Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Fachbereich Informatik
Department of Computer Science

Bachelor's Thesis

in Computer Science

Evaluating the Performance of IPsec Enhanced ESP vs. Google PSP Security Protocol

by
Maximilian Wagner

Supervisor	Prof. Dr. Hannes Tschofenig
Advisor	Prof. Dr. Michael Rademacher
Submission date	June 5, 2025

Erklärung

Ich versichere an Eides statt, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Ort, Datum	Unterschrift

Contents

List of Figures

Acronyms

AH	Authentication Header 3, 12
CPU	Central Processing Unit 6, 10, 11, 15, 16, 18, 19, 20, 21, 22, 23, 24
CSV	Comma-Seperated Values 18
ECMP	Equal Cost Multi Path 6, 7
EESP	Enhanced Encapsulating Security Payload 1, 6, 7, 8, 9, 10, 11, 12, 14, 15, 23, 24
ESP	Encapsulating Security Payload 1, 2, 3, 4, 6, 7, 8, 10, 12, 15, 18, 19, 20, 21, 23, 24
FID	Flow Identifier 7
ICV	Integrity Check Value 5, 8
IETF	Internet Engineering Task Force 1
IKE	Internet Key Exchange 3, 4, 12, 15, 23
IKEv2	Internet Key Exchange Version 2 2
IP	Internet Protocol 4, 9, 12
IP-TFS	Internet Protocol Traffic Flow Security 8
IPsec	Internet Protocol Security 1, 2, 3, 12, 15, 18, 20, 21, 23, 24
IPsecME	IP Security Maintenance and Extensions Working Group 1
IPv4	Internet Protocol Version 4 8, 9, 13
IPv6	Internet Protocol Version 6 6, 8, 9, 13
IV	Initialization Vector 4, 8, 12, 13
KDF	Key Derivation Function 12, 23
MITM	Man-in-the-Middle 3, 23
NAT	Network Address Translation 7
NIC	Network Interface Card 12, 23, 24
OSI	Open Systems Interconnection 2
PSP	PSP Security Protocol 1, 12, 13, 14, 15, 23, 24
QoS	Quality of Service 6
RAM	Random Access Memory 12, 15
RFC	Request for Comment 1
SA	Security Association 3, 4, 5, 7, 8, 9, 10, 11, 12, 15, 16, 18, 19, 20, 21, 22, 23, 24
SAD	Security Association Database 4
SPI	Security Parameters Index 4, 7, 12

TS Traffic Selector 4

UDP User Datagram Protocol 7

VC Virtualization Cookie 13

VM Virtual Machine 15, 16, 17

VNI Virtual Network Identifier 13

VPN Virtual Private Network 3, 24

WAN Wide Area Network 2

1 Introduction

In the modern landscape of globally interconnected networks, organizations are increasingly distributing their infrastructure across multiple locations and countries. This shift has made it imperative to secure traffic between those locations, as well as into the cloud. Secure network communication has become the backbone of information security as large volumes of data are consistently exchanged. With these large amounts of traffic comes the need for efficient network traffic encryption to ensure no unauthorized third parties get access to the information being transmitted.

The go-to solution to secure transit between networks since before the year two-thousand is Internet Protocol Security (IPsec) with its Encapsulating Security Payload (ESP) protocol. Over time, new protocols have emerged that adapt more effectively to evolving requirements and often offered better performance, higher security or more features tailored to data center and cloud needs. Meanwhile, security protocols are evolving to better fit in these high-speed environments.

Two such protocols are both aiming to become the successor to ESP for general purpose, high performance network security. The new contenders are Google's PSP Security Protocol and the Enhanced Encapsulating Security Payload protocol.

1.1 Motivation

Before the PSP Security Protocol was developed, Google did an analysis on existing industry-standards. Among those standards was Internet Protocol Security (IPsec). They came to the conclusion that IPsec did accomplish some of their goals, but was not able to support their scale because of hardware offload and security requirements [1].

For the amount of connections that they require, the memory usage would be well beyond commodity offload engines' capabilities. IPsec was also lacking support for encryption keys per layer-4 connection, which is one of their security requirements [1]. The IPsec Enhanced Security Payload (ESP) Protocol was at this point (2022) 27 years old [2] with the most recent change made in 2005 [3]. The sort of current massive scale use cases were not considered, as they did not exist in the same form back then.

With newer transport encryption solutions like OpenVPN and Wireguard, there can be made an argument, that ESP was obsoleted. This, as a result, lead to the efforts of overhauling ESP for current use cases in the form of Enhanced ESP (EESP).

At the IP Security Maintenance and Extensions Working Group (IPsecME) meeting during the IETF 122 conference, the EESP draft was raised to a candidate for adoption and was consequently adopted as a working group document [4]. The document is currently an active internet draft and on its way to become an official RFC. As a prospective new RFC, there is a need for EESP to be evaluated and tested against current and past standards, which leads to the goal for this thesis: Evaluating the Performance of IPsec EESP vs. Google PSP Security Protocol.

1.2 Thesis Structure

This thesis is split into five parts. In Chapter 2 will be the necessary knowledge to understand IPsec and the key exchange mechanism. Thereafter, in chapter 3, the new EESP protocol will be explained in detail with all the changes coming from ESP.

Next, in chapter 4.3, the architecture and goals of the Google-developed PSP Security Protocol will be described along with some details for the reference implementation. In the following chapter 5 the experimental setup as well as the testing results can be found.

The evaluation of the results from chapter 5 can be found in chapter 6.1 whereafter in chapter 6.2 future work to be done on the topic will be outlined.

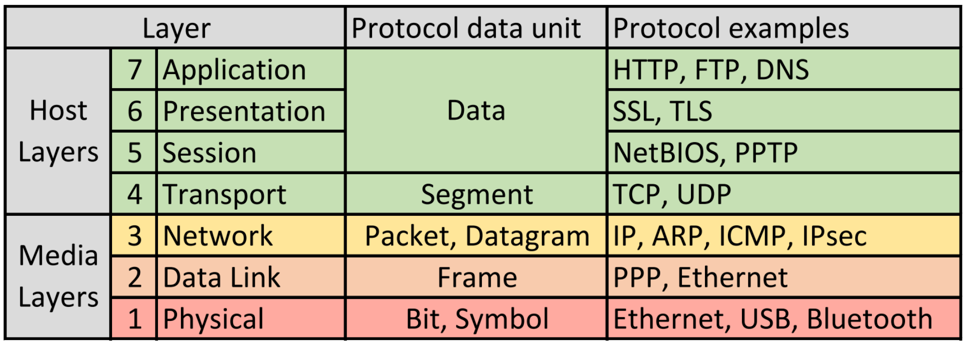
2 Basics

Before getting into the new Enhanced Encapsulating Security Payload protocol, there is some prerequisite knowledge to understand why changes to ESP needed to be made and why it matters.

These prerequisites are the Open Systems Interconnection (OSI) Model, how OSI layer-3 encryption is done and how IPsec is currently used in conjunction with IKEv2 and ESP.

2.1 Open Systems Interconnection (OSI) Model

The Open Systems Interconnection model, published in 1994 by the International Organization for Standardization, provides "[..] a common basis for the coordination of standards development for the purpose of systems interconnection [..]" [5]. The end goal of the OSI model is to allow for connected systems to be able to communicate with each other in different layers of abstraction without having to worry too much about what is underneath.



Layer			Protocol data unit	Protocol examples
Host Layers	7	Application	Data	HTTP, FTP, DNS
	6	Presentation		SSL, TLS
	5	Session		NetBIOS, PPTP
	4	Transport	Segment	TCP, UDP
Media Layers	3	Network	Packet, Datagram	IP, ARP, ICMP, IPsec
	2	Data Link	Frame	PPP, Ethernet
	1	Physical	Bit, Symbol	Ethernet, USB, Bluetooth

Figure 2.1: OSI layers and the corresponding protocols

As can be seen in figure 2.1, IPsec operates on layer-3, the network layer of the OSI model. "The network layer provides the functional and procedural means of transferring packets from one node to another connected in "different networks"" [6]. Solutions like IPsec exist to encrypt those packets, so that no middleman can read its contents.

2.2 Secure Communication on Layer-3

There are three different operation modes for network layer encryption. The different types "Site-to-Site", "Peer-to-Peer" and "Road Warrior" can be seen in figure 2.2. The aforementioned middleman would be somewhere inside the Wide Area Network (WAN) where the packets are routed through.

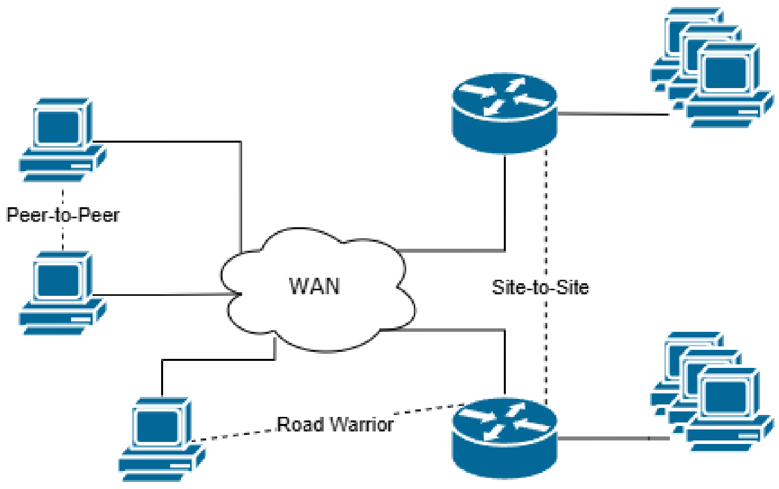


Figure 2.2: Network layer encryption operation modes

A so-called Man-in-the-Middle (MITM) is a third party that intercepts network traffic between two or more peers to gather any useful information contained in the packets. To stay undetected, the captured packets can be forwarded after processing. Without encryption, the content of the packets could be seen by this unauthorized middleman, unbeknownst to the communicating peers.

With these so-called Virtual Private Network (VPN) solutions, there is no dependency on encryption on a per-application basis, although it is a best practice also known under the term layered security.

2.3 Internet Protocol Security (IPsec)

The oldest of the widely used VPN (or more generally layer-3 encryption) solutions is Internet Protocol Security. IPsec is a protocol suite consisting of the two different protocols Authentication Header (AH) and the already mentioned Encapsulating Security Payload (ESP).

Since AH does not provide any encryption, it is out of scope for this thesis. ESP however does provide encryption and is the reference point from whence Enhanced ESP was developed. Thus the inner workings of ESP warrant a closer look.

2.3.1 Internet Key Exchange Version 2 (IKEv2)

But before getting into ESP it is necessary to take a quick look at the Internet Key Exchange (IKE) protocol, primarily to define terminology which is used further on.

"IKE is a component of IPsec used for performing mutual authentication and establishing and maintaining Security Associations (SAs)" [7]. A Security Association is a "set of security information relating to a given network connection or set of connections" [8]. "The combination of a given Security Parameter Index (SPI) and Destination Address uniquely identifies a particular "Security Association"" [8]. Other parameters may also be supported but that is implementation-specific [8].

A Security Association generally includes the following for Authentication Header.

- Authentication algorithm and algorithm mode being used with the IP Authentication Header (required) [8]
- Key(s) used with the authentication algorithm in use with the Authentication Header (required) [8]

The following is generally included for Encapsulating Security Payload.

- Encryption algorithm, algorithm mode, and transform being used with the IP Encapsulating Security Payload (required) [8]
- Key(s) used with the encryption algorithm in use with the Encapsulating Security Payload (required) [8]
- Presence/absence and size of a cryptographic synchronization or initialization vector field for the encryption algorithm (required)
- Authentication algorithm and mode used with the ESP transform, if any is in use (recommended) [8]
- Authentication key(s) used with the authentication algorithm that is part of the ESP transform, if any (recommended) [8]

For both AH and ESP the following is recommended to be included.

- Lifetime of the key or time when key change should occur [8]
- Lifetime of this Security Association [8]

- Source Address(es) of the Security Association, might be a wildcard address if more than one sending system shares the same Security Association with the destination [8]
- Sensitivity level (for example, Secret or Unclassified) of the protected data [8]

On the sender side the user ID and destination address is used to select an appropriate Security Association. On the receiving side a combination of SPI value and destination address is used to select the correct SA [8].

Whether or not traffic needs to encrypted or decrypted is defined via a so-called Traffic Selector (TS). If for a given TS there is no Security Association, it is the job of IKE to negotiate a SA [7].

2.3.2 Encapsulating Security Payload (ESP)

"The IP Encapsulating Security Payload (ESP) is designed to provide integrity, authentication, and confidentiality to IP datagrams" [8]. ESP can also provide an anti-replay service and traffic flow confidentiality which are options selected at the time of Security Association [3].

The packet format of ESP as defined in RFC 4303 can be seen in figure 2.3.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ ----
|               Security Parameters Index (SPI)                 | ^Int.
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |Cov-
|                      Sequence Number                          | |ered
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ | ----
|                    Payload Data (variable)                    | |   ^
~                                                               ~ |   |
|                                                               | |Conf.
+               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |Cov-
|               |     Padding (0-255 bytes)                     | |ered
+-+-+-+-+-+-+-+-+               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |   |
|                               |  Pad Length   | Next Header   | v   v
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ ------
|         Integrity Check Value-ICV   (variable)                |
~                                                               ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2.3: Top-Level Format of an ESP Packet [3]

The Security Parameters Index (SPI) is an arbitrary 32-bit value which is used to map incoming packets to a specific SA. These maps along with their corresponding source and destination IP addresses are stored in the Security Association Database (SAD) where the lookup of the right SA for incoming ESP packets is performed [3].

The Sequence Number is a 32-bit counter which increases by one for each packet sent. Sequence numbers are not shared between SAs. Since the sequence number is not allowed to cycle, a new SA must be negotiated before the transmission of the 2^{32}nd packet [3].

The Payload Data field is variable in length and contains the data of the original IP packet. If an algorithm that requires an Initialization Vector (IV) was selected on SA establishment, this field also contains the IV, typically directly preceding the ciphertext [3].

An optional Padding field with up to 255 bytes is included for encryption algorithms that require the plaintext to be a multiple of a number in length and to ensure that the resulting ciphertext terminates on a 4 byte boundary [3].

”The Integrity Check Value (ICV) is a variable-length field computed over the ESP header, Payload, and ESP trailer fields” [3]. The value is provided by a separate integrity algorithm or a combined mode algorithm that uses an ICV, provided that the integrity service is selected on SA establishment [3].

3 IPsec Enhanced ESP (EESP)

The Enhanced Encapsulating Security Payload protocol builds upon the existing ESP protocol [9]. It is designed to modernize and overcome limitations in the ESP protocol" [9]. EESP can be seen as a modern version of ESP that neither updates nor obsoletes ESP [9].

3.1 Goals

The broad goal of modernizing the ESP protocol can be broken up into a few key points.

- Making the protocol extensible via header options adapted from IPv6 [9]
- Allowing for CPU pinning and Quality of Service (QoS) support via header options [9]
- Optionally exposing parts of the inner packet for middleboxes [9]
- Enabling Equal Cost Multi Path (ECMP) to distribute network load [9]
- Increasing performance in comparison to ESP

Arguably the most important goal is performance, which will be the focus of chapter 3.3.

3.2 Architecture and Changes

To achieve the mentioned goals, several changes had to be made to ESP. First, let's compare the top-level packet format of ESP to EESP.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                          Base Header                          ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                     Peer Header (variable)                    ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Payload Info Header (optional)                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Payload Data (variable)                  |
~                                                               ~
|               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               |          Padding (0-255 bytes)                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~              Integrity Check Value-ICV (variable)             ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.1: Top-Level Format of an EESP Packet [9]

As can be seen in figure 3.1, there have been significant changes in the layout of the packet in the form of semantic divisioning.

3.2.1 Base Header

As shown in figure 3.2, the Base header (like ESP) includes the Security Parameters Index. Unlike ESP, the SPI is preceded by several fields.

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |1|Version| Opt Len |   Flags   |         Session ID            |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                              SPI                              |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

Figure 3.2: Fixed Base Header [9]

The first bit is the ESP-in-UDP compatibility bit [9]. It signals, that encapsulation of ESP in UDP for the purposes of NAT-Traversal is supported [9][10].

The next field is the version field. This is also new coming from ESP, the lack of which made transparent updates to ESP not feasible [9] and necessitated the creation of the EESP protocol.

Next up is the flags field which includes several switches. Firstly the packet format. EESP has two different packet formats, those being the full and the optimized format [9].

Then there is the payload encryption mode. Without this flag, the payload info header is encrypted with the payload and thus private to the communicating peers [9]. Setting this bit (or using the crypt offset) the payload info header will be transmitted in plaintext which is mainly for use in data centers [9].

The last flag is for an absent sequence number. With ESP the field was mandatory and is now optional [3][9]. Not sending the sequence number and thus disregarding the anti-replay feature is mainly for the use case of sharing a SA between multiple senders, which is possible but not recommended [3].

Finally there are three reserved bits for future versions which currently are ignored by the receiver [9].

The next field is the session ID. This field contains additional information that might be used to identify the appropriate SA [9]. The draft for EESP does not specify a fixed use case but suggests to use this field to encode a Sub SA ID (see chapter 3.3) [9].

As with ESP, EESP uses a Security Parameters Index to determine the corresponding Security Association for any given packet. This field was taken over from ESP as-is.

The following base header field 'options' is optional and of variable size as defined in the opt len field [9]. As defined in the draft, there are four initial standard option types. Two of those are for padding the options field by one or more octets [9].

Then there is the EESP Flow Identifier (FID) option. FID options are used to carry characteristic information about the inner packet flow and as such should not change on a per packet basis [9]. One use of such flow information would be for Equal Cost Multi Pathing [9].

The fourth option is the EESP crypt offset. Apart from it being typically used for datacenter use [9], the draft does not specify any reason to use the crypt offset. It could be used for in-band network telemetry and auditing purposes.

3.2.2 Peer Header

The peer header follows the base header and is optional as well as variable in size.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Sequence Number (optional)                 |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          IV (optional)                        |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.3: Peer Header [9]

As can be seen in figure 3.3, the peer header contains the sequence number and Initialization Vector. The sequence number is used for anti-replay protection via a sliding receive window as with ESP [9][3]. This field, now optional, was mandatory in ESP, even when sharing an SA between multiple senders [3]. ESPs optional extended sequence numbers are now the default; EESP only supports the full 64 bit extended sequence number and unlike ESP transmits the full 64 bits and not just the least significant 32 bits [9][3].

The IV is cryptographic synchronization data to initialize the algorithm and in turn increase security [9][11]. Counter mode algorithms like AES-GCM may use the sequence number field instead which saves eight bytes on every packet [9].

3.2.3 Payload Info Header, Payload and Integrity

The payload info header mainly contains the 'next header' and 'pad length' fields as can be seen in figure 3.4. Only when using Internet Protocol Traffic Flow Security (IP-TFS) or when the payload is not a single IPv4 or IPv6 packet, the payload info header is needed [9].

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  0x0  |        Reserved       | Next Header   | Pad Length    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.4: Payload Info Header [9]

The next header field indicates the type of data contained in the payload data field [9]. Pad length specifies the number of bytes between payload data and ICV [9].

Immediately following the payload info header is the payload data field which contains the data of the original packet [9]. This field needs to be padded to a multiple of four bytes to align the Integrity Check Value [9]. Another primary factor that requires padding are block cipher algorithms that need a plaintext that is a multiple of some number of bytes long [9].

Lastly the ICV field contains a variable-length value computed over the EESP header and payload [9] to provide integrity guarantees for the sent packets.

3.2.4 Tunnel and Transport Mode

Considering all sub-headers of the EESP protocol, the full packet format can be assembled as seen in figure 3.5. The full packet format is the default for EESP [9].

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1|Version| Opt Len |   Flags   |        Session ID             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              SPI                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                  Options (variable, optional)                 ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Sequence Number (optional)                  |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         IV (optional)                         |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  0x0  |        Reserved       | Next Header   | Pad Length    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   L4 Payload Data (variable)                  |
~                                                               ~
|               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               |         Padding (0-255 bytes)                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~              Integrity Check Value-ICV (variable)             ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.5: Full EESP packet format [9]

When using tunnel mode, the payload info header is not needed because the payload of the EESP packet contains the original IPv4/IPv6 header [9]. The next header and pad length fields can be derived from the IP header which makes the payload info header contain exclusively redundant information [9]. Thus an optimized packet format was defined which can be seen in figure 3.6.

With establishment of a Security Association, the algorithms, modes and options are fixed and therefore the detailed EESP packet format for the given SA is fixed [9].

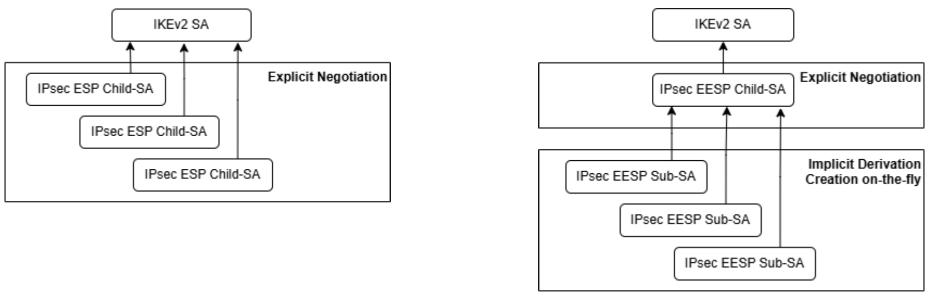
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1|Version| Opt Len |   Flags    |        Session ID            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              SPI                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                   Options (variable, optional)                ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Sequence Number (optional)                 |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          IV (optional)                        |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                     IPv4/IPv6 Header                          ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   L4 Payload Data (variable)                  |
~                                                               ~
|               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               |          Padding (0-255 bytes)                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~              Integrity Check Value-ICV (variable)             ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.6: Optimized EESP packet format [9]

3.3 Multi-core enhancements

Arguably the most important changes to ESP are the performance related ones. Enhanced Encapsulating Security Payload, through the use of sub-SAs, defines the possibility of per-CPU Security Associations. This feature was was introduced to ESP with RFC 9611 which defines the use of multiple child-SAs which are each bound to a specific resource for the same traffic selector [12].

Though RFC 9611 and EESP have the same goal, EESP takes a different approach. The former defines the explicit negotiation of additional child-SAs via the regular exchange used to create the first child-SA [12] whereas the latter transmits the amount of supported child-SAs to be created on-the-fly [9]. The difference in the hierarchy can be seen in figures 3.7a and 3.7b.



(a) ESP SA hierarchy when using RFC 9611

(b) EESP SA hierarchy using the new approach

Figure 3.7: RFC 9611 and EESP SA hierarchy difference

EESP interpretation of RFC 9611 provides a more flexible approach for asymmetric traffic patterns, especially in high-speed environments [9]. Particularly implementations that make use of hardware offload can profit from improved resource utilization and scalability when deriving keys on a per-packet basis, which mitigates the risk of performance degradation on the data-plane caused by a large number of keys [9]. The keys for the sub-SAs are derived using the session ID which for this use case is named sub-SA ID [9].

Performance going from one CPU core to thirty supposedly scales linearly from 2 Gbps to 60 Gbps, according to RFC 9611 [12]. How these results were achieved was not disclosed in the document. The results of this thesis' testing can be seen in chapter 5. An evaluation of the results can be found in chapter 6.1.

4 PSP Security Protocol (PSP)

Google PSP Security Protocol is a layer-3 encryption protocol not unlike ESP. ”PSP uses several of the concepts from IPsec ESP to provide an encryption encapsulation layer on top of IP that is streamlined and custom-built to address the requirements of large-scale data centers” [13].

4.1 Goals

Just like with EESP, there have been changes to address the shortcoming of ESP in the current landscape of computer networks. The Goals of PSP are:

- Enable scaling to a large number of SAs without requiring massive amounts of on-NIC RAM [13]
- Consolidation of ESP and AH into one protocol using different modes of operation [13]
- Streamlining the feature set and encapsulation format for large-scale data centers [13]

4.2 Architecture and Changes

To achieve its goals, PSP takes a hardware first approach, relying on Network Interface Cards (NIC) to handle keys and encryption.

4.2.1 Key Management

The PSP Security Protocol does not utilize IKE like IPsec does, and no exchange protocol was defined in the architecture specification [13]. The key derivation is one big difference to IPsec.

Every NIC has an ephemeral master key which is not shared with its host or any other NIC [13]. On initial handshake between two NICs, the sender requests an encryption key from the receiver [13]. This encryption key is derived from the master key and the SPI using a Key Derivation Function [13]. Another exchange follows with inverted sender and receiver so that both peers have an encryption key for the established Security Association [13].

The sender uses the stored encryption key provided by the receiver to encrypt the packet [13]. The receiver derives the key on a per-packet basis so that the key provided to the sender does not have to be stored locally [13]. This effectively cuts the keys that need to be stored on the NICs in half, compared to ESP.

4.2.2 Tunnel and Transport Mode

The packet format of PSP in transport mode can be seen in figure 4.1. Some fields have the same use in Enhanced ESP. For any fields not explained here, refer to chapter 3.2.

The Header Extension Length (Hdr Ext Len) field is similar to EESPs Option Length field. A non-zero value indicates the length of the optional virtualization cookie and/or other header extension fields [13].

The following two bit wide field is reserved and to be set to zero on transmit [13]. Following the crypt offset, there are two bytes related to packet sampling. The ”S” bit triggers packet sampling at the receiver and the ”D” bit declares this packet to be dropped after sampling [13]. Packet sampling defines the process of analyzing certain packets instead of every packet to collect statistics about traffic flow to reduce the impact on device performance [14].

The version field, unlike EESP, does not define *just* the PSP header version. In this field is encoded, the combination of header version and used algorithm [13]. This results in four of the possible sixteen ”version” numbers to be occupied already [13].

The ”V” bit indicates whether the virtualization cookie is present or not [13]. Another difference to EESP is the Initialization Vector no longer being optional; this field is now mandatory [13].

Following the IV field is the Virtualization Cookie (VC) [13]. This optional field may contain a Virtual Network Identifier (VNI)[1] or other data as defined by implementation [13].

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  |  Hdr Ext Len  | R | Crypt Off |S|D|Version|V|1|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              SPI                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Initialization Vector                     |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Virtualization Cookie (Optional)                |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Extension Headers (variable, optional)             |
~                                                               ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   L4 Payload Data (variable)                  |
~                                                               ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                    Integrity Check Value-ICV                  |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4.1: PSP transport mode packet format

The architecture specification does not explicitly show where these extension headers are supposed to go. It is stated, that the PSP packet format complies with RFC 6564 (Uniform Format for IPv6 Extension Headers) [13]. Therefore, when assuming compliance with RFC 8200 (IPv6 Specification), the extension headers are located right in front of the layer-4 payload, as can be seen in figure 4.1.

The tunnel mode packet format does not differ from the tunnel mode format, apart from the original IPv4/IPv6 header preceding the layer-4 payload [13], as can be seen in figure 4.2.

[1]A Virtual Network Identifier can be used to differentiate virtual network segments [15].

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  |  Hdr Ext Len  | R | Crypt Off |S|D|Version|V|1|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              SPI                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Initialization Vector                     |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Virtualization Cookie (Optional)                |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Extension Headers (variable, optional)             |
~                                                               ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                      IPv4/IPv6 Header                         ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     L4 Payload Data (variable)                |
~                                                               ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                      Integrity Check Value-ICV                |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4.2: PSP tunnel mode packet format

4.3 Implementation

Currently, there are no publically available tools to configure and use PSP. The public Github repository contains proof-of-concept tools that work on network captures and an exemplary implementation for the Linux kernel version 5.15, but there is no provided way to actually use the PSP protocol.

After almost three years since publication of the protocol and with an EESP implementation on the horizon, it is unlikely that such a tool will be published still. The comparison to EESP can therefore not be done empirically, as there is no way to generate any performance metrics. A theoretical comparison will be made in chapter 6.

5 Testing

Performance metrics could not be generated for both Enhanced Encapsulating Security Payload and PSP Security Protocol. There is no widely available and usable implementation of PSP and there is no EESP implementation available yet.

There does exist an implementation of RFC 9611 (Per-Resource Child Security Associations) for IPsec ESP which will likely be included in the next release of strongSwan[1].

Apart from on-the-fly key derivation for sub-SAs, there is no real difference between ESP using RFC 9611 and EESP. Therefore performance should at least be similar, once all Security Associations are created. The results shown in chapter 5.2 were gathered using the per-CPU SA feature of strongSwan as a substitute for EESP. For Google PSP there is no viable substitute from which data can be collected.

5.1 Experimental Setup

The host on which the environment was created, is a baremetal installation[2] of Proxmox. Proxmox is a type 1 hypervisor[3] like VMware ESXi and XenServer.

5.1.1 Virtual Environment

On the host, two VMs were created with 8 vCPUs and 8 Gigabytes of RAM each named ”sun” and ”moon”. The VMs share a virtual network using the virtio-net driver with 8 queues. For every CPU pair between the VMs, there is one network queue, as visualized in figure 5.1. Traffic sent from sun on CPU core 0 is configured to arrive on core 0 on moon and so on.

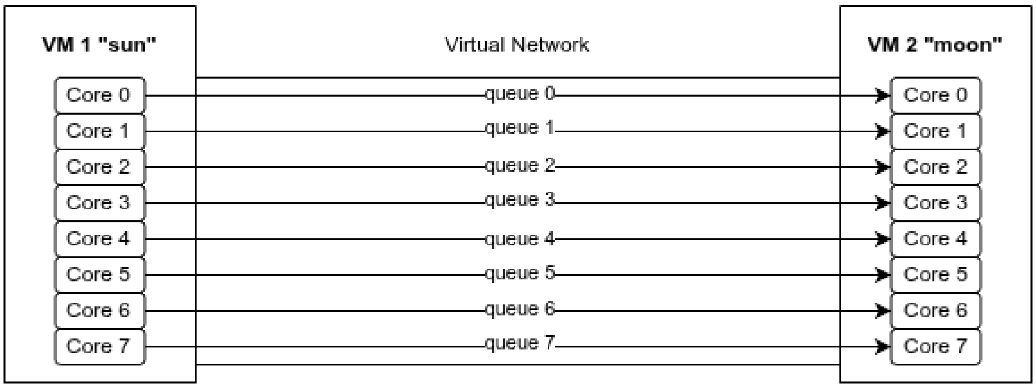


Figure 5.1: Network queue configuration

Both VMs are running the Archlinux-based distribution CachyOS with the latest kernel version 6.15.

5.1.2 Software and Configuration

Currently, there is no other software which has an actively developed per-CPU SA feature, other than strongSwan, which is why this solution to configure IPsec was chosen. There has been no new version of strongSwan since completion of the feature. Therefore the software needs to be compiled from source.

StrongSwan was compiled from the ”master” branch of the official public repository[4] on May 28, 2025, which is when the per-cpu-sas branch was merged. The commands that need to be run on each VM

[1]stronSwan is a comprehensive IKE implementation that can be used to configure IPsec [16]
[2]A baremetal installation means that the operating system to be installed is itself not virtualized.
[3]Type 1 hypervisors are managing virtual environments on bare metal with direct access to hardware as opposed to type 2, which manage those environments using software run on an operating system [17].
[4]The official repository can be found on Github: `https://github.com/strongswan/strongswan`

can be seen in figure 5.2. Running the ”git reset” command sets the working tree to the state used in testing but may be omitted.

```
git clone https://github.com/strongswan/strongswan.git
cd strongswan
git reset 72e3b7dcc887bb471a8c1ffc462b7ac882f653d2
./autogen.sh
./configure --prefix=/usr --sysconfdir=/etc --enable-systemd --enable-swanctl \
   --enable-kernel-netlink --enable-aesni
make -j8
make install
```

Figure 5.2: Commands used to compile strongSwan

StrongSwan is configured to create per-CPU sub-SAs. The connection configuration for VM sun can be seen in figure 5.3. Figure 5.4 depicts the configuration for the same connection on VM moon.

```
# file: /etc/swanctl/swanctl.conf
connections {
   lake {
      local_addrs = 192.168.0.10
      remote_addrs = 192.168.0.20

      local {
         auth = pubkey
         certs = sunCert.pem
         id = sun.strongswan.org
      }
      remote {
         auth = pubkey
         id = moon.strongswan.org
      }
      children {
          water {
            per_cpu_sas = yes
            mode = transport

            start_action = trap
            updown = /usr/local/libexec/ipsec/_updown iptables
            hostaccess = yes
            esp_proposals = aes128-sha256-x25519
         }
      }
      version = 2
      mobike = no
      proposals = aes128-sha256-x25519
   }
}
```

Figure 5.3: IPsec connection configuration for VM sun

```
# file: /etc/swanctl/swanctl.conf
connections {
   lake {
      local_addrs = 192.168.0.20
      remote_addrs = 192.168.0.10

      local {
         auth = pubkey
         certs = moonCert.pem
         id = moon.strongswan.org
      }
      remote {
         auth = pubkey
         id = sun.strongswan.org
      }
      children {
          water {
            per_cpu_sas = yes
            mode = transport

            start_action = trap
            updown = /usr/local/libexec/ipsec/_updown iptables
            hostaccess = yes
            esp_proposals = aes128-sha256-x25519
         }
      }
      version = 2
      mobike = no
      proposals = aes128-sha256-x25519
   }
}
```

Figure 5.4: IPsec connection configuration for VM moon

To use this connection, several certificates are needed. A quickstart guide on how to create these can be found in the strongSwan documentation[5].

The following files need to be present in the /etc/swanctl/ directory on VM sun:

- x509ca/strongswanCert.pem
- x509/sunCert.pem
- rsa/sunKey.pem

In the same directory on VM moon, the following files need to be present:

- x509ca/strongswanCert.pem
- x509/moonCert.pem
- rsa/moonKey.pem

StrongSwan is now ready to create the connection. To activate the connection, the daemon has to be started and the connection configuration has to be loaded. The commands to accomplish that can be found in figure 5.5.

[5]The certificate documentation can be found at `https://docs.strongswan.org/docs/latest/pki/pkiQuickstart`

```
systemctl enable --now strongswan
swanctl --load-all
```

Figure 5.5: Commands used to start the IPsec connection

The connection is now active. All further communication between the peers will use IPsec ESP with the per-CPU Security Associations feature.

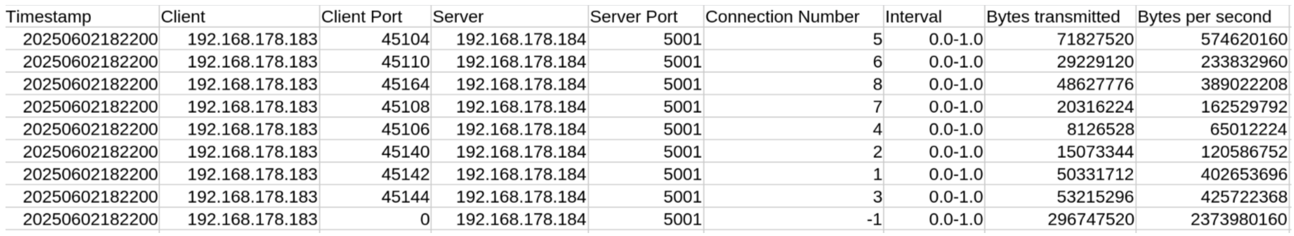
5.1.3 Data Collection

To collect the throughput data, iperf2[6] was used with the command line flags in figure 5.6.

```
# command run on moon
iperf -s
# command run on sun
iperf -c moon -P8 -t600 -fM -i1 -yC -o output.csv
```

Figure 5.6: Commands used to collect throughput data

The iperf instance will be started with eight parallel connections and output its results in one second intervals. Data in output.csv is formatted as can be seen in figure 5.7. The labels were not present in the output; they were added after data collection to clarify the data contained in the columns.



Timestamp	Client	Client Port	Server	Server Port	Connection Number	Interval	Bytes transmitted	Bytes per second
20250602182200	192.168.178.183	45104	192.168.178.184	5001	5	0.0-1.0	71827520	574620160
20250602182200	192.168.178.183	45110	192.168.178.184	5001	6	0.0-1.0	29229120	233832960
20250602182200	192.168.178.183	45164	192.168.178.184	5001	8	0.0-1.0	48627776	389022208
20250602182200	192.168.178.183	45108	192.168.178.184	5001	7	0.0-1.0	20316224	162529792
20250602182200	192.168.178.183	45106	192.168.178.184	5001	4	0.0-1.0	8126528	65012224
20250602182200	192.168.178.183	45140	192.168.178.184	5001	2	0.0-1.0	15073344	120586752
20250602182200	192.168.178.183	45142	192.168.178.184	5001	1	0.0-1.0	50331712	402653696
20250602182200	192.168.178.183	45144	192.168.178.184	5001	3	0.0-1.0	53215296	425722368
20250602182200	192.168.178.183	0	192.168.178.184	5001	-1	0.0-1.0	296747520	2373980160

Figure 5.7: Example throughput data from the IPsec test run

Every one of the eight connections gets output with transmitted bytes and the calculated transmission speed. Connection number ”-1” consolidates all eight connections by adding them together. This ”connection” represents cumulative network traffic that was sent in the last one second interval over the eight connections.

CPU utilization data was captured while iperf was running using a custom script. The bash script used can be seen in figure 5.8. It uses the mpstat tool to get the CPU utilization for each core. Captured data is transformed to CSV[7] and stored in a new ”cpu.csv” file.

[6]iperf is a tool for measuring network performance.
[7]CSV files are plain text files that hold tabular data seperated by commas

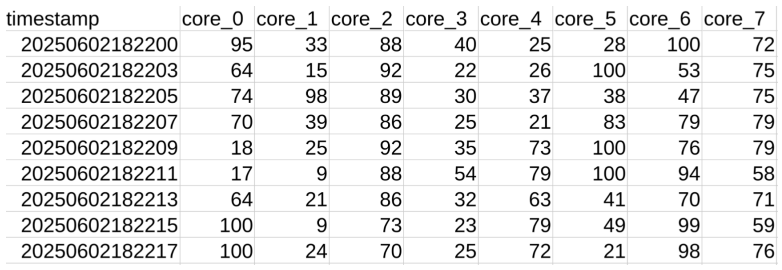
```
#!/bin/bash

echo "timestamp,core_0,core_1,core_2,core_3,core_4,core_5,core_6,core_7" > cpu.csv

while true; do
  ts=$(date +%Y%m%d%H%M%S)
  vals=$(mpstat -P 0-7 1 1 | awk '/^[0-9]/ { printf "%.2f\n", 100 - $12 }' | paste
      -sd "," - | cut -c8-)
  echo "$ts,$vals" >> cpu.csv
  sleep 1
done
```

Figure 5.8: Bash script used to capture CPU utilization data

Data in cpu.csv is formatted as can be seen in figure 5.9.



timestamp	core_0	core_1	core_2	core_3	core_4	core_5	core_6	core_7
20250602182200	95	33	88	40	25	28	100	72
20250602182203	64	15	92	22	26	100	53	75
20250602182205	74	98	89	30	37	38	47	75
20250602182207	70	39	86	25	21	83	79	79
20250602182209	18	25	92	35	73	100	76	79
20250602182211	17	9	88	54	79	100	94	58
20250602182213	64	21	86	32	63	41	70	71
20250602182215	100	9	73	23	79	49	99	59
20250602182217	100	24	70	25	72	21	98	76

Figure 5.9: Example CPU Utilization data from the sender of the IPsec test run

The script needs to be run parallel to iperf, preferrably in a seperate terminal to be able to terminate collection at will. If the process is not terminated, data collection will continue indefinitely.

5.2 Results

First off, the throughput results. In chart 5.10 is the data captured by iperf with ESP without the per-CPU SAs feature, with the per-CPU SAs feature as well as without any encryption at all.

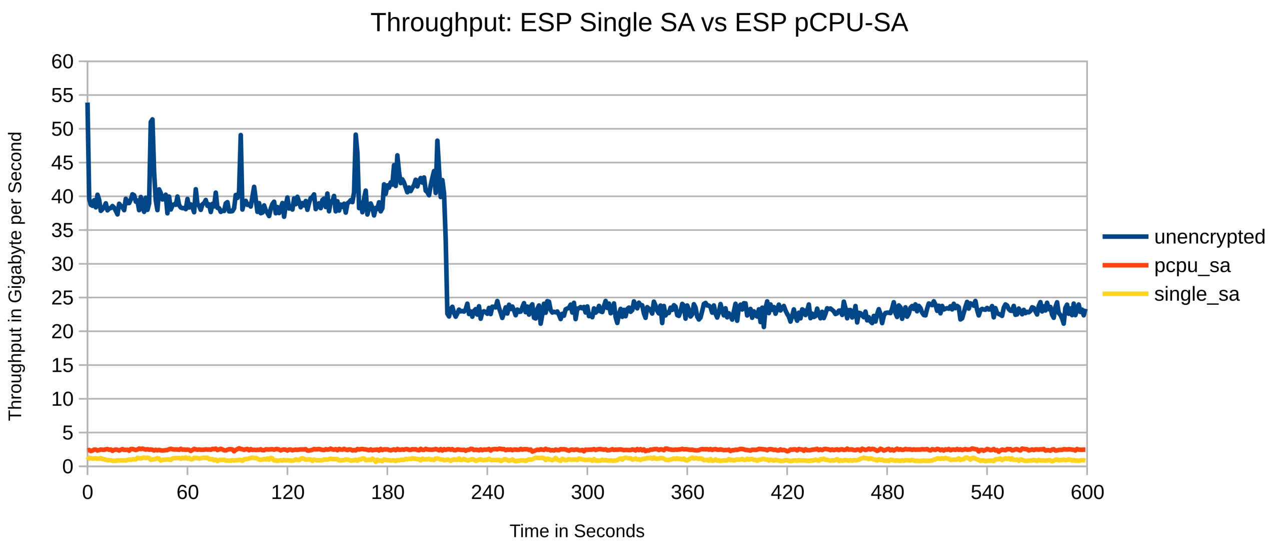


Figure 5.10: Unencrypted and Encrypted Throughput

The average unencrypted throughput of the first 3.6 minutes is 39.31 GB/s with a maximum and a minimum of respectively 53.89 GB/s and 37.32 GB/s. Over the next two seconds throughput drops rapidly from 40.46 GB/s to 22.61 GB/s. The average throughput thereafter is 23.03 GB/s with a maximum and a minimum of respectively 24.49 GB/s and 20.63 GB/s. There is more jitter in the former part compared to the latter.

The average IPsec ESP single SA throughput is 0.83 GB/s. Maximum and minimum values are 1.31 GB/s and 0.66 GB/s, respectively. There is very little jitter over the entire ten minute run.

The average IPsec ESP per-CPU SA throughput is 2.47 GB/s. Maximum and minimum values are 2.67 GB/s and 2.16 GB/s, respectively. Just like with the single SA run, there is very little jitter.

Next up are the CPU utilization metrics. In chart 5.11 is the sender and receiver CPU utilization for the unencrypted run.

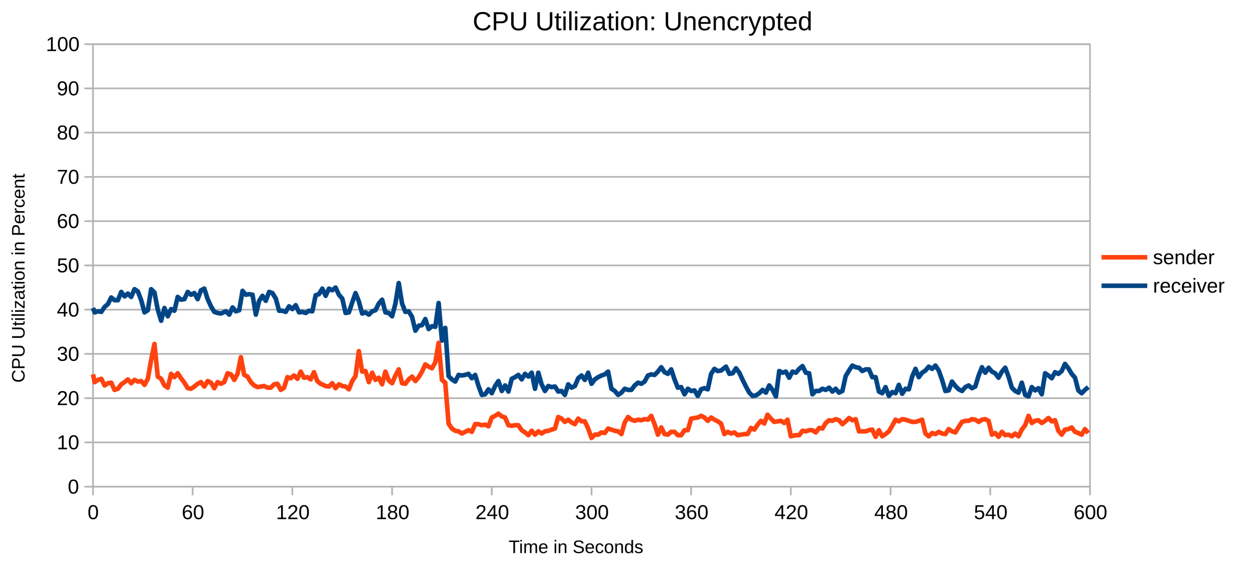


Figure 5.11: CPU Utilization without Encryption

The drop in throughput seen in chart 5.10 is also present in chart 5.11. In the first 3.6 minutes the average sender CPU utilization is 24.2% with a maximum and minimum of 32.5% and 14.3%, respectively. Average CPU utilization of the latter part is 13.5% with a maximum of 16.5% and a minimum of 11.0%.

The receiver side trends similarly but with a higher utilization overall Utilization of the former part is 40.8% on average with a maximum and minimum of 46.0% and 25.0%, respectively. CPU Utilization in the latter part is 23.8% on average with a maximum of 27.8% and a minimum of 20.4%. The receiver has on average 10.2% higher CPU utilization.

In chart 5.12 is the sender and receiver CPU utilization for the IPsec ESP per-CPU SAs run.

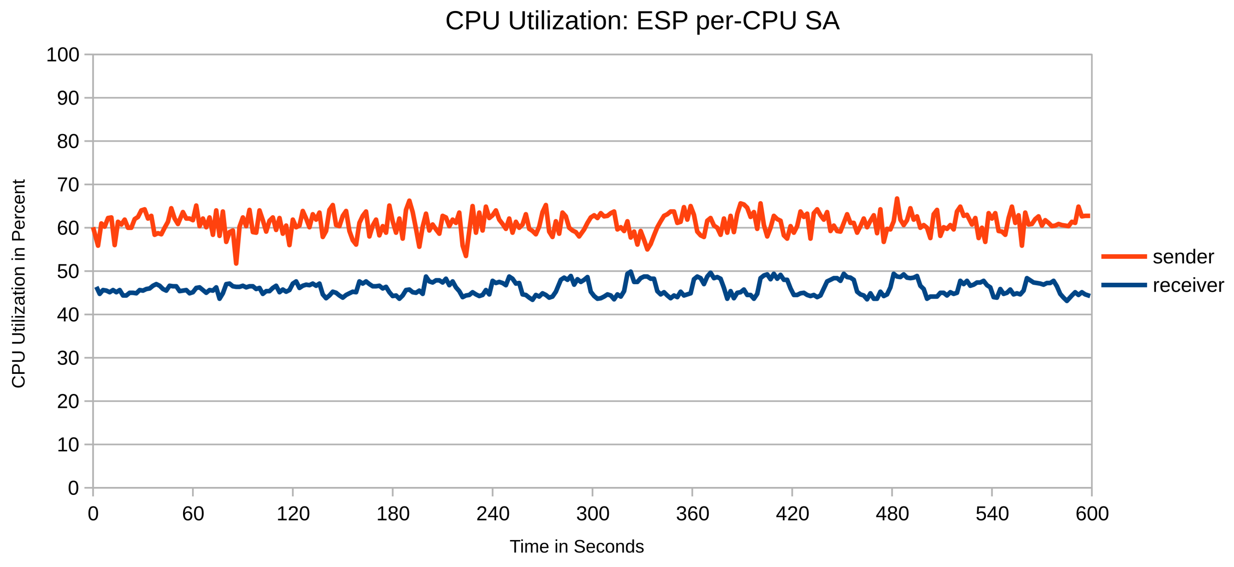


Figure 5.12: CPU Utilization with ESP per-CPU SAs

Just like with the throughput, CPU utilization is very stable when using single SA ESP. The average sender CPU utilization is 61.0% with a maximum of 66.8% and a minimum of 51.8%.

On the receiver side, utilization is 46.0% on average with a maximum and minimum of 49.9% and 43.1%, respectively. The receiver has 15.0% lower CPU utilization on average.

In chart 5.13 is the sender and receiver CPU utilization for the IPsec ESP single SA run.

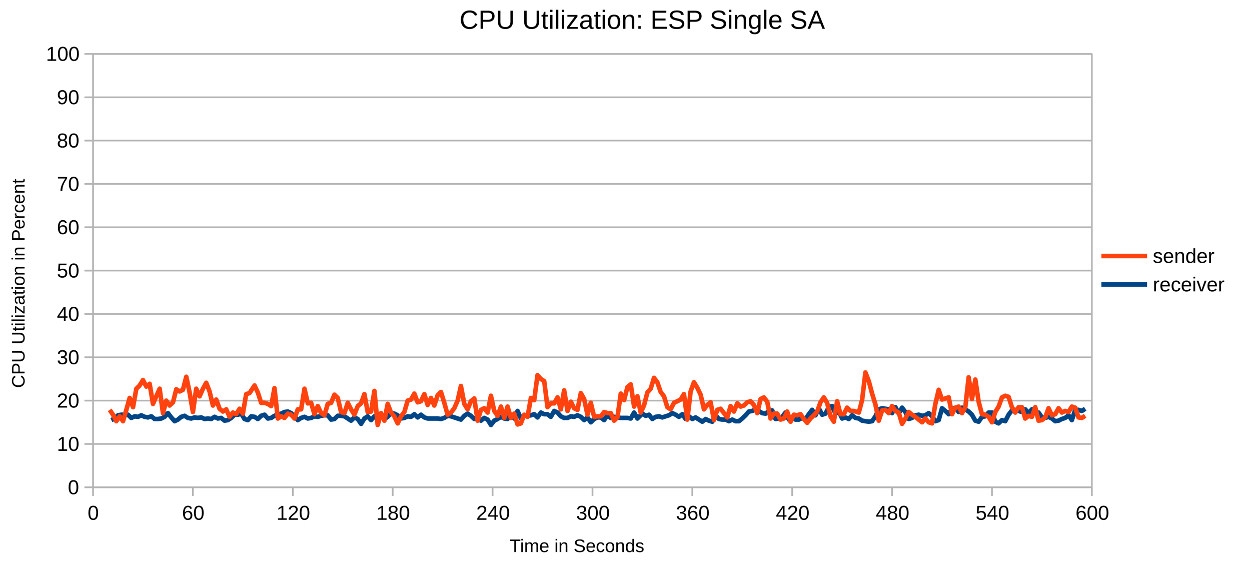


Figure 5.13: CPU Utilization with ESP single SA

Without the per-CPU SAs feature, utilization is similarly stable as with the feature enabled.

The average sender CPU utilization is 18.8% with a maximum of 26.5% and a minimum of 14.4%.

On the receiver side, utilization is 16.4% on average with a maximum and minimum of 18.8% and 14.4%, respectively. The receiver has 2.5% lower CPU utilization on average.

The difference between the sender and receiver is almost negligible in comparison to the unencrypted and per-CPU SA runs. With a single SA, the difference is 2.5%.

Using a single SA decreases CPU utilization by 1.0% in comparison to using no encryption. When using per-CPU SAs, CPU utilization is 54.6% higher compared to using a single SA.

The average CPU utilization is:

- 18.6% when using no encryption,
- 17.6% when using a single SA and
- 53.3% when using per-CPU SAs.

This concludes the gathered performance metrics. The full data set as well as this thesis can be found on my Github page: `https://github.com/maxweigner/pcpu-sa-performance`.

6 Summary

Both Google PSP and IPsec Enhanced ESP took ESP as a starting point and made necessary changes to achieve their goals. There are some differences in packet format, but the main difference between the two is the approach to storage and usage of encryption keys which enable multithreading and thus a performance increase.

Although an empirical comparison could not be made, it is possible to compare the new protocols theoretically. This is not optimal but serves as a baseline for expectations.

The concept is similar. Two or more peers have to exchange secret key material that is considered the master secret. With every packet sent, there is additional, public information contained that is used similar to a salt[1] to derive a new key from the master secret using a Key Derivation Function on a per-packet or per-flow basis, depending on implementation.

Using these derived keys, the CPU cores can encrypt, encapsulate, send and receive packets independently of each other. With regular ESP, this would not be possible while still maintaining the anti-replay feature which eliminates an entire class of attacks when enabled. When using this feature using multiple cores, for every packet sent or received, the state between cores would have to be synchronized to be able to set the right sequence number or drop repeated packets.

EESP takes the approach of the RFC 9611 proposed per-CPU SAs extension to ESP one step further to eliminate repeated IKE exchanges while retaining the anti-replay feature for each traffic flow. Using the master secret and the session ID contained in the packet, an anti-replay subspace is realized with its own sequence number independent of other cores' sequence numbers. This would not have been possible to do transparently with ESP.

Google however, decided to drop the anti-replay feature with PSP. The reason is presumably to simplify packet processing and enable offloading of the entire protocol to the Network Interface Card. Making decisions on whether to drop packets within a sliding window might be too resource intensive for commodity hardware. A feature that comes with this approach, is the possibility to keep the master secret on the NIC in ephemeral storage, inaccessible to the host. An attacker that has compromised one of the communicating hosts would not be able to access the master secret and therefore would not be able to decrypt past traffic that was captured by a Man-in-the-Middle.

When comparing IPsec EESP and Google PSP, it is to be expected that PSP would have the higher throughput; the reason for that is hardware offload. Processing on the NIC could be easily hardware accelerated, which would not be feasible for the host computer's CPU. Hardware built especially to process PSP traffic would, in theory, be able to achieve much higher throughput than any software implementation.

This does not apply to EESP. Though cryptography and encapsulation offload is possible and supported by the Linux kernel, the Security Associations must be synchronized between the kernel and the Network Interface Card [19]. The keys are not stored on the NIC [19], which means the kernel and therefore the host is still actively involved. Therefore, throughput has a higher performance ceiling compared to EESP when using PSP, assuming proper hardware is available.

6.1 Evaluation

The results in chapter 5.2 show, that performance does increase going from one Security Association to eight by almost three times. No linear scaling, as claimed in RFC 9611, was observed.

This may be because of this thesis' experimental setup. A virtualized environment may have unforeseen bottlenecks that can be avoided by using a baremetal installation as well as hardware offload onto the

[1]"Salting refers to adding random data to a hash function to obtain a unique output [..]"[18] for the same input material.

Network Interface Card. Also, this layer of abstraction may cause additional overhead that a bare metal installation does not have.

The single SA run has a lower sender-receiver CPU utilization difference in comparison with the per-CPU SA run. This may be due to overhead caused by traffic distribution over all available Security Associations. This also indicates, that there may be a limit to how much the per-CPU SA feature can be scaled while still increasing performance in a meaningful way.

Interestingly, there is a drop in throughput as well as in CPU utilization around the 3.6 minute mark of the unencrypted run. There is no obvious reason as to why that would be the case. The assumption would have to be an issue in the experimental environment, specifically the virtualization.

6.2 Future Work

More tests with a fully developed IPsec Enhanced Encapsulating Security Payload implementation have to be done. Since currently there are no available implementations, performance can only be estimated via the per-CPU SA feature proposed in RFC 9611.

The missing tool for using Google PSP would have to be made or published. As it stands, their call for wider usage of PSP seems half-hearted at best. Should the tool or another implementation come out, further testing using full hardware offload need to be done.

Finally, a comparison between Enhanced ESP and other VPN solutions would be interesting in varying environments - virtualized, baremetal and NIC hardware offloaded. Comparisons to OpenVPN and Wireguard would be appropriate, though both solutions do not offer the full feature set that EESP provides. The extra features may make it the only feasible option in enterprise environments, where traceability and various other network flow metrics are important.

7 Bibliography

[1] A. Vahdat and S. H. Yeganeh. “Announcing PSP’s cryptographic hardware offload at scale is now open source,” Accessed: Apr. 10, 2025. [Online]. Available: `https://cloud.google.com/blog/products/identity-security/announcing-psp-security-protocol-is-now-open-source?hl=en`.

[2] R. Atkinson, *IP Encapsulating Security Payload (ESP)*, RFC 1827, Aug. 1995. DOI: `10.17487/RFC1827`. [Online]. Available: `https://www.rfc-editor.org/info/rfc1827`.

[3] S. Kent, *IP Encapsulating Security Payload (ESP)*, RFC 4303, Dec. 2005. DOI: `10.17487/RFC4303`. [Online]. Available: `https://www.rfc-editor.org/info/rfc4303`.

[4] “Enhanced Encapsulating Security Payload (EESP) Document History,” Accessed: Apr. 14, 2025. [Online]. Available: `https://datatracker.ietf.org/doc/draft-klassert-ipsecme-eesp/history/`.

[5] ISO/IEC 7498-1:1994, *Information technology — Open Systems Interconnection — Basic Reference Model: The Basic Model*, Nov. 1994. [Online]. Available: `https://www.iso.org/standard/20269.html`.

[6] Wikipedia contributors. “OSI model,” Accessed: Apr. 8, 2025. [Online]. Available: `https://en.wikipedia.org/wiki/OSI_model`.

[7] C. Kaufman, P. E. Hoffman, Y. Nir, P. Eronen, and T. Kivinen, *Internet Key Exchange Protocol Version 2 (IKEv2)*, RFC 7296, Oct. 2014. DOI: `10.17487/RFC7296`. [Online]. Available: `https://www.rfc-editor.org/info/rfc7296`.

[8] R. Atkinson, *Security Architecture for the Internet Protocol*, RFC 1825, Aug. 1995. DOI: `10.17487/RFC1825`. [Online]. Available: `https://www.rfc-editor.org/info/rfc1825`.

[9] S. Klassert, A. Antony, and C. Hopps, “Enhanced Encapsulating Security Payload (EESP),” Internet Engineering Task Force, Internet-Draft draft-ietf-ipsecme-eesp-00, Apr. 2025, Work in Progress, 44 pp. [Online]. Available: `https://datatracker.ietf.org/doc/draft-ietf-ipsecme-eesp/00/`.

[10] V. Volpe, M. Stenberg, B. Swander, L. DiBurro, and A. Huttunen, *UDP Encapsulation of IPsec ESP Packets*, RFC 3948, Jan. 2005. DOI: `10.17487/RFC3948`. [Online]. Available: `https://www.rfc-editor.org/info/rfc3948`.

[11] National Institute of Standards and Technology. “Initialization Vector (IV),” Accessed: May 14, 2025. [Online]. Available: `https://csrc.nist.gov/glossary/term/initialization_vector`.

[12] A. Antony, T. Brunner, S. Klassert, and P. Wouters, *Internet Key Exchange Protocol Version 2 (IKEv2) Support for Per-Resource Child Security Associations (SAs)*, RFC 9611, Jul. 2024. DOI: `10.17487/RFC9611`. [Online]. Available: `https://www.rfc-editor.org/info/rfc9611`.

[13] Google, *PSP Architecture Specification*, Nov. 17, 2022. Accessed: May 3, 2025. [Online]. Available: `https://github.com/google/psp/blob/main/doc/PSP_Arch_Spec.pdf`.

[14] Huawei Technologies Co., Ltd. “NetStream Packet Sampling,” Accessed: Jun. 3, 2025. [Online]. Available: `https://support.huawei.com/enterprise/en/doc/EDOC1100034074/60f64bd/netstream-packet-sampling`.

[15] Hewlett Packard Enterprise Development. “vni,” Accessed: Jun. 3, 2025. [Online]. Available: `https://arubanetworking.hpe.com/techdocs/AOS-CX/10.12/HTML/vxlan/Content/VXLAN_cmds/vni-10.htm`.

[16] The strongSwan Team. “strongSwan,” Accessed: Jun. 3, 2025. [Online]. Available: `https://strongswan.org`.

[17] Amazon Web Services, Inc. “What’s the difference between Type 1 and Type 2 Hypervisors? ”[Online]. Available: `{https://aws.amazon.com/compare/the-difference-between-type-1-and-type-2-hypervisors/}`.

[18] N. Devi. “What Is a Salt and How Does It Boost Security? ”[Online]. Available: {https://www.loginradius.com/blog/identity/what-is-salt}.

[19] S. Nelson and L. Romanovsky. “XFRM device - offloading the IPsec computations. ”[Online]. Available: {https://docs.kernel.org/networking/xfrm_device.html}.