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

Implementação e Teste de uma Verificação de Rota Baseada em Amostragem para o Protocolo de Roteamento em Origem PolKA

Henrique Coutinho Layber

Orientadores: Roberta Lima Gomes, Magnos Martinello

Universidade Federal do Espírito Santo

20 de Março de 2025

Contexto

Em sistemas de Roteamento em Origem (SR), normalmente implementados em Redes Definidas por Software (SDNs)[1] o nó de entrada define a rota que o pacote deve seguir.

Contexto

Em sistemas de Roteamento em Origem (SR), normalmente implementados em Redes Definidas por Software (SDNs)[1] o nó de entrada define a rota que o pacote deve seguir.

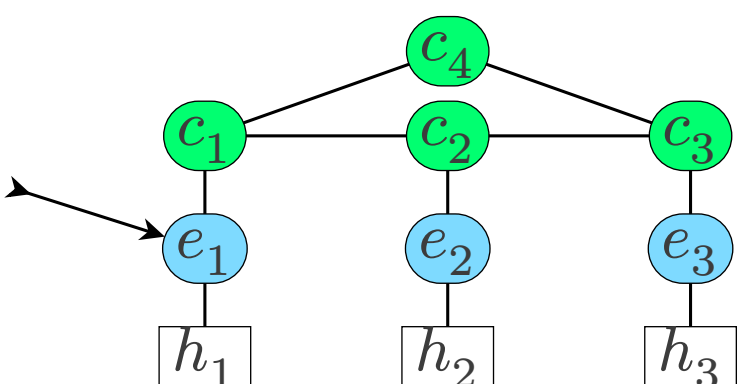
Há uma necessidade de verificar que o pacote seguiu a rota definida pelo nó de entrada, não apenas por questões de segurança, mas também para garantir que a rede esteja funcionando corretamente e configurada adequadamente.

Apresentação do Problema

Assuma um pacote $h_1 \rightarrow h_3$:

1. A rota é definida pelo nó de entrada (*ingress node*)
2.
3.

Define a rota (c_1, c_4, c_3)

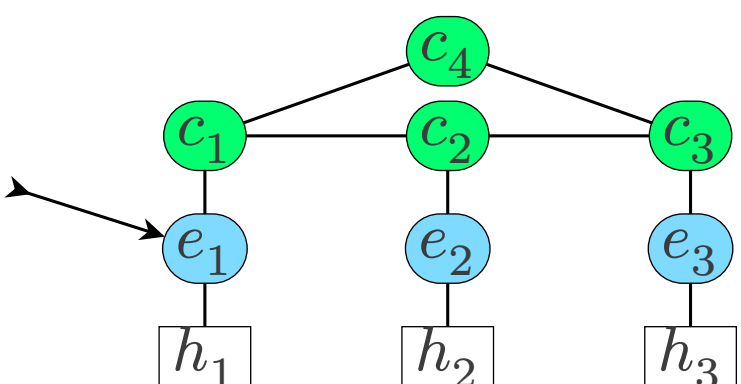


Apresentação do Problema

Assuma um pacote $h_1 \rightarrow h_3$:

1. A rota é definida pelo nó de entrada (*ingress node*)
2. ???
3.

Define a rota (c_1, c_4, c_3)

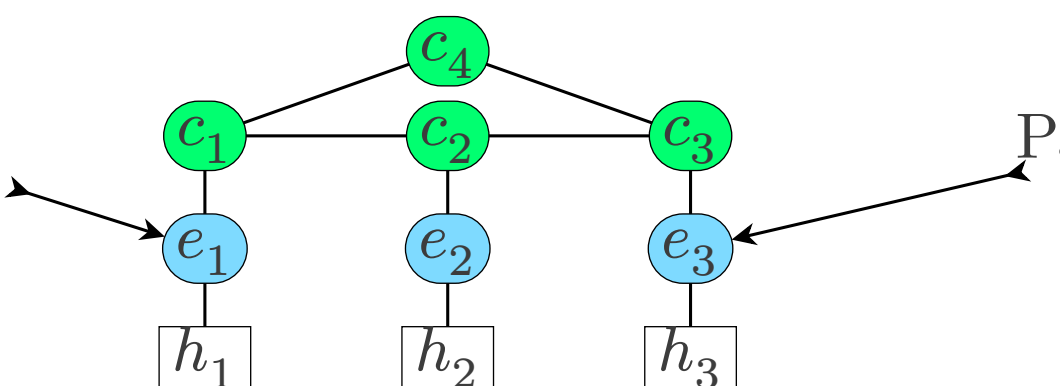


Apresentação do Problema

Assuma um pacote $h_1 \rightarrow h_3$:

1. A rota é definida pelo nó de entrada (*ingress node*)
2. ???
3. O pacote chega ao nó de destino (*egress node*)

Define a rota
(c_1, c_4, c_3)



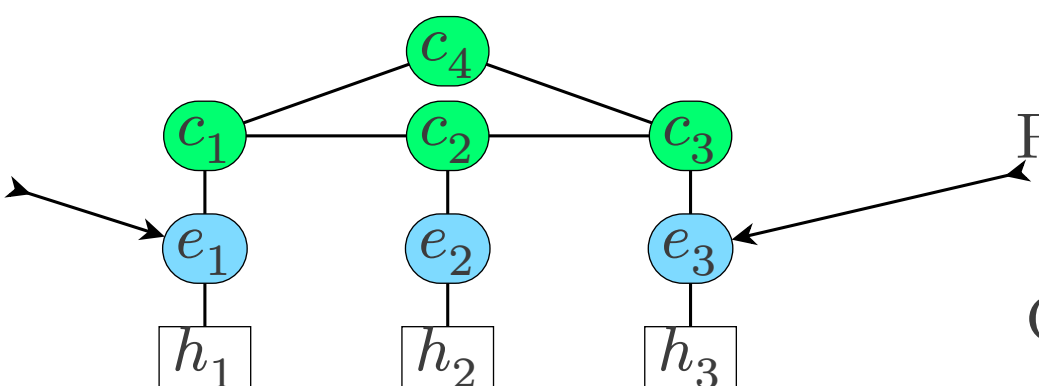
Pacote chega em e_3

Apresentação do Problema

Assuma um pacote $h_1 \rightarrow h_3$:

1. A rota é definida pelo nó de entrada (*ingress node*)
2. ???
3. O pacote chega ao nó de destino (*egress node*)

Define a rota (c_1, c_4, c_3)



Pacote chega em e_3

Como garantir que rota realizada $= (c_1, c_4, c_3)$?

Formalizando o Problema

Rota definida $\underset{i \to e}{P}$ é a sequência de nós s_d: $\left(s_{d_1}, s_{d_2}, ..., s_{d_n}\right)$.

Formalizando o Problema

Rota definida $\underset{i \to e}{P}$ é a sequência de nós s_d: $\left(s_{d_1}, s_{d_2}, ..., s_{d_n}\right)$.
Rota realizada P_j é a sequência de nós s_r coletada $\left(s_{r_1}, s_{r_2}, ..., s_{r_n}\right)$.

Formalizando o Problema

Rota definida $\underset{i \to e}{P}$ é a sequência de nós s_d: $\left(s_{d_1}, s_{d_2}, ..., s_{d_n}\right)$.
Rota realizada P_j é a sequência de nós s_r coletada $\left(s_{r_1}, s_{r_2}, ..., s_{r_n}\right)$.
Precisamos simplesmente verificar se $\underset{i \to e}{P} = P_j$

Formalizando o Problema

Rota definida $\underset{i \to e}{P}$ é a sequência de nós s_d: $\left(s_{d_1}, s_{d_2}, ..., s_{d_n}\right)$.
Rota realizada P_j é a sequência de nós s_r coletada $\left(s_{r_1}, s_{r_2}, ..., s_{r_n}\right)$.
Precisamos simplesmente verificar se $\underset{i \to e}{P} = P_j$

Os nós estão corretos: $(c_1, c_4, c_3) \neq (c_1, c_2, c_3)$

Formalizando o Problema

Rota definida $\underset{i \to e}{P}$ é a sequência de nós s_d: $\left(s_{d_1}, s_{d_2}, ..., s_{d_n}\right)$.
Rota realizada P_j é a sequência de nós s_r coletada $\left(s_{r_1}, s_{r_2}, ..., s_{r_n}\right)$.
Precisamos simplesmente verificar se $\underset{i \to e}{P} = P_j$

Os nós estão corretos: $(c_1, c_4, c_3) \neq (c_1, c_2, c_3)$
Não há nós a mais: $(c_1, c_4, c_3) \neq (c_1, c_4, c_4, c_3)$

Formalizando o Problema

Rota definida $\underset{i \to e}{P}$ é a sequência de nós s_d: $\left(s_{d_1}, s_{d_2}, ..., s_{d_n}\right)$.
Rota realizada P_j é a sequência de nós s_r coletada $\left(s_{r_1}, s_{r_2}, ..., s_{r_n}\right)$.
Precisamos simplesmente verificar se $\underset{i \to e}{P} = P_j$

Os nós estão corretos: $(c_1, c_4, c_3) \neq (c_1, c_2, c_3)$
Não há nós a mais: $(c_1, c_4, c_3) \neq (c_1, c_4, c_4, c_3)$Validade não interessa

Formalizando o Problema

Rota definida $\underset{i \to e}{P}$ é a sequência de nós s_d: $\left(s_{d_1}, s_{d_2}, ..., s_{d_n}\right)$.
Rota realizada P_j é a sequência de nós s_r coletada $\left(s_{r_1}, s_{r_2}, ..., s_{r_n}\right)$.
Precisamos simplesmente verificar se $\underset{i \to e}{P} = P_j$

Os nós estão corretos: $(c_1, c_4, c_3) \neq (c_1, c_2, c_3)$
Não há nós a mais: $(c_1, c_4, c_3) \neq (c_1, c_4, c_4, c_3)$
Não há nós a menos: $(c_1, c_4, c_3) \neq (c_1, c_3)$

Formalizando o Problema

Rota definida $\underset{i \to e}{P}$ é a sequência de nós s_d: $\left(s_{d_1}, s_{d_2}, ..., s_{d_n}\right)$.
Rota realizada P_j é a sequência de nós s_r coletada $\left(s_{r_1}, s_{r_2}, ..., s_{r_n}\right)$.
Precisamos simplesmente verificar se $\underset{i \to e}{P} = P_j$

Os nós estão corretos: $(c_1, c_4, c_3) \neq (c_1, c_2, c_3)$
Não há nós a mais: $(c_1, c_4, c_3) \neq (c_1, c_4, c_4, c_3)$
Não há nós a menos: $(c_1, c_4, c_3) \neq (c_1, c_3)$
Os nós estão na ordem correta: $(c_1, c_4, c_3) \neq (c_4, c_1, c_3)$

Formalizando o Problema

Rota definida $\underset{i \to e}{P}$ é a sequência de nós s_d: $\left(s_{d_1}, s_{d_2}, ..., s_{d_n}\right)$.
Rota realizada P_j é a sequência de nós s_r coletada $\left(s_{r_1}, s_{r_2}, ..., s_{r_n}\right)$.
Precisamos simplesmente verificar se $\underset{i \to e}{P} = P_j$

Os nós estão corretos: $(c_1, c_4, c_3) \neq (c_1, c_2, c_3)$
Não há nós a mais: $(c_1, c_4, c_3) \neq (c_1, c_4, c_4, c_3)$
Não há nós a menos: $(c_1, c_4, c_3) \neq (c_1, c_3)$
Os nós estão na ordem correta: $(c_1, c_4, c_3) \neq (c_4, c_1, c_3)$
Não é preciso conhecer os elementos da sequência!

Formalizando o Problema

Rota definida $\underset{i \to e}{P}$ é a sequência de nós s_d: $\left(s_{d_1}, s_{d_2}, ..., s_{d_n}\right)$.
Rota realizada P_j é a sequência de nós s_r coletada $\left(s_{r_1}, s_{r_2}, ..., s_{r_n}\right)$.
Precisamos simplesmente verificar se $\underset{i \to e}{P} = P_j$

Os nós estão corretos: $(c_1, c_4, c_3) \neq (c_1, c_2, c_3)$
Não há nós a mais: $(c_1, c_4, c_3) \neq (c_1, c_4, c_4, c_3)$
Não há nós a menos: $(c_1, c_4, c_3) \neq (c_1, c_3)$
Os nós estão na ordem correta: $(c_1, c_4, c_3) \neq (c_4, c_1, c_3)$
Não é preciso conhecer os elementos da sequência!

Solução proposta

PathSec[2]: Sugere uma solução baseada em amostragem como uma extensão do PolKA, alguns pacotes se tornam sondas, e recebem os campos adicionais de `timestamp` e `l_hash`. É usada uma composição de funções com um hash criptográfico, baseada no *Hash-based Message Autentication Code* (HMAC).

Solução proposta

PathSec[2]: Sugere uma solução baseada em amostragem como uma extensão do PolKA, alguns pacotes se tornam sondas, e recebem os campos adicionais de `timestamp` e `l_hash`. É usada uma composição de funções com um hash criptográfico, baseada no *Hash-based Message Autentication Code* (HMAC).

Isso define um modelo de multiassinatura. O controlador conhece os segredos dos nós (`nodeID`) e calcula a assinatura final de $\underset{i \to e}{P}$ após o *ingress edge* enviar os metadados, e o *egress edge* captura a assinatura final de P_j.

Solução proposta

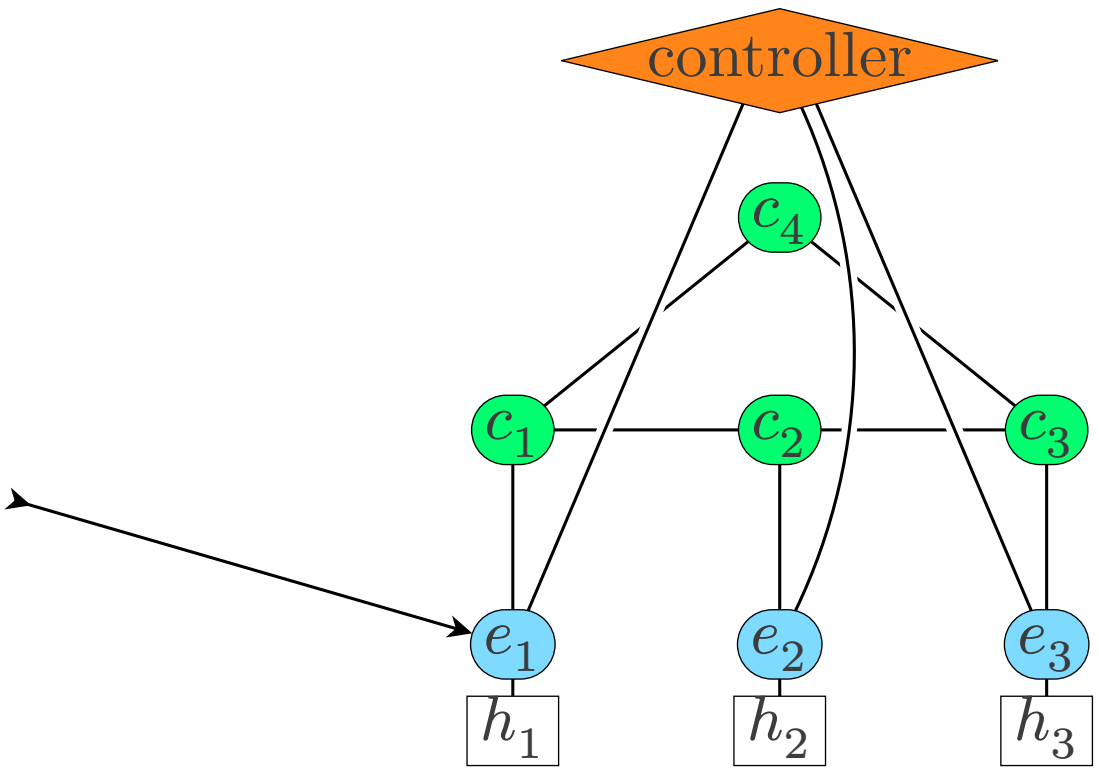
$$\texttt{l\_hash}_0 = \texttt{timestamp}$$

$$\texttt{l\_hash}_n = \mathrm{Hash}\Big(\texttt{nodeID}_{c_n} \parallel \texttt{portID}_n \parallel \texttt{l\_hash}_{n-1}\Big)$$

Define a rota

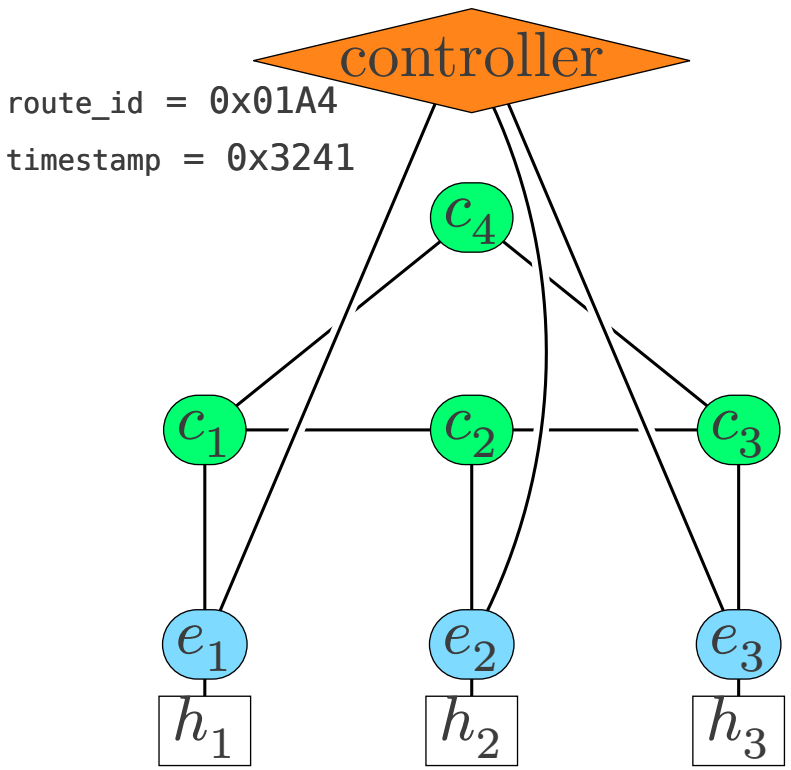
(c_1, c_4, c_3)

```
route_id = 0x01A4
timestamp = 0x3241
```



Valores ilustrativos.

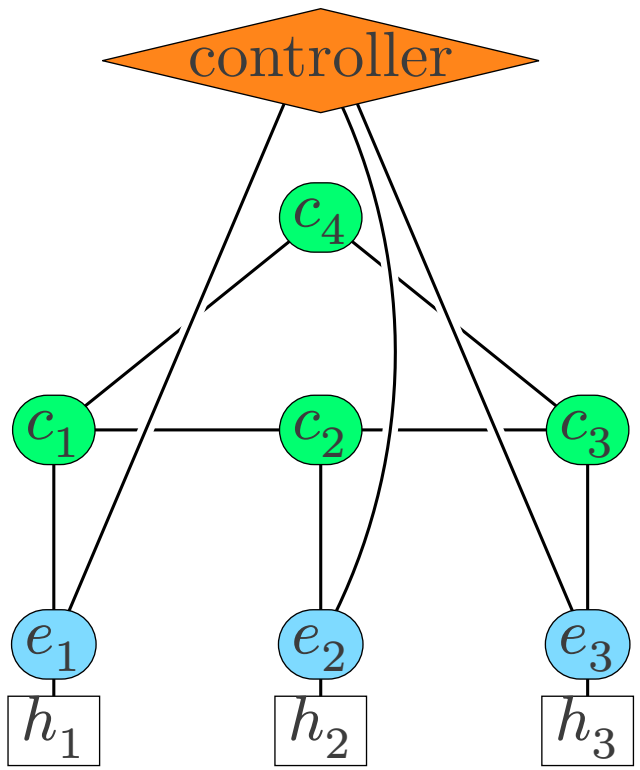
Envia metadados
ao controlador
(e também ao core)



Valores ilustrativos.

$\texttt{l\_hash}_n = \text{Hash}(\texttt{nodeID}, ..., \texttt{l\_hash}_{n-1})$

Enquanto o controlador
calcula a assinatura
de ref., o core calcula
a assinatura nativamente

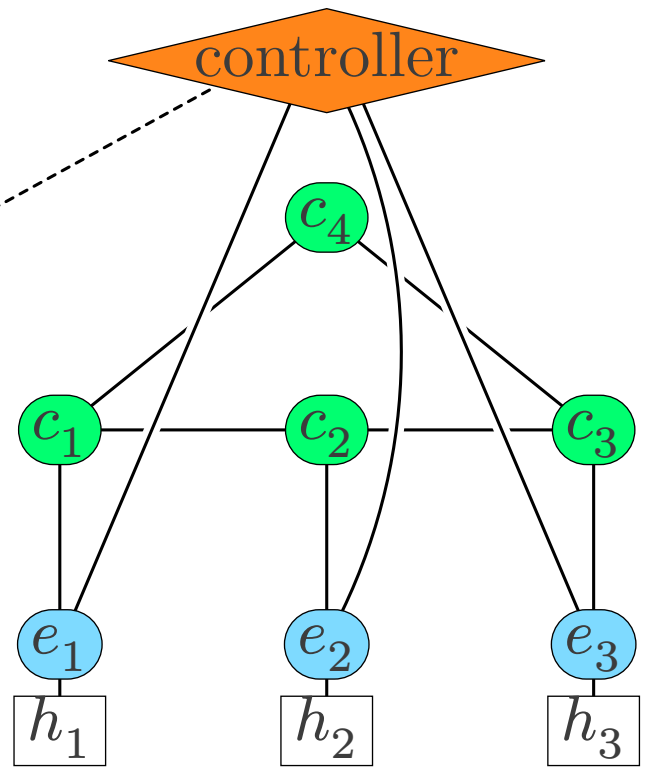


Valores ilustrativos.

$$\texttt{l\_hash}_n = \mathrm{Hash}(\texttt{nodeID}, \texttt{portID}, \texttt{l\_hash}_{n-1})$$

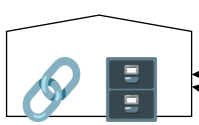
$\texttt{timestamp} = \texttt{0x3241}$ $\texttt{route\_id} = \texttt{0x01A4}$

$$c_1 : H\left(\texttt{0x3241} \| \text{node\_id}_{c_1} \ldots\right) = \texttt{0xc206}$$
$$c_4 : H\left(\texttt{0xc206} \| \text{node\_id}_{c_4} \ldots\right) = \texttt{0xe71f}$$
$$c_3 : H\left(\texttt{0xe71f} \| \text{node\_id}_{c_3} \ldots\right) = \texttt{0xa39b}$$



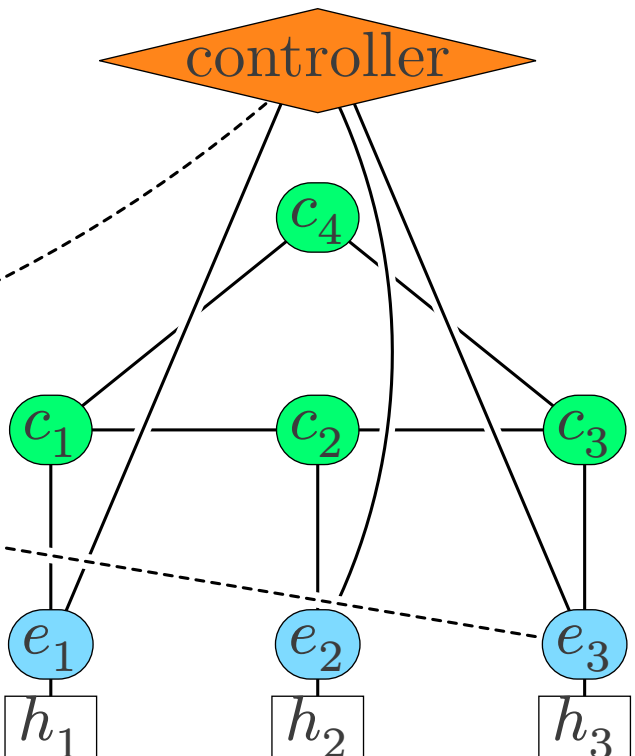
Valores ilustrativos.

Os resultados são enviados ao smart contract



Controller: {`flow_id: 0x3241, l_hash: 0xa39b`}

Egress: {`flow_id: 0x3241, l_hash: 0x3b26`}

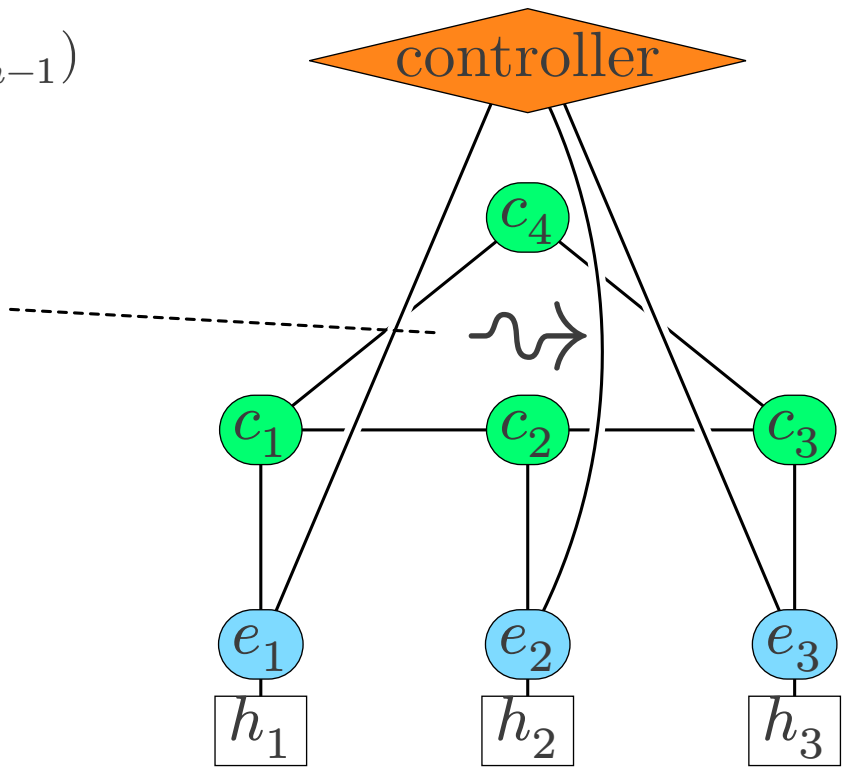


Valores ilustrativos.

$$\texttt{l\_hash}_n = \text{Siphash}(\texttt{nodeID} \parallel \texttt{portID} \parallel \texttt{timestamp} \parallel 0000000, \texttt{l\_hash}_{n-1})$$

$$\texttt{timestamp} = \texttt{0x3241}\ \texttt{route\_id} = \texttt{0x01A4}$$

$$c_1 : H\left(\texttt{0x3241} \| \text{node\_id}_{c_1} \ldots\right) = \texttt{0xc206}$$
$$c_2 : H\left(\texttt{0xc206} \| \text{node\_id}_{c_2} \ldots\right) = \texttt{0xb38f}$$
$$c_3 : H\left(\texttt{0xb38f} \| \text{node\_id}_{c_3} \ldots\right) = \texttt{0x3b26}$$



Valores ilustrativos.

Implementando

Foi extendido de um projeto de PolKA em P4 já existente[3]. O Mininet-wifi[4] foi usado para simular a rede. Scapy[5] foi usado para fazer *sniffing* dos pacotes trafegados.
Foi utilizado um hash criptográfico *Siphash-c-d*, mais especificamente `SipHash-2-4-32`[6], ou seja, 2 *c-rounds* e 4 *d-rounds*, com uma chave de 64bits e entrada de 32bits.

Implementando

Foi extendido de um projeto de PolKA em P4 já existente[3]. O Mininet-wifi[4] foi usado para simular a rede. Scapy[5] foi usado para fazer *sniffing* dos pacotes trafegados.
Foi utilizado um hash criptográfico *Siphash-c-d*, mais especificamente `SipHash-2-4-32`[6], ou seja, 2 *c-rounds* e 4 *d-rounds*, com uma chave de 64bits e entrada de 32bits.

$$\texttt{l\_hash}_n = \texttt{SipHash-2-4-32}\left(\overbrace{\texttt{nodeID}_{c_n}}^{16\text{b}} \| \overbrace{\texttt{portID}_n}^{9\text{b}} \| \overbrace{\texttt{timestamp}}^{32b} \| \overbrace{0000000}^{7b}, \overbrace{\texttt{l\_hash}_{n-1}}^{32b}\right)$$

SipHash, *c-rounds* e *d-rounds*

SipHash é um algoritmo baseado em *Add-Rotate-Xor* (ARX) (BLAKE, ChaCha20, etc), e uma sequência específica de operações ARX é chamado de *SipRound*.

SipHash, *c-rounds* e *d-rounds*

SipHash é um algoritmo baseado em *Add-Rotate-Xor* (ARX) (BLAKE, ChaCha20, etc), e uma sequência específica de operações ARX é chamado de *SipRound*.

c é o número iterações *SipRounds* entre cada ingestão de um novo bloco.

SipHash, *c-rounds* e *d-rounds*

SipHash é um algoritmo baseado em *Add-Rotate-Xor* (ARX) (BLAKE, ChaCha20, etc), e uma sequência específica de operações ARX é chamado de *SipRound*.

c é o número iterações *SipRounds* entre cada ingestão de um novo bloco.

d é o numero iterações *SipRounds* após ingerir todos os blocos, e antes de produzir a saída.

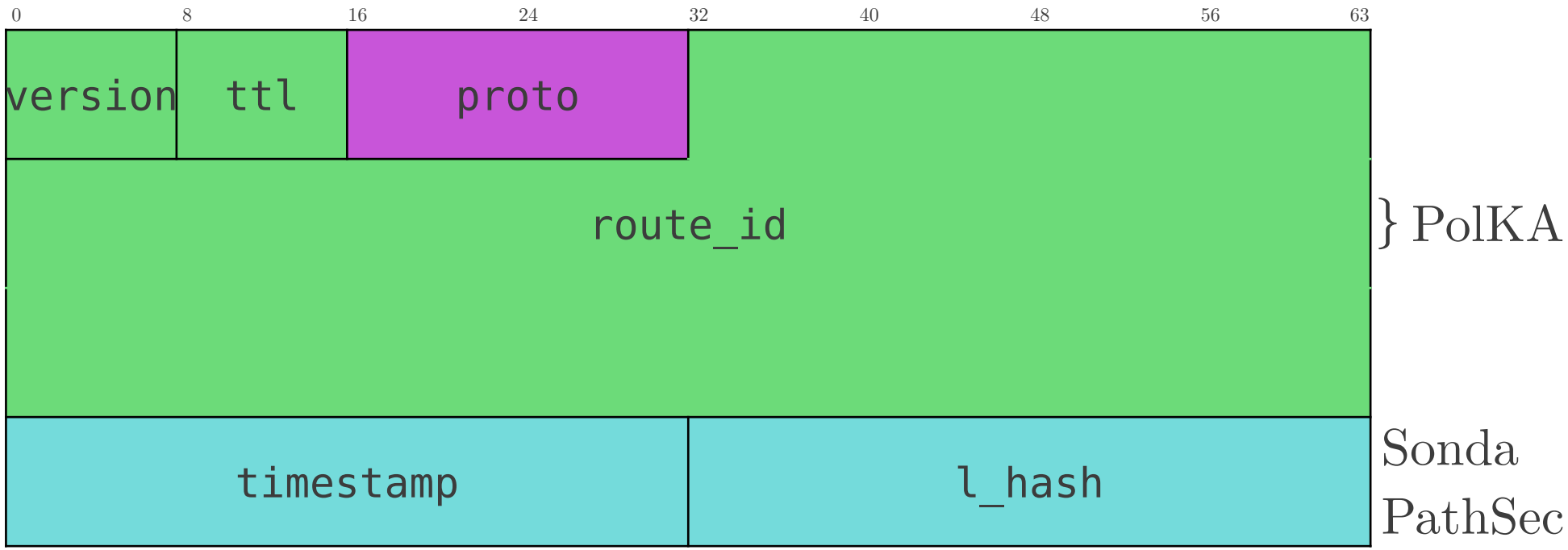
SipHash, *c-rounds* e *d-rounds*

SipHash é um algoritmo baseado em *Add-Rotate-Xor* (ARX) (BLAKE, ChaCha20, etc), e uma sequência específica de operações ARX é chamado de *SipRound*.

c é o número iterações *SipRounds* entre cada ingestão de um novo bloco.

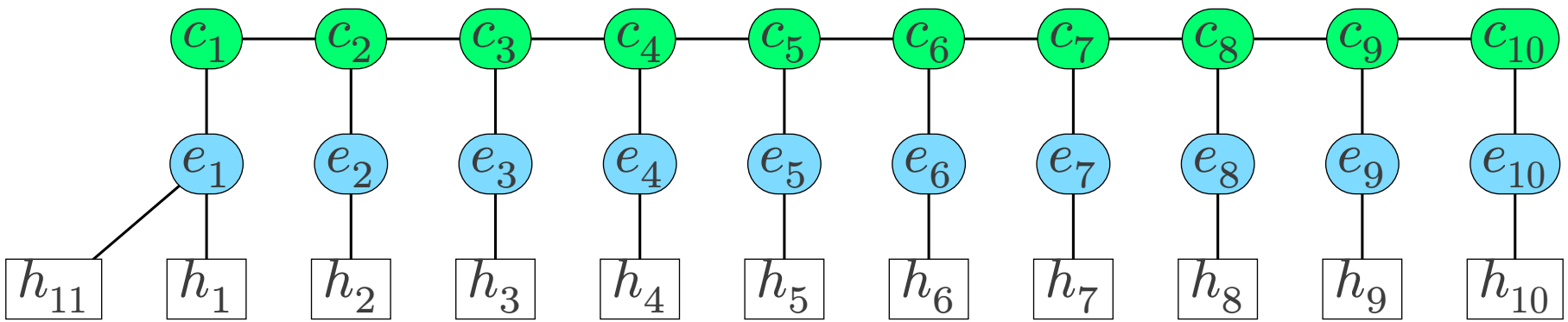
d é o numero iterações *SipRounds* após ingerir todos os blocos, e antes de produzir a saída.

SipHash garante a máxima segurança MAC para $c \geq 2; d \geq 4$[6].


0
8
16
24
32
40
48
56
63
version
ttl
proto
route_id
} PolKA
timestamp
l_hash
Sonda
PathSec

Testes

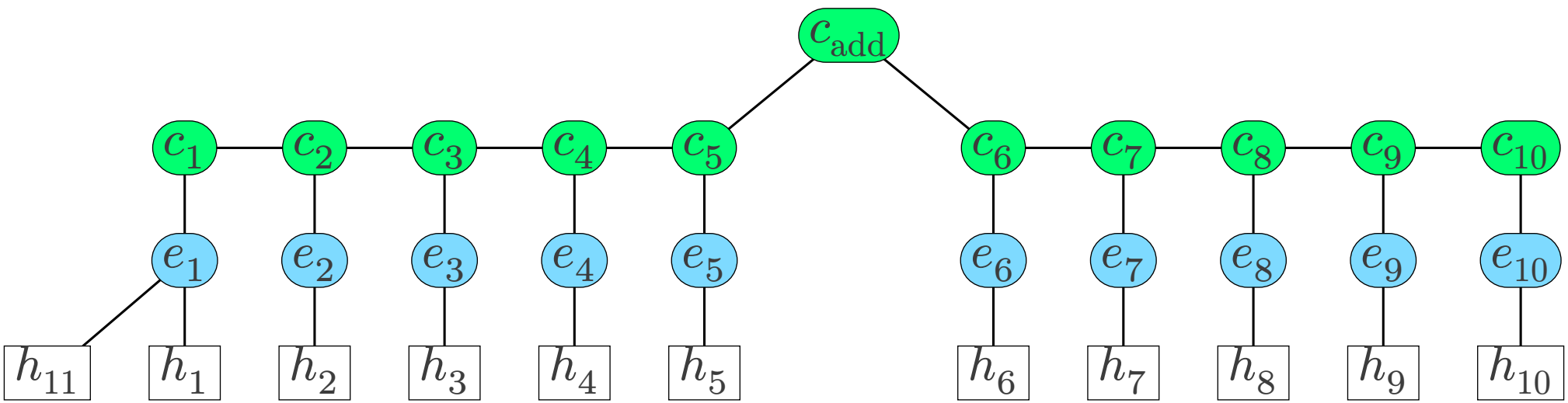
Partindo de uma topologia base



Foram testadas as seguintes condições:

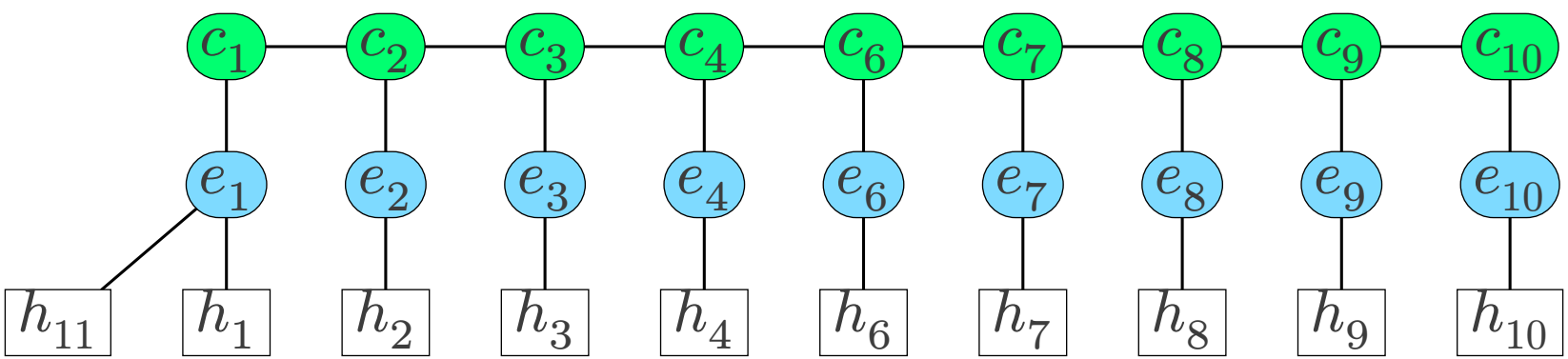
Testes

Adição



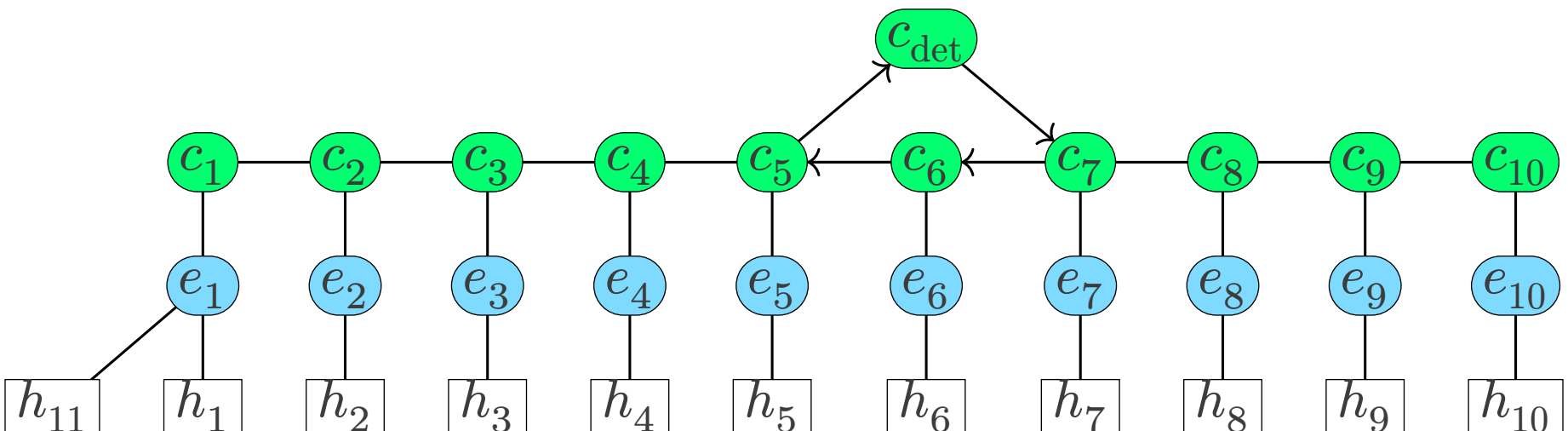
Testes

Salto



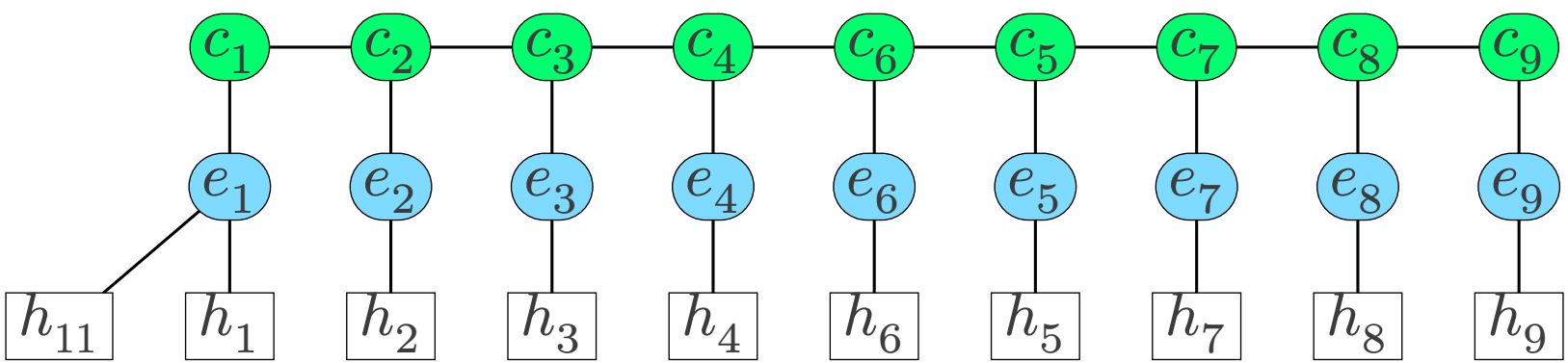
Testes

Desvio



Testes

Fora de ordem



Resultados

Para um pacote $h_1 \rightarrow h_{10}$ foram observados os seguintes resultados de `l_hash` intermediários:

(`route_id` = `0x707b3a1d61d1d0d8b9fc91e442d0360dfc8bba4`)

Referência		Adição		Salto		Desvio		Fora de ordem	
e_1	0x61e8d6e7	e_1	0x61e8d6e7	e_1	0x61e8d6e7	e_1	0x61e8d6e7	e_1	0x61e8d6e7
c_1	0xd25dc935	c_1	0xd25dc935	c_1	0xd25dc935	c_1	0xd25dc935	c_1	0xd25dc935
c_2	0x245b7ac5	c_2	0x245b7ac5	c_2	0x245b7ac5	c_2	0x245b7ac5	c_2	0x245b7ac5
c_3	0xa3b38b83	c_3	0xa3b38b83	c_3	0xa3b38b83	c_3	0xa3b38b83	c_3	0xa3b38b83
c_4	0x26aee736	c_4	0x26aee736	c_4	0x26aee736	c_4	0x26aee736	c_4	0x26aee736
c_5	0xf9b47914	c_5	0xf9b47914			c_5	0xf9b47914	c_6	0x4b5a6c5a
		c_{add}	0x250822a2						
c_6	0x18c6d8d1	c_6	0x20d21d49	c_6	0x4b5a6c5a	c_{det}	0x250822a2	c_5	0xde3862a0
c_7	0xb69b99ec	c_7	0xdd03c4c2	c_7	0x002346d3	c_7	0x40298bb9	c_7	0x648556ec
c_8	0xfe6117f8	c_8	0x292e8608	c_8	0x7ec711aa	c_8	0xe13dcc9b	c_8	0x144e1d1b
c_9	0xc8d9fbde	c_9	0x32419384	c_9	0x5ee32b7b	c_9	0x1bf62c19	c_9	0x9e818f34
c_{10}	0xa6293a25	c_{10}	0xf4bcdf07	c_{10}	0xc973a219	c_{10}	0xdd3a6675	c_{10}	0x6f694bc

Perguntas?

Henrique Coutinho Layber

https://github.com/Henriquelay/PolKA-halfsiphash

Bibliography

[1] S. A. Jyothi, M. Dong, and P. B. Godfrey, "Towards a flexible data center fabric with source routing," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, in SOSR '15. Santa Clara, California: Association for Computing Machinery, 2015. doi: 10.1145/2774993.2775005.

[2] M. Martinello *et al.*, "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), 2024, pp. 1–7. doi: 10.1109/NFV-SDN61811.2024.10807493.

[3] 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. doi: 10.23919/ONDM51796.2021.9492363.

[4] R. R. Fontes, S. Afzal, S. H. B. Brito, M. A. S. Santos, and C. E. Rothenberg, "Mininet-WiFi: Emulating software-defined wireless networks," in *2015 11th International Conference on Network and*

Service Management (CNSM), 2015, pp. 384–389. doi: 10.1109/CNSM.2015.7367387.

[5] B. P. secdev, "Scapy." [Online]. Available: https://scapy.net/

[6] J.-P. Aumasson and D. J. Bernstein, "SipHash: a fast short-input PRF." [Online]. Available: https://eprint.iacr.org/2012/351