Segment Routing over IPv6 (SRv6) Proof of Transit

draft-iannone-spring-srv6-pot-00

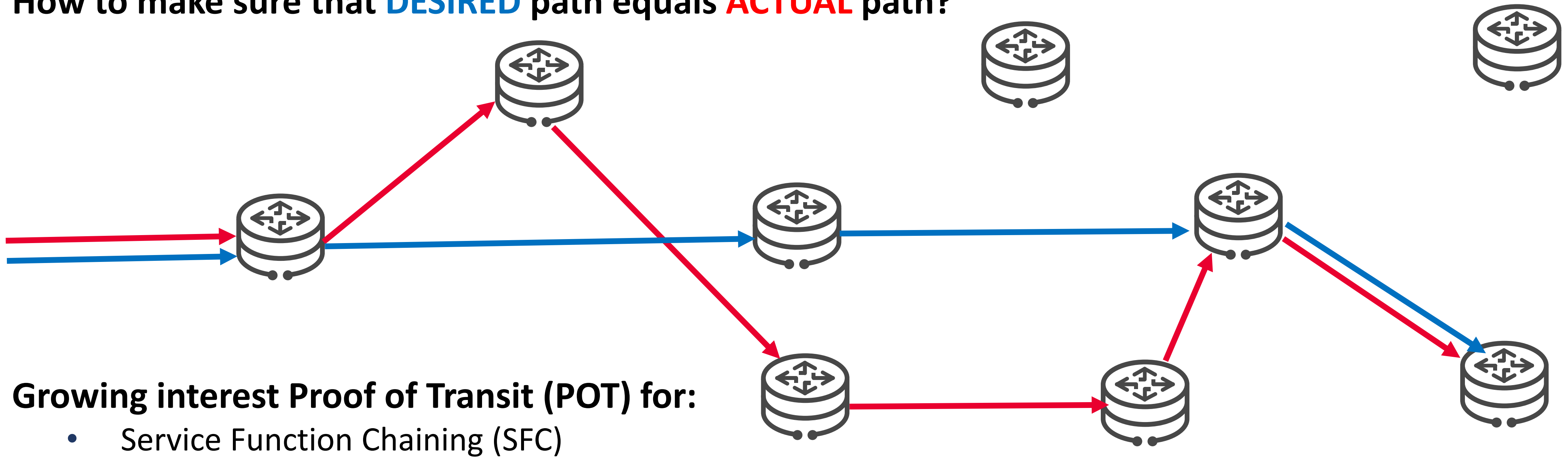
L. Iannone, A. Fressancourt

IETF 119 – Brisbane

Path Verification Problem Statement

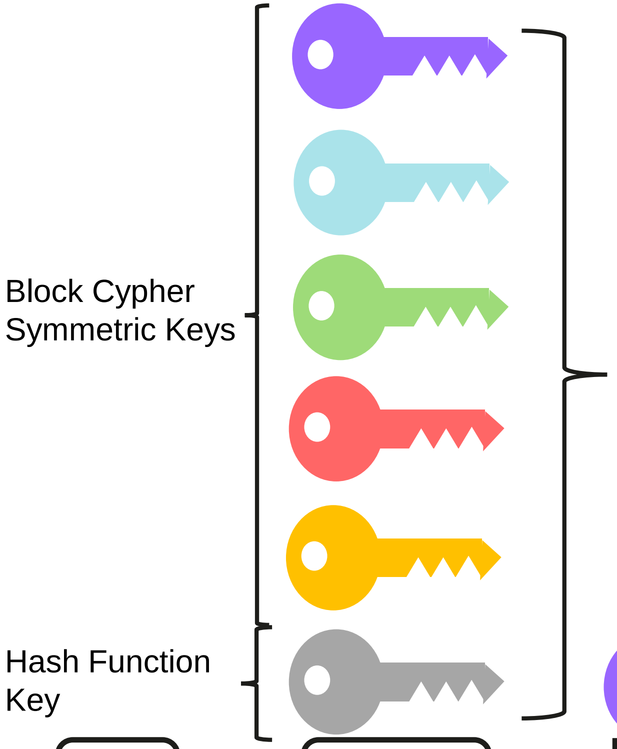
(or should we say segment verification…)

- **Path Validation via Proof of Transit makes sure the path we chose is the actual path the traffic travels**
- **How to make sure that DESIRED path equals ACTUAL path?**

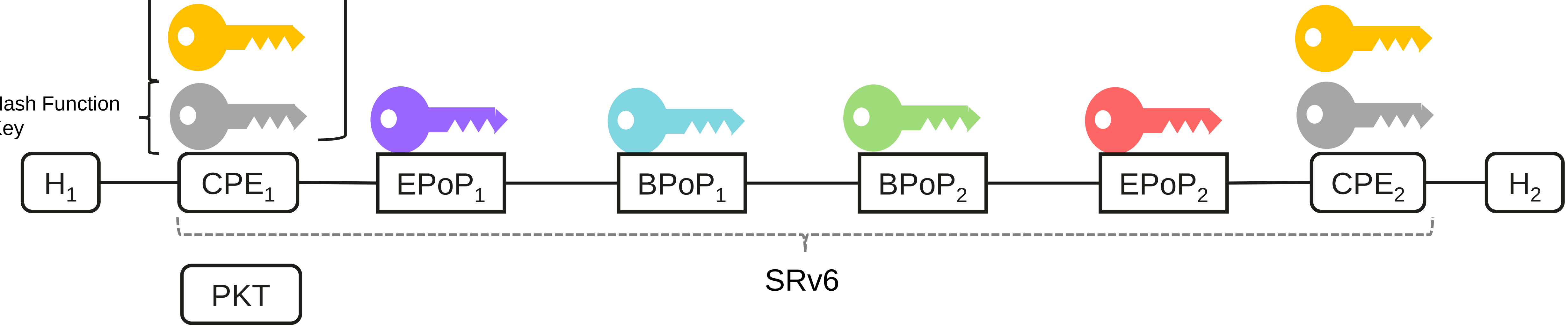


- **Growing interest Proof of Transit (POT) for:**
  - Service Function Chaining (SFC)
  - Workload identity
  - Traffic path compliance
  - uRPF validation
  - Segment Routing Security

POT in Practice: Ingress Node Operations

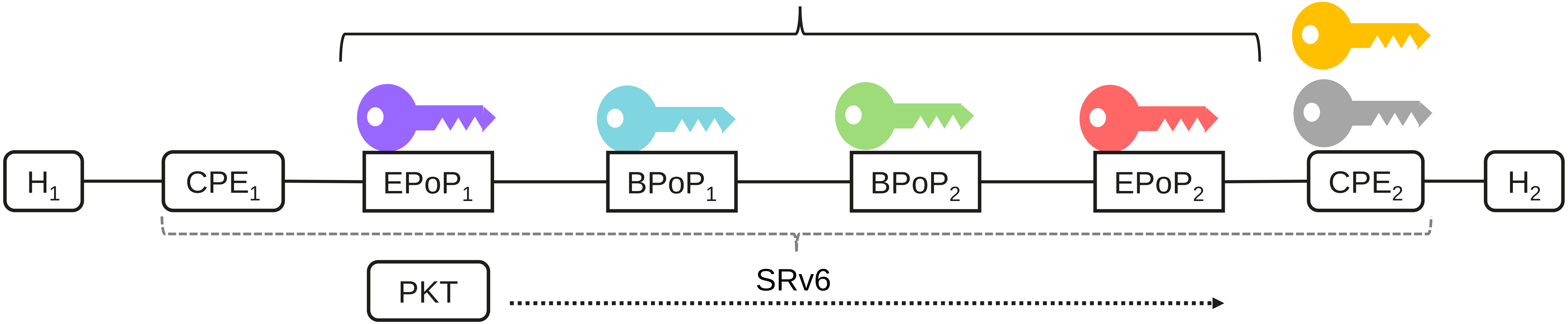


1. Generate HMAC using key pre-shared with egress
   - Algorithm can be SHA1 or SHA256 according to RFC 8754
2. Generate random Nonce
3. Repeatedly encrypt HMAC using Nonce and pre-shared keys starting from last segment



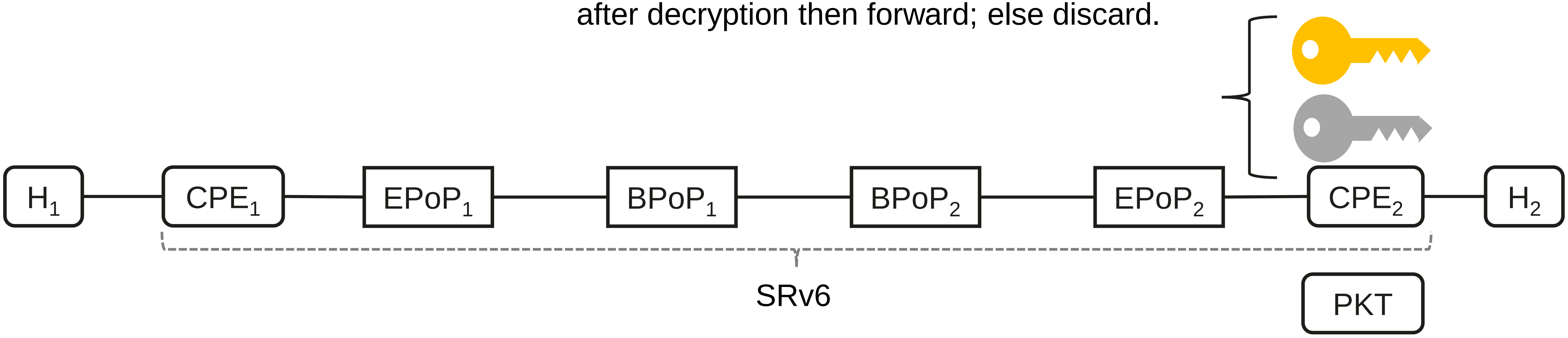
POT in Practice: Middle Segments Operations

- For each packet containing PoT TLV:
  1. Retrieve pre-shared key
  2. Decrypt HMAC once
  3. Forward Packet to next segment



POT in Practice: Last Segment Operations

- For each packet containing PoT TLV:
  1. Retrieve pre-shared key
  2. Decrypt HMAC once
  3. Compute packet HMAC
     - If newly compute HMAC == HMAC obtained from the packet after decryption then forward; else discard.



Recursive Cryptography as POT

1. <u>Packet creation:</u>

Given the set of keys

$$[k_1, k_2, k_3, k_4, k_D, k_{HSD}]$$

Identified by a ***Key Set ID*** and a randomly generated value **N**, the following Path Verification Factor (PVF) is computed:

$$PVF_{HSD} = HMAC_{k_{HSD}}(SID\text{-}list, \ldots.)$$

$$PVF_1 = Enc_{k_1}(Enc_{k_2}(Enc_{k_3}(Enc_{k_4}(Enc_{k_D}(PVF_{HSD}), N),N),N),N),N)$$

Then along with the SRH the following is sent:

$$M = Key\ Set\ ID\ |N|\ PVF_1$$

2. <u>Packet processing</u>

Each intermediate node ***i*** receives a packet

$$M = Key\ Set\ ID\ |\ N\ |\ PVF_{i-1}$$

and computes:

$$PVF_i = Dec_{k_i}(PVF_{i-1,}, N)$$

Then it sends:

$$M = Key\ Set\ ID\ |N|\ PVF_i$$

3. <u>Packet verification:</u>

The destination receives a packet:

$$M = Key\ Set\ ID\ |\ N|\ PVF\_N_{i-1,t}$$

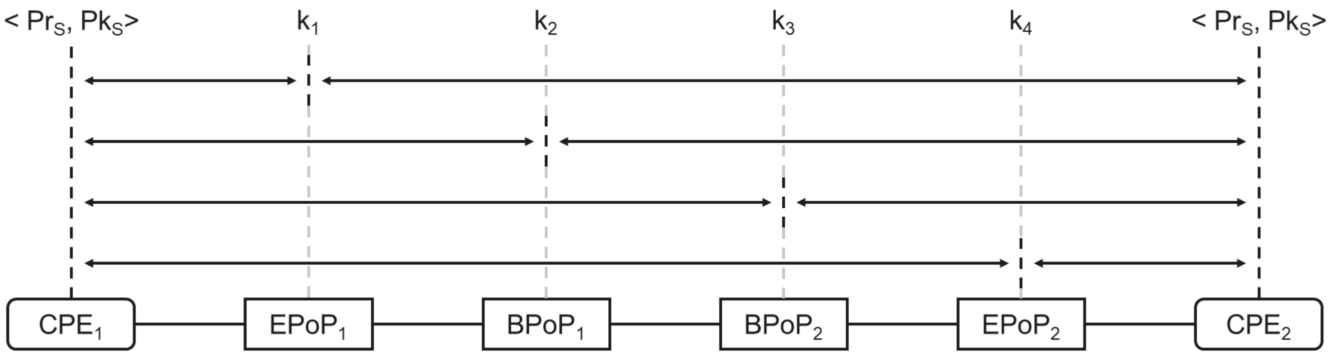
and computes:

$$PVF_{HSD} = Dec_{k_D}(PVF_{D-1}, N)$$

To verify the HMAC, it computes:

$$PVF_{HSD}' = HMAC_{k_{HSD}}(SID\text{-}list, \ldots.)$$

If $PVF_{HSD} = PVF_{HSD}'$, then SID-list and the path has been verified



Proof of Transit Segment Routing Header TLV

<table>
<tr><td colspan="8">0</td><td colspan="8">1</td><td colspan="8">2</td><td colspan="8">3</td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td colspan="8">Type</td><td colspan="8">Length</td><td colspan="8">Reserved (8 bits)</td><td colspan="8">Nonce Length (8 bits)</td></tr>
<tr><td colspan="32">Key Set ID (32 bits)</td></tr>
<tr><td colspan="32">Nonce (variable length)</td></tr>
<tr><td colspan="32">Encrypted HMAC (variable length)</td></tr>
</table>

- Improvements compared to SRv6 HMAC:
  - Sequential encryption and decryption of the HMAC ➔ verification that the routers listed in the SID list have been crossed
- Drawbacks compared to SRv6 HMAC:
  - Intermediate routers are unable to verify the SID list’s integrity.

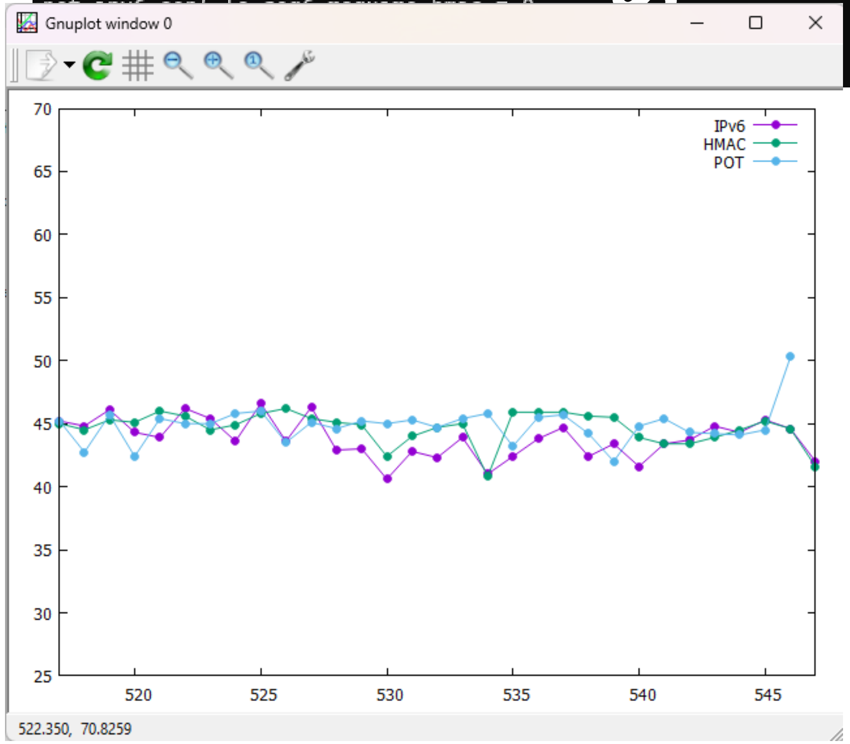
Linux Kernel Implementation

```
#include <linux/types.h>
#include <linux/seg6.h>

#define SEG6_PVAL_FIELD_LEN 32

#define SEG6_PVAL_ALGO_NAME_MAX_LENTH   64
#define SEG6_PVAL_SECRET_MAX_LENTH      64
#define SEG6_HMAC_SECRET_MAX_LENTH      64

/* These should go in
 * include/uapi/linux/seg6.h
```

- WSL Kernel 6.1.21.1
  - Header (TLV) Operations
    - Crypto: AES/XTS
  - Sysctl API
  - CLI Configuration
    - Support in iproute2 (v6.6.0)

```
ggx@DebianNoVPN:~/linux-kernel$ sudo sysctl net.ipv6 |grep seg6
net.ipv6.conf.all.seg6_enabled = 0
net.ipv6.conf.all.seg6_require_hmac = 0
net.ipv6.conf.all.seg6_require_pval = 0
net.ipv6.conf.default.seg6_enabled = 0
net.ipv6.conf.default.seg6_require_hmac = 0
net.ipv6.conf.default.seg6_require_pval = 0
net.ipv6.conf.docker0.seg6_enabled = 0
net.ipv6.conf.docker0.seg6_require_hmac = 0
net.ipv6.conf.docker0.seg6_require_pval = 0
net.ipv6.conf.eth0.seg6_enabled = 0
net.ipv6.conf.eth0.seg6_require_hmac = 0
net.ipv6.conf.eth0.seg6_require_pval = 0
net.ipv6.conf.lo.seg6_enabled = 0
```

Sysctl API




```
ggx@DebianNoVPN:~/linux-kernel/iproute2-main$ sudo ./ip/ip sr pval set 338 aes
Enter secrets for Path Validation (blank to delete):
ggx@DebianNoVPN:~/linux-kernel/iproute2-main$ sudo ./ip/ip sr pval set 666 chacha20
Enter secrets for Path Validation (blank to delete):
ggx@DebianNoVPN:~/linux-kernel/iproute2-main$ sudo ./ip/ip sr pval show
pval: id 338 algo aes secret "thisisatest"
pval: id 666 algo chacha20 secret "thisisasecret"
```

Conclusion

- **Initial evaluation quite promising**
  - **Cost in terms of delay similar to Segment-by-Segment HMAC**
    - (in the specific setup)

- **Happy to discuss more and show you a demo!**