2023 IEEE 9th International Conference on Network Softwarization (NetSoft) | 979-8-3503-9980-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/NetSoft57336.2023.10175482

In-situ Proof-of-Transit for Path-Aware Programmable Networks

Everson Scherrer Borges*†, Vitor Berger Bonella*, Abraão Jesus dos Santos*, Gabriel Tetzner Menegueti*, Cristina Klippel Dominicini†, Magnos Martinello*
*Department of Informatics, Federal University of Espírito Santo, Brazil.
Emails: {vitor.bonella, abraao.santos, gabriel.menegueti}@edu.ufes.br, magnos.martinello@ufes.br
†Department of Informatics, Federal Institute of Espírito Santo, Brazil
Email: {everson, cristina.dominicini}@ifes.edu.br

***Abstract*—This paper presents a scalable and efficient solution for secure network design that involves the selection and verification of network paths. The proposed approach addresses the challenge of extending compliance policies to cover path-aware programmable networks by decoupling the routing/forwarding mechanisms from the Proof-of-Transit (PoT) implementation. Thus, two concepts are bounded: i) a source routing mechanism based on a fixed routeID representing a unique identifier per path, which serves as a key for the PoT lookup table; ii) the "in situ" that allows to collect telemetry information in the packet while the packet traverses a path. The former enables path selection with policy at the edge, while the later allows to perform path verification without extra probe-traffic. A P4 programmable language prototype demonstrates the effectiveness of this approach to protect against deviation attacks with low overhead. The results show a significant reduction in network's forwarding state for fat-tree topologies depending on the workload per path (flows/path).**

***Index Terms*—Path-Aware; Path Verification; Proof-of-transit; IOAM; In-networking Programming**

I. Introduction

In the current Internet architecture, routers determine how a packet should be forwarded based on its destination. The forwarding decision relies on each router's local routing table. Each entry in the routing table associates a reachable destination with the next-hop on the path. Unfortunately, in this architecture, there is almost no means for path verification [1], and an application can only assume that a packet will eventually reach the destination without selecting a specific path [2], opening up numerous attack possibilities. For example, an adversary may deviate the traffic violating the security policy.

Internet Service Providers (ISPs) play a critical role in ensuring reliable data delivery. To maintain the highest level of service, ISPs must meet a Service Level Agreement (SLA). In today's rapidly evolving technological landscape, the demand for Network Function Virtualization (NFV [3]) and modern service chaining is increasing [4]. These new technologies require compliance with specific policies or regulations that specify the path that data must take through the network, including the specific nodes it must pass through. Additionally, ISPs must be able to prove that packets have passed through a set of service functions to ensure the delivery of accurate and secure data [5]. In short, ISPs must meet the SLA for data delivery in their network and comply with regulations to maintain the trust of their customers and remain competitive.

To meet these requirements, modern routing must have two properties: **path-awareness** [2] and **path-verifiability** [1]. Path-awareness allows endpoints to choose network paths by exposing path information at the network or transport layers. Path-verifiability provides Proof-of-Transit mechanisms to securely confirm that all packets within a given path passed through the intended nodes.

In this paper, we examine two concepts: (i) **Strict Source Routing (SSR)**, where a source node adds a route label in the packet header to specify all the nodes in an end-to-end path [6]; and (ii) **In situ Operations, Administration, and Maintenance (IOAM)**, which collects operational and telemetry information in the packet while the packet traverses a path [7]. The former enables path selection and reduces the control signaling and latency related to path setup [6], while the later allows to perform path verification without extra probe-traffic [5]. Our proposal, called PoT-PolKA, is a novel lightweight and scalable in-situ PoT approach for path-aware programmable networks that combines the SSR provided by **Pol**ynomial **K**ey-based **A**rchitecture (**PolKA**) [6] with a new version of the PoT mechanism introduced by a IETF RFC draft [5] based on the Shamir's secret sharing scheme [8].

The design relies on the semantic of PolKA routing that specifies a *routeID* which is decoded at each node by a polynomial modulo operation for packet forwarding. This *routeID* expresses the entire path for the packet, i.e., not its destination address, but how to traverse each node until it reaches the destination. The second part is devoted to path-verifiability for which a shared secret is distributed by the controller to the nodes in the path through a secure channel. At the ingress edge, metadata is added to in-situ header, and a cumulative number is updated at each hop. At the egress edge, the verifier node checks if the cumulative number in the packet header matches its secret.

In comparison to the IETF RFC Draft [5], this work offers a combined SSR and packet-path binding approach with **PolKA** and introduces Mersenne numbers for a feasible implementation in programmable switches. The unique *routeID*, unchanged throughout the path, allows aggregation of flows avoiding to store per-flow state on routers. This

Authorized licensed use limited to: TRINITY COLLEGE LIBRARY DUBLIN. Downloaded on July 18,2023 at 09:55:52 UTC from IEEE Xplore. Restrictions apply.

routeID serves as a key for PoT table lookup to support the path verification. Our implementation uses Mersenne numbers for efficient computation of modulo operations required by the Shamir mechanism. To the best of our knowledge, this is the first open-source implementation of PoT with Shamir mechanism using P4.

As proof-of-concept, a prototype is built for PoT-PolKA network (edge and core nodes) developed in P4 programmable language. For validation, PoT-PolKA scalability analysis is presented for different workloads. For performance benchmarks, experimental results are carried out in Mininet emulated environment, showing PoT-PolKA low overhead to provide protection against traffic deviation attacks.

II. Problem Definition and Related Works

In this paper, we want to answer the following research question: How to prove that a traffic flow follows the correct path for path-aware programmable networks? To answer this question, we investigate the feasibility of a lightweight and scalable in-situ Proof-of-Transit approach for path-aware programmable networks.

A. Definitions and Scope

Proof-of-Transit (PoT) is a security mechanism for verifying the path through which a packet was forwarded [5]. PoT is also known as path verification [1], which enables the destination to securely retrieve paths taken by packets. In contrast to conventional traceroute solutions, PoT uses cryptography to secure the metadata used to retrieve packet trajectories. PoT brings more transparency of the underlying packet delivery and, consequently, empowers operators/users to enforce their preferred paths.

For the path-aware properties of the network, we provide a SSR mechanism that allows the source to select the specific path that the packet will traverse. The term "in-situ" means that the Operations, Administration, and Maintenance (OAM) information is collected within the packet while the packet traverses a network domain, rather than send extra packets dedicated to OAM [7]. Path enforcement, path validation and the discovery of path properties [2] are important security aspects [1], but they are out of the scope of this paper.

Our assumption for the adversary model is that a network attacker is able to deviate the traffic violating its security policy, which leads to *forwarding inconsistencies* (i.e., path deviation attacks) as follows [1]:

Skipping: A malicious router redirects the packet and skips other router(s) on the path, as illustrated in Fig. 1. For example, it skips 3 and the packet traverses the path $1 \rightarrow 2 \rightarrow 4 \rightarrow 5$. Packet forwarding skips one or more intermediate routers.Thus, some routers on the intended path do not forward the packet.

Addition: Packet forwarding first detours and then returns to the expected path. Packet forwarding thus visits one or more routers that otherwise are not expected. For example, in Fig. 2, the modified path is $1 \rightarrow 2 \rightarrow 6 \rightarrow 3 \rightarrow 4 \rightarrow 5$.

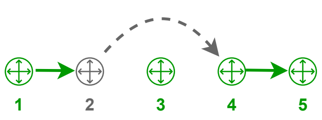


Figure 1: Skipping

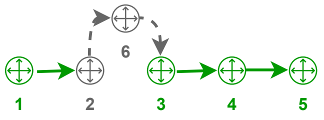


Figure 2: Addition

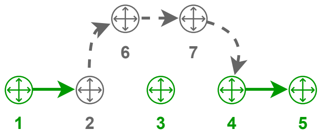


Figure 3: Partial Detour

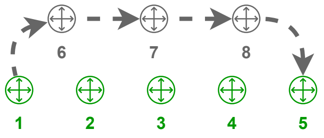


Figure 4: Complete Detour

Path detour: Malicious router R_i causes a packet to deviate from the intended path, but later the packet returns to the correct router. Partial detour is illustrated in Fig. 3, where the forwarding deviates from some but not all of the expected routers (path $1 \rightarrow 2 \rightarrow 6 \rightarrow 7 \rightarrow 4 \rightarrow 5$). Complete detour is shown in Fig. 4, where the forwarding deviates entirely from the expected sequence (path $1 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 5$).

B. Related Works

In this section, we review some ideas of related works on PoT. Among the main techniques to provide PoT are: nested encryption, nested hash, and cryptography mechanisms [7].

ICING is a *nested encryption* technique that uses aggregate MACs and self-certifying names. It relies on configuring shared keys for each node pair to compute and verify proofs with a stack of self-certifying node names and xor operations. However, it incurs high control plane overhead due to proof-of-consent requirements from path nodes and has a complex implementation due to its variable header stack requirement.

Different from ICING, OPT [9] does not include the list of on-path nodes in the packet header. OPT assumes that all nodes trust the source S, and each on-path node N_i generates a shared symmetric key k_i with S. OPT refers to secrets as origin and path validation (OPV); it allocates one OPV_i field for each N_i in the packet header with 128 bits each. Although it imposes less complexity to the control plane compared to [10], it also requires variable header size (on-path OPV_i). Extended-OPT [11] is a variant of OPT that suggests to keep the same complexity when nodes do not trust the source.

Orthogonal sequence verification (OSV) [12] belongs to the *nested hash* category. It follows the same design principle as OPT, but with lighter orthogonal sequences. OSV also relies on a Path Validation Field (PVF) and an Original Validation Field (OVF) per on-path node, but achieves faster computation of those fields. Specifically, the source first generates an $m \times m$ Hadamard matrix H [13], using vectors of H as the credentials for on-path nodes.

Still in the nested hash category, PPV [14] takes a different approach with probabilistic path validation. Its premise is that each packet does not need to be marked by all of the routers it visits (at most two routers along the forwarding path). It is based on per-flow path validation, so that PPV routers only mark packets with a certain probability.

Authorized licensed use limited to: TRINITY COLLEGE LIBRARY DUBLIN. Downloaded on July 18,2023 at 09:55:52 UTC from IEEE Xplore. Restrictions apply.

Table I: PoT design comparison

Method	Routing	Path Info	Route Identifier	Policy	Control Plane Overhead	Data Plane Overhead
Original PoT RFC draft	Table-Based Routing	None	None	Per flow	Routing tables PoT configs	Path tracing POT metadata
PoT RFC Draft with Segment Routing	List-based Source Routing	In clear	Variable	Per flow	PoT configs	Path tracing PoT metadata routeID
PoT-PolKA	PolKA Source Routing	Encoded	Constant	Per path	POT configs	POT metadata routeID

As an example of *cryptography mechanisms*, Shamir's secret sharing scheme [8] is a well-known method to secure a secret in a distributed way. In [5], the authors explore it to provide an in-situ PoT solution in an IETF RFC draft. It splits a secret into multiple parts and share them with the nodes of the path. The main idea uses polynomial interpolation over finite fields that can be divided in two main parts, share distribution and secret reconstruction (based on Lagrange Interpolation Formula [15] used to obtain the original polynomial). The main advantage with respect to the previous works is its lightweight in-situ fingerprint [7] with a small cumulative signature and low control plane overhead.

However, this PoT proposal has some drawbacks: (i) it depends on an integer modulo operation that is not commonly supported in programmable switches; (ii) if it uses traditional table-based routing, it still requires large numbers of table entries which restricts path selection [6]; and (iii) it aggregates PoT policies per flow, impacting the scalability of the PoT solution. An alternative is to replace table-based routing by Segment Routing [16] in order to allow path selection, but, since it represents the path as a list of nodes and updates this list on each hop, it depends on the implementation of variable size headers and does not keep unchangeable the route identifier.

As detailed in the next section, PoT-PolKA solves the described problems by proposing a design for in-situ PoT with Shamir scheme that uses PolKA SSR and a feasible implementation of the modulo operation with Mersenne numbers. Table I shows a design comparison, showing that PoT-PolKA has capabilities that are unique compared to existing works: **i)** encoded path information; **ii)** constant route identifier; **iii)** aggregation per path not per flow; and **iv)** *routeID* overhead for the data plane, instead of path tracing or variable size header.

III. PoT-PolKA proposal

This section introduces PoT-PolKA proposal, presenting a comparative overview with the existing IETF RFC in section III-A. In subsection III-B, a step-by-step design is described with its implementation in P4 code.

A. Overview

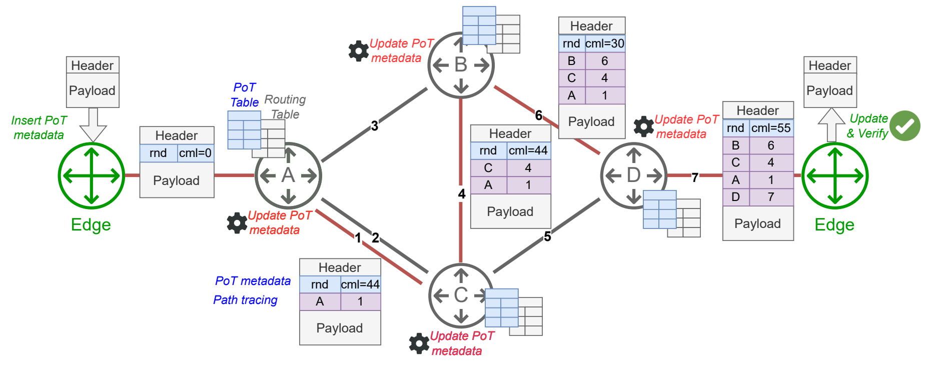
Fig. 5 (a) presents the overview of IETF RFC Draft [5] based on the Shamir secret sharing [8] method. We provide more information about the Shamir mechanism in Appendix V-A. The system parameters are provisioned by the controller and header metadata is updated at every hop. At the egress node, the collected meta-data allows to reconstruction of the secret for path verification. Thus, PoT metadata ($rnd, cml = 0$) is inserted into the packet header at the edge. In node A, the PoT table is checked in order to update the PoT metadata and its respective routing table to forward the packet to the output link. The packet in transit has its POT metadata updated with its path tracing ($A, link_1$). Then in node B, the process is repeated until the egress edge node, updating the PoT metadata ($rnd, cml = 44$) and stacking its path tracing ($C, link_4$). The path verification is performed at the egress edge that checks whether the collected meta-data matches with the cumulative PoT metadata ($rnd, cml = 55$). It is important to note that each core node stores tables for routing and PoT parameters.

On the other hand, PoT-PolKA design offers a path-binding property by using a SSR approach based on PolKA [6], which explores the Residue Number System (RNS) and Chinese Remainder Theorem (CRT). PolKA encodes the path in a *routeID* Fig. 5 (b) using the RNS in contrast to the conventional table-based, which depends on routing tables, or list-based representations, which transports the path information "in clear" inside the packet header. Then, PolKA core nodes use this encoded route label to discover the output ports, by performing the forwarding as an arithmetic operation: the remainder of division of the *routeID* by its own *nodeID*. However, if an attacker is able to mirror a port, then path deviation attacks can occur. Thus, assuming she/he gets access to *nodeID* and *portID*, despite PolKA first security barrier, this will not guarantee the forwarding consistency, so that a PoT is required to protect against packet path deviation.

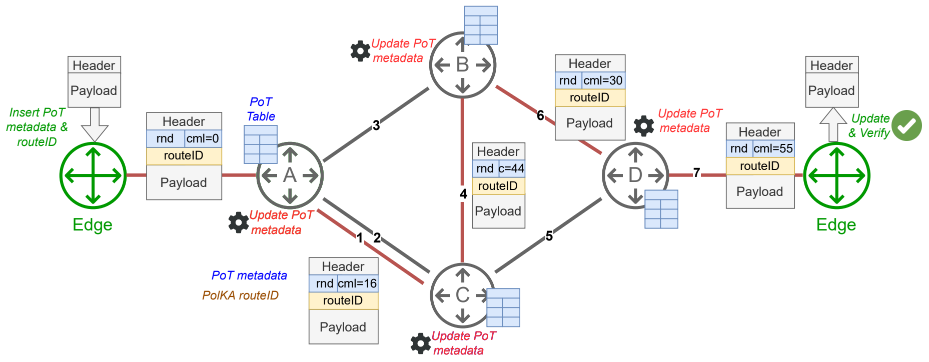
PoT-PolKA solves problems in traditional PoT solutions by proposing a design based on Shamir's secret sharing scheme and PolKA. It uses programmable P4 switches and a small packet digest (PoT metadata) to ensure the path-binding property. The PolKA *routeID* acts as a key to check the nodes along the defined path and update the PoT metadata at each hop. The egress node verifies if the packet traversed all the specified nodes without the need for storing path tracing information, as the *routeID* uniquely identifies the network path.

Limitations of the approach and additional security analysis can be seen in [5], with proofs of robustness for *inter-node* and *inter-packets* passive attacks. However, the current solution does not mitigate *replay* and *pre-replay* attacks, requiring a mitigation mechanism to be included in future versions.

Authorized licensed use limited to: TRINITY COLLEGE LIBRARY DUBLIN. Downloaded on July 18,2023 at 09:55:52 UTC from IEEE Xplore. Restrictions apply.



(a) PoT proposal by IETF Draft



(b) General Design of PoT-PolKA

Figure 5: Comparison of proposals: IETF Draft [5] and PoT-PolKA

B. Design

Fig. 6 explains the PoT-PolKA step-by-step. The design is structured in three main steps: the computations at control plane, configuration of the data plane, and, finally, the path verification in the egress edge node.

1) Step1 : PoT Computation at the control plane: The PoT-PolKA algorithm leverages on Shamir's Secret Sharing scheme [8]. The principle is to define a single secret, represented by a polynomial, that is associated with a particular set of $n+1$ nodes that typically represent the path to be verified. Thus, a polynomial of degree n is selected as a secret at the control plane. A set of $n+1$ points of this polynomial will be assigned to $n+1$ nodes. Each of these $n+1$ points is called a "share" of the secret.

For the **edge nodes**, a private polynomial ($Poly_1$) is selected (see fig. 6), and its zero degree coefficient gives the secret (e.g. $Secret = 10$). When a path is selected to be verified, for each pair of edge nodes, $(x, Poly_1(x) \ mod \ M)$ must be computed. For example, see the green box ($P1 = 16, M = 31, B = 5, S = 10$).

The novelty in comparison to the IETF RFC Draft [5] is the introduction of (Mersenne) numbers for a feasible implementation in programmable switches. Since the modulo operation is not natively supported in P4 language, we propose to use Mersenne numbers ($M = (2^B) - 1$,($B = 5, M = 31$)) to efficiently compute the modulo operation. Thus, a modulo with a Mersenne number can be calculated by a $shift$ and an and operator (&). Suppose a $K \ mod \ P$ operation, this computation can be done with elementary operations whether P is a mersenne number and K is smaller than $(1 << (2 * B)) - 1$, where B is the power of two of the mersenne number $((2^B) - 1)$. The algorithm written in P4 is shown in Code 1.

For the **core nodes**, a public polynomial ($Poly_2$), as there are $n+1$ nodes in the path, the polynomials ($Poly_1$,($Poly_2$)) should be of degree n, is chosen and the verifier egress node can reconstruct the n degree polynomial ($Poly_3$) only when all the points are correctly retrieved. The shares of the secret are the points on ($Poly_1$) chosen for a path length of 4 nodes. For example, let $x_0 = 1$, $x_1 = 3$, $x_2 = 5$, $x_3 = 7$. $Poly_1(1) = 16$

$= (x_0, y_0) = (1, \ 16)$

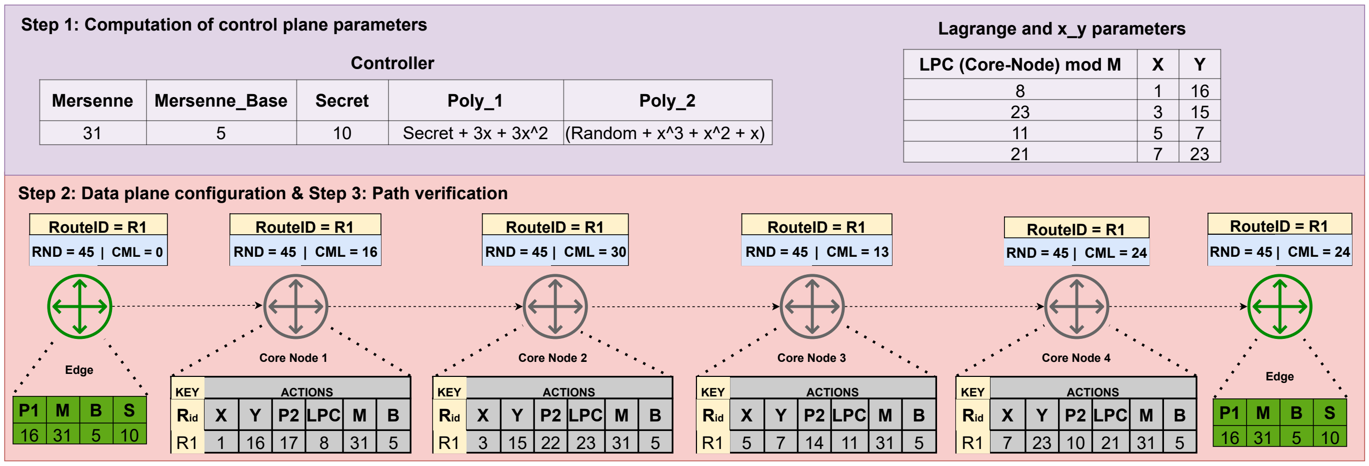
$Poly_1(3) = 15 = (x_1, y_1) = (3, \ 15)$

$Poly_1(5) = 7 = (x_2, y_2) = (5, \ 7)$

$Poly_1(7) = 23 = (x_3, y_3) = (7, \ 23)$

Lagrange polynomial interpolation is used for secret reconstruction to a given set of points on the curve [5]. The Lagrange Polynomial Constants ($LPC's$) [15] are computed by the Controller and communicated to the nodes. Since the points are $x_0 = 1$, $x_1 = 3$, $x_2 = 5$, $x_3 = 7$ in the example, ($LPC's$) can be computed as follows:

Authorized licensed use limited to: TRINITY COLLEGE LIBRARY DUBLIN. Downloaded on July 18,2023 at 09:55:52 UTC from IEEE Xplore. Restrictions apply.



Mersenne	Mersenne_Base	Secret	Poly_1	Poly_2
31	5	10	Secret + 3x + 3x^2	(Random + x^3 + x^2 + x)

LPC (Core-Node) mod M	X	Y
8	1	16
23	3	15
11	5	7
21	7	23

Edge:

P1	M	B	S
16	31	5	10

Core Node 1:

KEY	ACTIONS					
Rid	X	Y	P2	LPC	M	B
R1	1	16	17	8	31	5

Core Node 2:

KEY	ACTIONS					
Rid	X	Y	P2	LPC	M	B
R1	3	15	22	23	31	5

Core Node 3:

KEY	ACTIONS					
Rid	X	Y	P2	LPC	M	B
R1	5	7	14	11	31	5

Core Node 4:

KEY	ACTIONS					
Rid	X	Y	P2	LPC	M	B
R1	7	23	10	21	31	5

Edge:

P1	M	B	S
16	31	5	10

Figure 6: PoT-PolKA design step-by-step

$$LPC(x_0) = \frac{0-x_1}{x_0-x_1} * \frac{0-x_2}{x_0-x_2} * \frac{0-x_3}{x_0-x_3} = \frac{105}{48}\ mod\ 31 = 8$$

$$LPC(x_1) = \frac{0-x_0}{x_1-x_0} * \frac{0-x_2}{x_1-x_2} * \frac{0-x_3}{x_1-x_3} = \frac{35}{16}\ mod\ 31 = 23$$

$$LPC(x_2) = \frac{0-x_0}{x_2-x_0} * \frac{0-x_1}{x_2-x_1} * \frac{0-x_3}{x_2-x_3} = -\frac{21}{16}\ mod\ 31 = 11$$

$$LPC(x_3) = \frac{0-x_0}{x_3-x_0} * \frac{0-x_1}{x_3-x_1} * \frac{0-x_2}{x_3-x_2} = \frac{45}{48}\ mod\ 31 = 21$$

2) Step 2 : Data plane configuration:: In this stage, the parameters are assigned to a PoT table at the nodes. According to PolKA routing [6], the *routeID* is the key (e.g., $routeID$ = R1 = 10979360238159862843) needed to perform the actions in the table. Also, the *nodeID*s are generated and associated to these nodes in the path ($Core_{Node}(1)$ = 65579, $Core_{Node}(2)$ = 65581, $Core_{Node}(3)$ = 65593, $Core_{Node}(4)$ = 65599)[1].

It is worth noting that each parameter is kept secret by individual nodes (i.e. precisely the points on $Poly_1$, the share of $Poly_2$, LPC, M, B). Only the constant coefficient (RND) of $Poly_2$ is public, whereas x value and non-constant coefficient of $Poly_2$ are secret.

On the edge (green table), they receive the information about the secret ($Secret$) and the fixed polynomial ($Poly_1$). The core nodes receive respectively the pair (X,Y) and the LPC of the node, and the polynomial ($Poly_2$). There are some conditions to choose the polynomials: Assuming that K_1 is the degree of $poly_1$, and K_2 is the degree of $poly_2$ with N nodes in the core, we need $K_1 < N$ and $K_2 < N$. Thus, as we use polynomials of minimum degree equal 2, the number of core nodes must be at least 3. In Figure 7, $poly_1$ has degree 2 and $poly_2$ has degree 3, so the minimum number of core nodes must be 4. Finally, the Mersenne M with B is assigned to all nodes.

Data plane computation: In operation, each packet carries its PoT metadata with a random value (RND), generated by the edge, and a cumulative of secret (CML) that is initially zero. The CML is updated by every core node by computing the current CML with the Equation 1, which is implemented in the P4 language (as detailed in Code 1):

$$CML = (CML + (Poly_1(X) + Poly_2(X)) * LPC)\ mod\ M \quad (1)$$

3) Step 3: Path verification: In the verifier node, the verification is made by comparing if the CML in the packet header and the $VERIFY$ value are equal (Equation 2):

$$VERIFY = (S + RND)\ mod\ M \quad (2)$$

```
action calc_cml(){
    meta.new_cml = (meta.y + meta.poly2) * meta.lpc;
    meta.new_cml = (meta.new_cml & meta.mersenne) +
         (meta.new_cml >> meta.mersenne_b);
    if (meta.new_cml > meta.mersenne){
        meta.new_cml = meta.new_cml - meta.mersenne;
    }
    meta.new_cml = hdr.potPolka.cml + meta.new_cml;
}
apply { // PoT-PolKA pipeline
    if (hdr.potPolka.isValid()){
        // Calculate egress port using PolKA SR
        srcRoute_nhop();
        // Table lookup to initialize PoT parameters
        pot_param.apply();
        // Calculate and update CML
        calc_cml();
        hdr.potPolka.cml = meta.new_cml;
        // Set egress port
        standard_metadata.egress_spec = meta.port;
    }else{drop();}
}
```

Code 1: PoT-PolKA Data Plane Computation in P4 Code

As can be seen in Fig. 6, the RND remains fixed during the path, but the CML is computed hop by hop. So, in the egress edge, the PoT applies the equation 2. Given that the $VERIFY$ = 24 computed in the edge is equal to CML = 24 in the packet header, the path verification is confirmed.

[1]PolKA project with github implementation and examples can be found at: https://nerds-ufes.github.io/polka/

Authorized licensed use limited to: TRINITY COLLEGE LIBRARY DUBLIN. Downloaded on July 18,2023 at 09:55:52 UTC from IEEE Xplore. Restrictions apply.

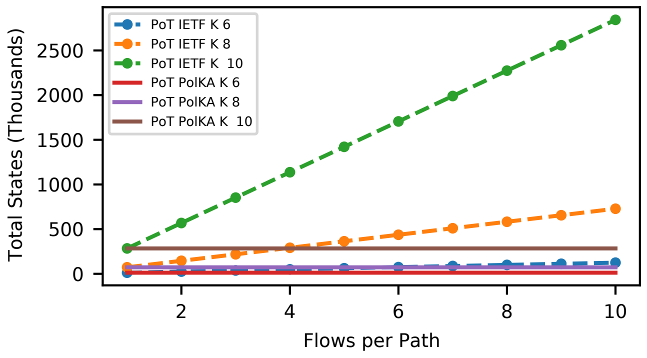


Figure 7: Number of states for Fat-Tree topologies.

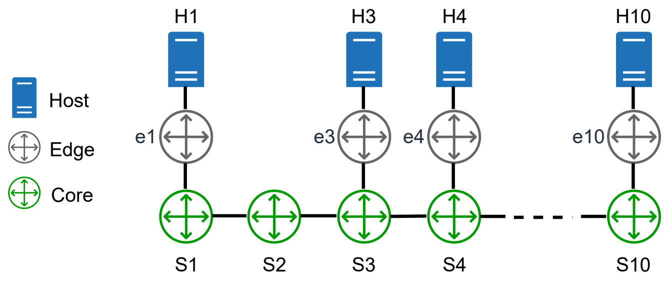


Figure 8: Linear Fabric Topolology

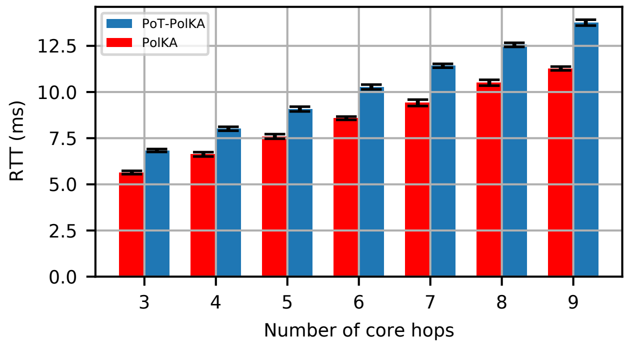


Figure 9: RTT comparison between PolKA and PoT-PolKA

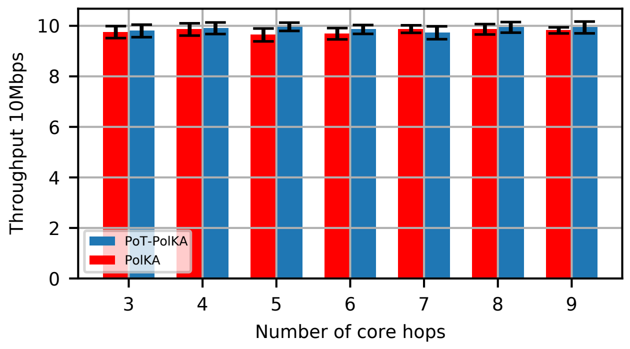


Figure 10: Throughput between PolKA and PoT-PolKA

IV. Evaluation

The evaluation methodology has two components. The first assesses scalability by comparing the network state reduction achieved by PoT-PolKA to the IETF RFC draft [5] for different fat-tree topology sizes and workloads (flows per path). The second involves conducting experiments on the Mininet emulation platform to measure the added latency of PoT-PolKA in comparison to PolKA, as the number of hops along the path varies. The deviation of traffic in the third scenario violates the security policy and undermines the reliable delivery of data within the network.

A. Scalability analysis

For evaluation purposes, we assume that each rule (i.e. network state) is a flow entry for exact matching at the PoT table. So, for the IETF RFC draft, we have $N = flows\_per\_path * number\_of\_paths * path\_length * 2$. For PoT-PolKA, as it aggregates multiple flows that cross a path avoiding to store per-flow state on routers, then the number of network states is $N = number\_of\_paths * path\_length * 2$.

Fig. 7 presents a network state requirements comparison between PoT-PolKA and the PoT proposal of the IETF RFC draft. We use fat-tree topologies for different $K = 6, 8, 10$ under a variable workload (flows per path) from 1 to 10. The pod path lengths were calculated for all combinations of nodes, either for *intra* or *inter* pod. As can be seen, the heavier is the workload per path, the greater is the reduction achieved by PolKA on the total number of states. For example, for a fat-tree with $K = 10$ and a workload of 6 flows per path, the reduction achieves 83,3% and 90% for 10 flows per path.

B. Experiments and Proof-of-Concept

To evaluate the main functionalities of PoT-PolKA, we developed a prototype in the software switch *bmv2 simple_switch* with the v1model architecture as the target. The scenario of Fig. 8 shows a linear fabric topology with edge and core nodes emulated in Mininet. The main objective is to compare the overhead of our PoT-PolKA proposal with the pure PolKA source routing approach (with no PoT mechanism), as the number of hops increases in the core network (e.g., from 3 hops for path H1 $\rightarrow$ H3 to 9 for path H1 $\rightarrow$ H10). The physical setup consists of a server Dell PowerEdge T430, with an Intel Xeon E5-2620 v3 2.40GHz processor and 64GB of RAM. We ran experiments within an Ubuntu 18.04.6 LTS. To build our emulated environment, we used Mininet 2.6 with a P4 compiler and *bmv2* 1.15.0.

As shown in Fig. 9, the latency grows linearly with the increase on the number of hops. Comparing the PolKA vs. PoT-PolKA, we observe a small increase on latency by PoT-PolKA (around 4% when the path length is longer than 6). Throughput is essentially the same for both (Fig. 10), although it is just a comparative value because the link rates were limited to 10 Mbps, due to *bmv2 simple_switch* processing capacity in the emulation.

Authorized licensed use limited to: TRINITY COLLEGE LIBRARY DUBLIN. Downloaded on July 18,2023 at 09:55:52 UTC from IEEE Xplore. Restrictions apply.

C. Protection to path deviation

In Fig. 11, a network attacker is able to deviate the traffic, violating the security policy. By using the prototype of PoT-PolKA in the Mininet emulation, two flows are created at *src* host: (i) a green line flow crossing $S1 \rightarrow S2 \rightarrow S3 \rightarrow S4$ (from T=0s to T=40s); and (ii) a red line flow, deviated from the intended path by the attacker, going over $S1 \rightarrow S5 \rightarrow S4$ (from T=20s to T=40s).

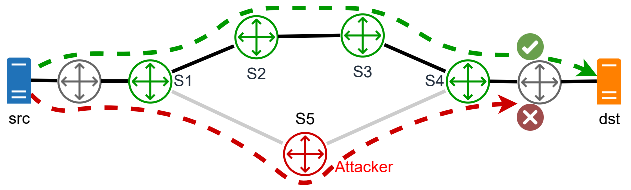


Figure 11: Path deviation attack

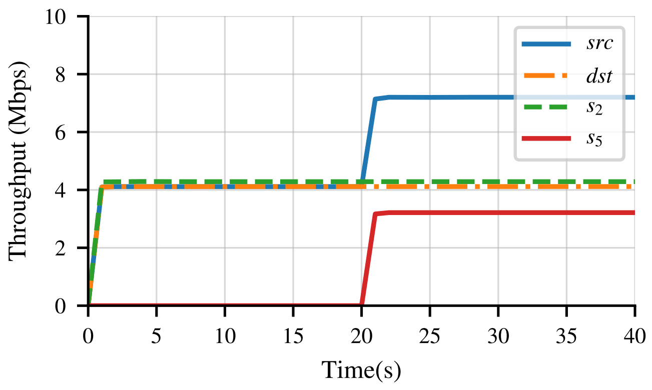


Figure 12: Throughput with PoT-PolKA protection

As shown in Fig. 12, at T=20s, a new flow of 4Mbps is initialized and the aggregated flows at *src* (blue line) increase from 4Mbps to 7Mbps. However, this new flow was deviated which leads to the red line at 3Mbps, whereas the orange line representing the *dst* host remains at 4Mbps. Since the egress edge node applied the PoT-PolKA verification, it drops 3Mbps of the aggregated flows, demonstrating the PoT-PolKA protection against the path deviation attack.

V. Conclusion

This paper introduces a novel in-situ PoT design for programmable networks that leverages PolKA source routing [6] for strict path selection and a modified version of PoT IETF RFC draft [5] for path verification. The design integrates seamlessly with P4 programmable switches, resulting in a scalable and efficient implementation. The proposed solution is validated through experiments performed using a software switch implementation and the Mininet emulation platform, evaluating metrics such as RTT and throughput.

The proposed design extends the existing IETF RFC drafts, by conceptually decoupling the routing/forwarding mechanisms and the PoT implementation. This is enabled by a fixed *routeID* that represents a unique path in the administrative domain and is used as a key for the PoT lookup table to support the path verification. Thus, we can aggregate PoT policies by path with a reduction on the number of network states, which can achieve up to 50 percent for fat-tree topologies with different workloads (flows/path).

We envision as future work to devote efforts to deploy our approach at P4 Tofino programmable switches [17], to include path enforcement and validation functionalities, and to extend [18] for security multipath routing.

Appendix I
Mathematical Background

A. Shamir's Secret Sharing

Shamir's Secret Sharing Scheme (SSSS) is a well-known method for securing secrets in a distributed manner. The basic idea behind SSSS is to divide a secret into multiple parts, called shares, which are distributed among different individuals. The secret can only be reconstructed and unlocked if a sufficient number of shares are combined. SSSS is based on polynomial interpolation over finite fields, which provides a secure and efficient method for dividing and reconstructing the secret.

SSSS has been widely adopted in various fields, including cryptography, data security, and network security. It is particularly useful in scenarios where a secret must be protected, but also needs to be shared among multiple parties.

Shamir's $(t, n) - threshold$ Secret Sharing Scheme can be divided by two main parts, share distribution and secret reconstruction. First in the share distribution phase, we create and distribute a limited number(n) of shares. After, in the secret reconstruction phase, we need at least the threshold number(t) shares to reconstruct the secret. Besides that, is necessary to select a prime number(p) to define the finite field $\mathbb{F}_p$.

In share distribution, the dealer randomly selects t coefficients $(a_i, ..., a_{t-1})$ with a uniform distribution . Then the dealer must construct the polynomial which SSSS is based, the polynomial is constructed based in equation 3, so it must have a degree $t-1$. The secret (S) to hide is contained in the polynomial and must be less than p $(p > S)$. A mod operation is needed because the main principle is to use a polynomial over finite fields.

$$\begin{aligned} f(x) &= S + a_1 x^1 + a_2 x^2 + ... + a_{t-1} a^{t-1} \; mod \; p \\ f(x) &= S + \sum_{i=1}^{t-1} a_i x^i \; mod \; p \end{aligned} \tag{3}$$

Before defining the polynomial, $(x, f(x) \; mod \; p)$ pairs need to be generated to distribute over the parts. The x for each part can be randomly selected (they must be different) or use a ordered selection based on the number of parts $(1, 2, 3, ..., n)$.

To reconstruct the secret at least (t) pairs must be used, with this pairs of points Lagrange Interpolation Formula [15] are used to obtain the original polynomial. The original

Authorized licensed use limited to: TRINITY COLLEGE LIBRARY DUBLIN. Downloaded on July 18,2023 at 09:55:52 UTC from IEEE Xplore. Restrictions apply.

polynomial has the secret as the zero degree, thus to recover the secret just calculate the function at zero $f(0) = S$. The Lagrange interpolation formula can be defined as in equation 4, where $s_{ik} = f(x)\ mod\ p$ of the shares.

$$f(x) = \sum_{k=1}^{t} s_{ik} \prod_{j=1, j\neq k}^{t} \frac{x - i_j}{i_k - i_j}\ mod\ p \tag{4}$$

B. Cumulative Shamir's Secret Sharing

As can be seen in section V-A, all the shares must be together at the same time to be able to reconstruct the secret, see equation 4. In this section, we discuss how to change S to be the secret step-by-step, i.e., save the information about the share (x,y).

The two main changes are in the share distribution and in reconstruction. The dealer has to give an additional information to the shares. The additional information is called LPC_i, LPC_i is the lagrange basis polynomial of Lagrange Interpolation Formula [15] at zero once the secret is $f(0) = S$. The dealer has the information about all the system, this guarantee he can see all shares to distribute the LPC_i along with all pairs.

$$LPC_i = \prod_{m=0, m\neq i}^{k} \frac{0 - x_m}{x_i - x_m}\ mod\ p \tag{5}$$

This way the secret (S) can be reconstructed cumulativel. Equation 6 shows that once s_{ik} and LPC_i is contained on the share, and the sum can be done step-by-step.

$$S = \sum_{k=1}^{t} s_{ik} * LPC_i \tag{6}$$

VI. Acknowledgments

Financial support from Brazilian agencies: CNPq, CAPES, FAPESP/MCTI/CGI.br (PORVIR-5G 20/05182-3, and SAWI 20/05174-0), FAPES (94/2017, 281/2019, 515/2021, 284/2021, 06/2022, 1026/2022, 941/2022). CNPq fellows Dr. Martinello 306225/2020-4. This work also received funds from the 2021 Google Research Scholar Award and the 2022 Intel Fast Forward Initiative.

References

[1] K. Bu *et al.*, "Unveiling the mystery of internet packet forwarding: A survey of network path validation," *ACM Comput. Surv.*, vol. 53, no. 5, sep 2020.

[2] B. Trammell, "Current open questions in path-aware networking," IRTF, RFC 9217, Mar 2022. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc9217

[3] R. Mijumbi *et al.*, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[4] P. Quinn, U. Elzur, and C. Pignataro, "Network service header (NSH)," IETF, RFC 8300, Jan 2018. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8300

[5] F. Brockners *et al.*, "Proof of transit," IETF, Internet-Draft draft-ietf-sfcproof-of-transit-08, Oct. 2020. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-brockners-proof-of-transit

[6] C. Dominicini *et al.*, "Polka: Polynomial key-based architecture for source routing in network fabrics," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2020, pp. 326–334.

[7] F. Brockners *et al.*, "In-situ OAM Deployment," IETF, Internet-Draft draft-ietf-ippm-ioam-deployment-01, Apr. 2022. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-ippm-ioam-deployment/01/

[8] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, p. 612–613, nov 1979.

[9] T. H.-J. Kim *et al.*, "Lightweight source authentication and path validation," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, p. 271–282.

[10] J. Naous *et al.*, "Verifying and enforcing network paths with icing," in *Proceedings of the 7th COnference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '11. New York, NY, USA: ACM, 2011.

[11] F. Zhang *et al.*, "Mechanized network origin and path authenticity proofs," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: ACM, 2014, p. 346–357.

[12] H. Cai *et al.*, "Source authentication and path validation in networks using orthogonal sequences," in *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, 2016, pp. 1–10.

[13] K.-J. Assmus, E.F., "Designs, codes and cryptography," *Springer Open*, 1996.

[14] B. Wu *et al.*, "Enabling efficient source and path verification via probabilistic packet marking," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, 2018, pp. 1–10.

[15] D. Quadling, "Lagrange's interpolation formula," *The Mathematical Gazette*, vol. 50, no. 374, pp. 372–375, 1966.

[16] C. Filsfils *et al.*, "The segment routing architecture," in *IEEE Global Communications Conference (GLOBECOM)*, 2015, pp. 1–6.

[17] C. Dominicini *et al.*, "Deploying polka source routing in p4 switches : (invited paper)," in *2021 International Conference on Optical Network Design and Modeling (ONDM)*, 2021, pp. 1–3.

[18] R. S. Guimarães *et al.*, "M-polka: Multipath polynomial key-based source routing for reliable communications," *IEEE Transactions on Network and Service Management*, 2022.

Authorized licensed use limited to: TRINITY COLLEGE LIBRARY DUBLIN. Downloaded on July 18,2023 at 09:55:52 UTC from IEEE Xplore. Restrictions apply.