee736	s_4	0x26aee736	s_4	0x26aee736	s_4	0x26aee736
s_5	0xf9b47914	s_5	0xf9b47914			s_5	0xf9b47914		
		s_{add}	0x250822a2					s_6	0x4b5a6c5a
s_6	0x18c6d8d1	s_6	0x20d21d49	s_6	0x4b5a6c5a	s_{det}	0x250822a2	s_5	0xde3862a0
s_7	0xb69b99ec	s_7	0xdd03c4c2	s_7	0x002346d3	s_7	0x40298bb9	s_7	0x648556ec
s_8	0xfe6117f8	s_8	0x292e8608	s_8	0x7ec711aa	s_8	0xe13dcc9b	s_8	0x144e1d1b
s_9	0xc8d9fbde	s_9	0x32419384	s_9	0x5ee32b7b	s_9	0x1bf62c19	s_9	0x9e818f34
s_{10}	0xa6293a25	s_{10}	0xf4bcdf07	s_{10}	0xc973a219	s_{10}	0xdd3a6675	s_{10}	0x6f694bc

Table 1: Intermediary `l_hash` for different test scenarios with the same seed

iii. **Detour** (Figure 2c): A switch is detoured between s_5 and s_7 into an attacker by hijacking the port that the packet should have taken exiting s_5 and delivering the packet in a new port to s_7, posing as s_6. The hashes will not match when validating the path past the attacker.

iv. **Out-of-order** (Figure 2d): Links are changed between s_5 and s_6. The packet will be delivered to s_6 before s_5, and the hashes will not match when validating the path past the attack.

Table 1 shows the intermediary checksums for each scenario, highlighting the differences caused by the adversarial conditions, although in deployment, only the last `l_hash` is every used for verification. Note that a scenario may result in a different amount of hashes to compare due to different amount of intermediary hops, so the table isn't completely filled to represent a hop that does not exist on a particular route. For all scenarios, a ping from h_1 to h_{10} was performed, resulting in `route_id` = `0x707b3a1d61d1d0d8b9fc91e442d0360dfc8bba4`.

4.2. Discussion

Recovering data from Mininet switches is not trivial, as it does not provide a straightforward way to extract data from the switches. This limitation makes it difficult to analyze the data in real-time, so our solution depends on sniffer libraries (Scapy) to capture packets and extract the metadata. This limitation could be overcome by having the switches natively output the metadata to our controller, which would allow for a more performant and scalable solution. It was not implemented to keep the implementation minimal.

The cryptographically secure hash solution is only secure when the key is a secret. The `node_id` secret is output in `ttl` *PolKA* header field for debugging and presentation

purposes, and this implementation should never be deployed in a real-world scenario. If the key is compromised, the entire system is compromised, as a malicious actor can easily generate the same checksums and be undetected, essentially signing whichever packets they want. Having the switch entry port included in the verification would mitigate this issue, as the attacker would need to know the exact port the packet entered the switch.

5. Conclusion and Future Work

The implementation and evaluation of **PathSec** demonstrate the feasibility of a path-aware routing mechanism that enhances network security through path verification. By leveraging hash-based authentication computed by stateless core switches, the system ensures that packets adhere to their intended routes, enabling a trusted entity to detect deviations caused by attacks or misconfigurations. The proof-of-concept implementation in Mininet with BMv2 switches programmed in P4 highlights the practicality of deploying such a mechanism in programmable networks. The experimental results confirm the effectiveness of the probing-based approach in identifying various adversarial scenarios, reinforcing its potential as a security measure for source-routing protocols like *PolKA*.

While the study validates the fundamental principles behind **PathSec**, future work could explore optimizations to improve scalability and efficiency in larger network deployments. Further research is needed to evaluate the system's performance under high traffic loads and its adaptability to dynamic network conditions. Additionally, integrating **PathSec** with other security mechanisms could enhance its resilience against more sophisticated attack strategies. Overall, this work provides a first step for advancing path-aware security in programmable networks and highlights the potential of verification-based approaches to strengthen network integrity.

References

[Aumasson and Bernstein 2012] Aumasson, J.-P. and Bernstein, D. J. (2012). SipHash: a fast short-input PRF. Cryptology ePrint Archive, Paper 2012/351.

[Ben Basat et al. 2020] Ben Basat, R., Ramanathan, S., Li, Y., Antichi, G., Yu, M., and Mitzenmacher, M. (2020). Pint: Probabilistic in-band network telemetry. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 662–680, New York, NY, USA. Association for Computing Machinery.

[Bosshart et al. 2014] Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. (2014). P4: programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95.

[Consortium 2025] Consortium, P. L. (2025). Behavioral model. `https://github.com/p4lang/behavioral-model`. Accessed: 2025-03-15.

[Dominicini et al. 2021] Dominicini, C., Guimarães, R., Mafioletti, D., Martinello, M., Ribeiro, M. R. N., Villaça, R., Loui, F., Ortiz, J., Slyne, F., Ruffini, M., and Kenny, E. (2021). Deploying polka source routing in p4 switches : (invited paper). In *2021 International Conference on Optical Network Design and Modeling (ONDM)*, pages 1–3.

[Dominicini et al. 2020] Dominicini, C., Mafioletti, D., Locateli, A. C., Villaca, R., Martinello, M., Ribeiro, M., and Gorodnik, A. (2020). Polka: Polynomial key-based architecture for source routing in network fabrics. In *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, pages 326–334.

[Fontes et al. 2015] Fontes, R. R., Afzal, S., Brito, S. H. B., Santos, M. A. S., and Rothenberg, C. E. (2015). Mininet-wifi: Emulating software-defined wireless networks. In *2015 11th International Conference on Network and Service Management (CNSM)*, pages 384–389.

[Jyothi et al. 2015] Jyothi, S. A., Dong, M., and Godfrey, P. B. (2015). Towards a flexible data center fabric with source routing. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, New York, NY, USA. Association for Computing Machinery.

[Lantz et al. 2010] Lantz, B., Heller, B., and McKeown, N. (2010). 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, New York, NY, USA. Association for Computing Machinery.

[Martinello et al. 2024] Martinello, M., Gomes, R. L., Borges, E. S., Layber, H. C., Bonella, V. B., Dominicini, C. K., Guimarães, R., Ribeiro, M., and Barcellos, M. (2024). Pathsec: Path-aware secure routing with native path verification and auditability. In *2024 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 1–7.

[secdev 2003] secdev, B. P. (2003). Scapy. `https://scapy.net/`. Accessed: 2024-09-26.