Politecnico di Torino

MACsec

A project on Media Access Control (MAC) Security over LAN and WAN technologies

Rollo Francesco s328739

Networks, Cloud and Application Security

September 2024

Contents

Chapter 1

MACsec Overview

MAC Security (MACsec), specified as **IEEE 802.1AE** [1], is an IEEE standard introduced in 2006 that defines a protocol providing Layer 2 security for wired ethernet LANs. Before exploring the technical specifications of MACsec and its functionality, it's essential to first understand the underlying need for this protocol.

IEEE 802 Local Area Networks (LANs) are widely used in environments where the integrity and reliability of data are critical, such as large corporate networks and public networks serving a broad range of customers. These networks often carry sensitive information and are managed by protocols that govern access and operation directly over the network infrastructure itself. Given the increasing complexity of these networks and the difficulty in fully securing them against unauthorized physical access, there is a growing need to prevent disruptions, data breaches, and unauthorized transmissions.

MACsec was developed to address these concerns. It enables **secure communication** between devices connected to the same LAN, ensuring that data remains confidential and protected from tampering. MACsec provides confidentiality, integrity and (data origin) authentication at the data link layer, allowing organizations to safeguard their network traffic against malicious activities such as eavesdropping or manipulation by unauthorized devices. This security protocol ensures that only authorized parties can transmit and receive data, effectively mitigating risks associated with unauthorized access or manipulation of network traffic.

Securing L2 data transmission with the MACsec protocol offers several benefits by achieving the following security properties:

- **Confidentiality**: through encryption, MACsec ensures that Ethernet frames cannot be inspected by unauthorized parties, safeguarding sensitive information from eavesdropping. Beware that MACsec offers two protection modes: integrity only, or integrity with confidentiality.
- **Connectionless data integrity**: MACsec verify that data has not been altered during transmission with the use of an ICV (Integrity Check Value), ensuring that any unauthorized modifications are immediately detected.
- **Data origin authentication**: MACsec guarantees that each Ethernet frame originates from a legitimate, authenticated device, providing proof of data origin, thanks to a preliminary key exchange process, and reducing the risk of spoofing attacks.
- **Replay protection**: the protocol prevents the reuse of captured data packets by ensuring that copied frames cannot be retransmitted without detection, defending against replay attacks.
- **Bounded receive delay**: MACsec includes protection against man-in-the-middle (MitM) attacks by monitoring transmission delays. It detects and flags any suspicious delays beyond a few seconds, which could indicate interception or manipulation.

Beyond its robust security features, MACsec supports high-speed network scalability, making it suitable for modern, high-throughput environments. It can also be fully implemented in hardware,

and it enables optimal latency due to pre-processing, ensuring efficient and low-latency network performance. It's worth noting that MACsec was introduced to the **Linux kernel** in version 4.7 (released in 2016) and has since undergone significant enhancements over the years.

MACsec protocol is particularly effective at mitigating a wide range of network attacks that target the data link layer. Some of the key attacks MACsec defends against include:

- **Man-in-the-Middle (MitM)**: attackers intercept and modify packets between two nodes, potentially reading or altering the communication. MACsec thwarts this by ensuring that only authorized nodes with valid encryption keys can participate in the communication.
- **Denial-of-Service (DoS)**: attackers flood a network with traffic to overwhelm resources and disrupt services. MACsec's anti-replay protection, which numbers each frame, prevents attackers from overwhelming the network with duplicate or manipulated frames.
- **MAC Spoofing**: attackers impersonate devices by altering their MAC address to gain unauthorized access to network resources. MACsec's authentication mechanisms prevent unauthorized devices from gaining access, ensuring only verified nodes can participate.
- **MAC Flooding**: attackers flood a switch with continuous packets to fill the switch's Content Addressable Memory (CAM) table. MACsec protects against this by maintaining secure and authenticated connections between switches and endpoints.
- **ARP Spoofing**: attackers manipulate Address Resolution Protocol (ARP) tables to redirect traffic to malicious devices. With MACsec, unauthorized nodes are unable to modify ARP tables since only authenticated devices can interact with network resources.
- **Rogue DHCP**: attackers impersonate the DHCP server to issue incorrect network configurations, disrupting services or launching MitM attacks. MACsec helps ensure that only authorized DHCP servers can communicate with network clients.

MACsec's ability to provide hop-by-hop encryption, preemptive authentication, and frame numbering for anti-replay protection makes it an essential tool in safeguarding the data link layer against these types of attacks.

1.1 MACsec Architecture

MACsec provides robust security for data transmission over Ethernet networks by offering two modes of protection: integrity-only and integrity with confidentiality. In the integrity-only mode, frames are transmitted in the clear but are still protected against tampering and replay attacks. In the confidentiality mode, authenticated encryption is used to protect the data payload.

MACsec typically employs the 128-bit Advanced Encryption Standard in Galois Counter Mode (**AES-GCM**), providing Authenticated Encryption with Additional Data (**AEAD**), which allows to authenticate and ensure integrity-protection of an entire frame, including all its headers. Other cipher suites, such as GCM-AES-256, GCM-AES-XPN-128, and GCM-AES-XPN-256, extend the standard offerings. While they provide additional security options, they are less frequently used. This flexibility allows MACsec to safeguard data at a granular level by encrypting the payload while leaving essential headers in cleartext for proper delivery. Furthermore, it can also be used for integrity-only protection, by passing the entire frame to the algorithm as additional data.

In MACsec terminology, a **Security Entity** (**SecY**) is an instance of the MACsec implementation within a node. To establish secure communication between security entities, both devices must be part of the same **Connectivity Association** (**CA**). A CA consists of a group of devices that share a common key and cipher suite. The shared key, called the **Connectivity Association Key** (**CAK**), is pre-shared, associated to a **Connectivity Association Key Name** (**CKN**) and used to authenticate devices.

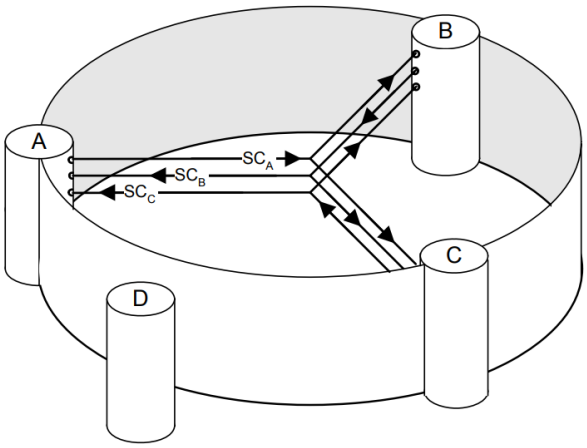

Figure 1.1: Secure communication between three stations within a CA

Within a CA, communication occurs through unidirectional **Secure Channels** (**SC**), which allow data transmission from one device to another or multiple devices. Each SC is identified by a **Secure Channel Identifier** (**SCI**), which is derived from the device's MAC address and port number. **Secure Associations** (**SAs**) are established within each SC, with each SA using a unique **Security Association Key** (**SAK**), a symmetric key used to encrypt the data. Each SAK within the same SC must be unique and associated with a **Secure Association Identifier** (**SAI**), which is composed of the SCI and the **Access Number** (**AN**). In a connection, it is not possible to have more than four SAs at the same time.

Each MACsec node contains two entities: the MAC Security Entity (**SecY**), introduced before, and the Key Agreement Entity (**KaY**). The SecY is responsible for managing all encryption and validation processes for incoming and outgoing frames. This entity primarily holds the cipher suite necessary to ensure the integrity of the packet and may also encrypt the MAC Service Data Unit (**MSDU**). Specifically, to verify the integrity of a frame, the SecY utilizes the key indicated by the SecTag to derive the Integrity Check Value (ICV) and compares it with the ICV received in the incoming frame.
On the other hand, the KaY is mainly tasked with the MACsec Key Agreement (**MKA**) protocol, not specified in the MACsec standard, which governs key management, distribution, and creation. Additionally, it serves as the primary method for recognizing other nodes within the Connectivity Association (CA) by verifying their possession of the same Connectivity Association Key (CAK).

1.2 MACsec Frame Structure

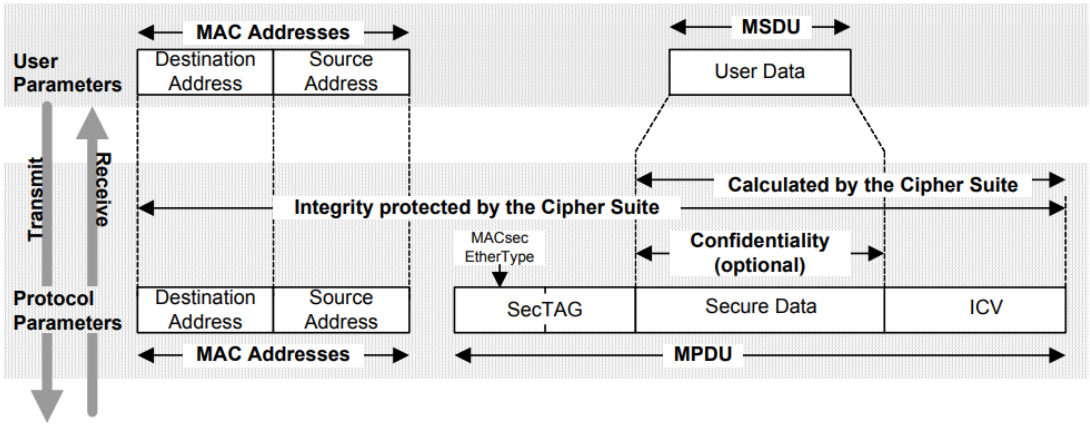

Figure 1.2: MACsec frame

When a packet passes through a MACsec-enabled device, several modifications are made to the original Ethernet frame. The frame is prepended with a Security Tag (**SecTAG**), which contains important metadata for securing the packet. The Ethertype field in the frame is changed to the MACsec Ethertype (**0x88e5**) to signal that the packet is protected by MACsec. Additionally, an Integrity Check Value (**ICV**) is appended to the frame. The ICV is calculated over the entire frame, including the SecTAG, the destination, and source MAC addresses, to ensure the integrity of the data during transmission.

When encryption is enabled, only part of the frame is encrypted, starting from the original Ethertype onward, including the IP headers and payload (represented as the **MSDU**). The MSDU's integrity is always protected, independent of whether encryption is configured or not.
The source and destination MAC addresses, along with the SecTAG, remain in cleartext but are protected by the ICV, as they are passed to the cipher suite as "additional data". This ensures that even though these headers are not encrypted, any attempt to tamper with them can be detected.

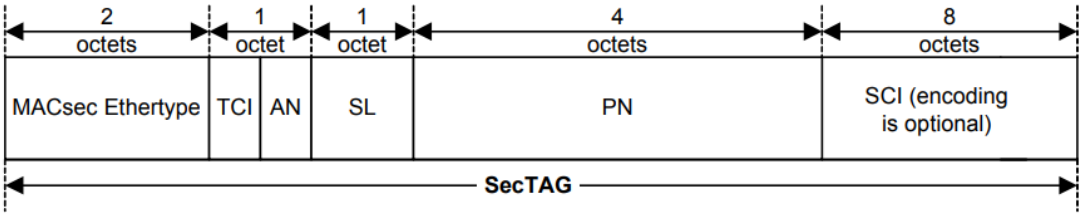

Figure 1.3: SecTAG format

The SecTAG is a critical component of the MACsec frame. It consists of several fields (shown in Figure 1.5) that define the configuration and security settings applied to the packet. Key fields in the SecTAG include:

- MACsec Ethertype (0x88e5): This field designates the packet as belonging to the MACsec protocol.
- Tag Control Information (TCI): This field contains several bits that describe the security settings, such as:
  1. Version numbering of the MACsec protocol
  2. Optional use of the MAC Source Address parameter to convey the SCI
  3. Optional inclusion of an explicitly encoded SCI
  4. Use of the EPON Single Copy Broadcast (SCB) capability without explicit SCI bit

5. Extraction of the User Data from MPDUs by systems that do not possess the SAK when confidentiality is not being provided
6. Determination of whether confidentiality or integrity alone are in use

- Association Number (AN): This 2-bit field identifies which Security Association (SA) is used within the Secure Channel (SC).
- Secure Length (SL): Indicates the length of the SecTAG field. If the frame length exceeds 48 bytes, this field is set to zero.
- Packet Number (PN): A 32-bit counter used to prevent replay attacks by tracking the sequence of transmitted packets.
- Secure Channel Identifier (SCI): A unique 64-bit value that combines the MAC address and port number to identify the sender of the packet in multi-node CAs.

1.3 MACsec Key Agreement

The **IEEE 802.1X-2010** [2] standard, also known as the **MACsec Key Agreement (MKA) protocol**, handles key distribution and authentication between nodes. MKA ensures mutual authentication by verifying that all devices in the CA possess the same CAK, thus enabling secure communication. The protocol also generates and distributes **Security Association Keys** (**SAKs**) used to encrypt transmitted frames.
In summary:

- **CAK**: a pre-shared key obtained through EAP authentication. It is associated with a CKN, an identifier used to distinguish between various CAKs.
- **SAK**: a symmetric key used to encrypt the frame's payload, generated by a random number generator (RNG) server, which during the Extensible Authentication Protocol over LAN (**EAPoL**) authentication process always corresponds to the authenticator.

The MKA protocol controls the possession of the CAK and distributes the SAKs through the **Key Server** (**KS**), which is elected from the nodes based on priority. If the CAK is obtained through external authentication, the Extensible Authentication Protocol (**EAP**) authenticator becomes the KS.

The generation of the keys follows two different procedures. The CAK is generated through an EAPOL authentication method carried out by the participants in the CA. In contrast, the SAK is generated using a Random Number Generator (RNG) server, and its distribution follows a different process performed by the Key Server (KS).

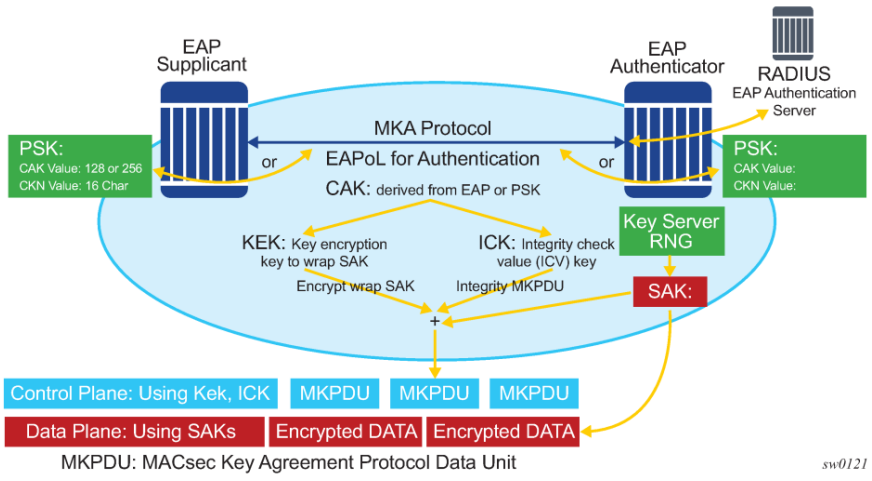

Figure 1.4: MACsec generating the CAK

Starting from the CAK, the Key Encrypting Key (**KEK**) and the Integrity Check Value Key (**ICK**) are derived. The KEK is used for the actual distribution of the SAK within the CA, while

the ICK is a key that will be employed to verify the integrity of the frame through the AES-CMAC algorithm.
The main body of the transmitted EAPOL frame, which contains the SAK, is known as the MACsec Key Agreement Protocol Data Unit (**MKPDU**). This unit is further protected for integrity using the ICV with the same ICK. Typically, this transmitted packet serves not only for the distribution of the key but also to confirm the active presence of other participants in the CA by continuously sending a signal. Additionally, it facilitates the distribution of the key used for integrity checking and the configuration of various nodes.

1.4 VLAN support

When MACsec is applied to protect a VLAN, the VLAN tag becomes part of the encrypted payload, ensuring that the entire frame, including VLAN-related information, is protected. This feature is particularly useful in environments where VLANs are used to segment traffic, as it ensures the confidentiality and integrity of VLAN-tagged traffic.

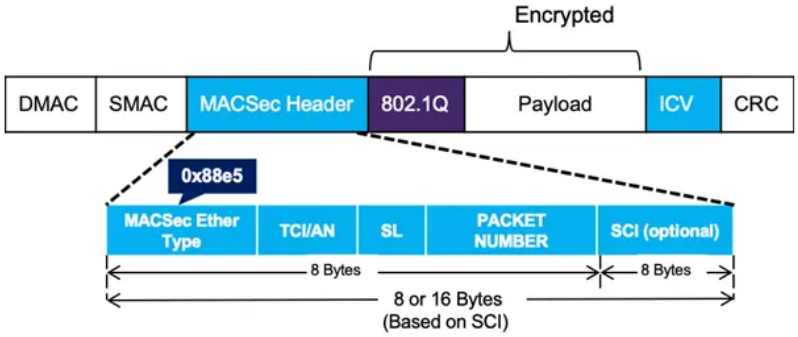

Figure 1.5: VLAN 802.1Q Tag Encrypted

1.5 Use cases

The increasingly widespread use of Ethernet as the primary communication method between data centers, switch/routers or servers is driven by the need for higher connection speeds compared to those used for Layer 3 or 4 connections. The main obstacle to the adoption of Ethernet technology was its security level, as the channel remained vulnerable to the various attacks mentioned before. MACsec can be implemented in various scenarios, which will be presented in the following sections.

1.5.1 LAN

The main use case for MACsec, as defined by the standard, is to secure a Local Area Network. In this scenario, multiple hosts connected to the same LAN are configured to ensure that all packets exchanged between them are encrypted and can only be received by nodes that support MACsec.

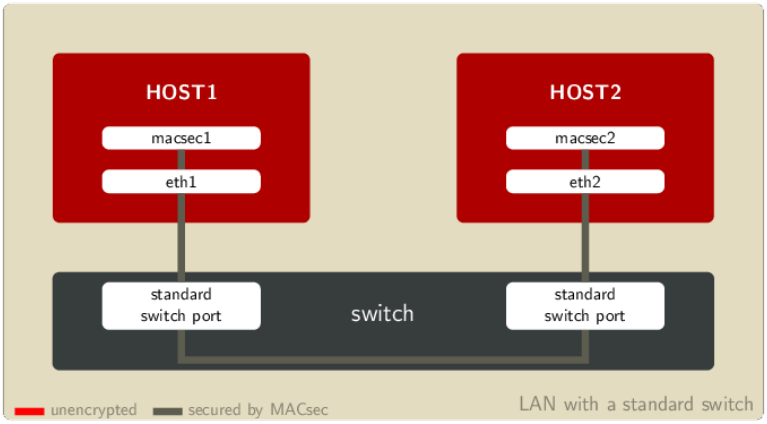

Figure 1.6: MACsec LAN use case with standard switches

In this setup, a LAN with standard switches is employed. All the traffic marked by a grey line is secured by MACsec. MACsec is terminated on the hosts and the switches are not capable of encrypting frames, but only to forward MACsec-protected frames between ports.

An alternative solution is to use switches that support MACsec. In this case, MACsec is enabled on the hosts as well as on the switch ports to which these nodes are connected. Typically, these switches employ robust Authorization, Authentication, and Accounting (**AAA**) mechanisms through the 802.1X standard, enabling port-based access control before allowing clients to connect to the network. Note that red links are not secured by MACsec.

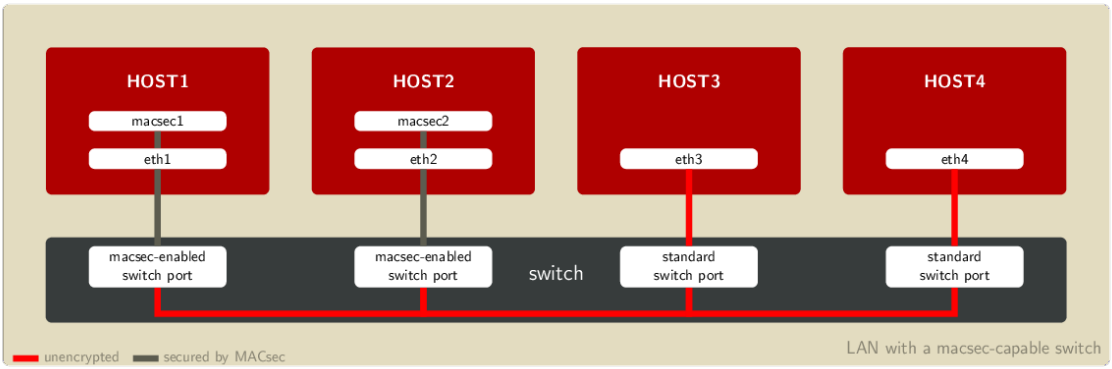

Figure 1.7: Alternative MACsec LAN use case with macsec-capable switches

As shown in the Figure 4.2, hosts 1 and 2, along with the first two switches, have MACsec enabled, while hosts 3 and 4 do not use MACsec. MACsec is terminated at the switch ports, ensuring secure communication between the first two hosts and switches. At the same time, the MACsec-enabled switches decapsulate MACsec frames when forwarding traffic to the non-MACsec hosts and encapsulate Ethernet frames from these non-MACsec nodes towards the MACsec-enabled hosts connected to them.

1.5.2 WAN

As organizations expand their networks across different geographical regions and service provider infrastructures, the need for securing Layer 2 communication becomes increasingly important, not only for ensuring confidentiality and data integrity, but also for minimizing latency and overhead. Traditional security methods like IPsec, while highly secure, can introduce significant processing overhead and higher latency due to their complex encryption and tunneling mechanisms at Layer 3. **MACsec over WAN** provides a solution by extending Layer 2 encryption across various WAN technologies, ensuring confidentiality, integrity, and authentication for Ethernet traffic even in wide-area environments.

MACsec over WAN can be implemented using various technologies and architectures depending on the underlying WAN infrastructure. Below are several common deployment scenarios.

MACsec over QinQ

Enterprises with multiple geographically distributed sites rely on service providers to link their networks using QinQ (**IEEE 802.1ad**), also known as provider bridging. In this setup, customer VLANs (C-VLANs) are encapsulated inside service provider VLANs (S-VLANs) for transport across the provider's infrastructure. In this scenario:

- MACsec is applied to the customer traffic at the network edge, securing the customer's Ethernet frames before they enter the service provider's network.
- Inside the service provider's network, the customer traffic is encapsulated within an S-VLAN tag. The original MACsec-encrypted frame remains intact and encrypted.
- The encrypted customer frame is delivered across the service provider's network without being decrypted, ensuring end-to-end security for the customer's Layer 2 traffic.

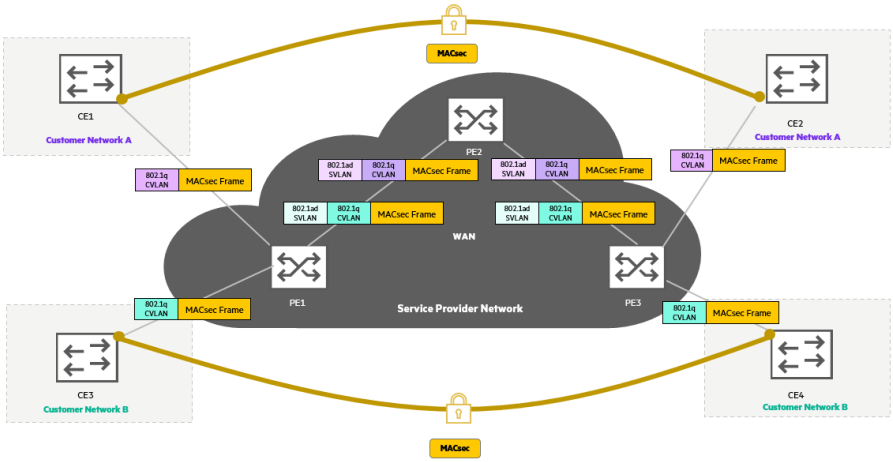

Figure 1.8: MACsec over QinQ (HPE Aruba Networking ©)

MACsec over VXLAN/GREtap/GENEVE

An organization may have a distributed data center architecture with Layer 2 connectivity across different sites and they often require secure extensions over IP-based WANs. **VXLAN** (Virtual Extensible LAN), **GREtap** (Generic Routing Encapsulation with Ethernet bridging), and **GENEVE** (Generic Network Virtualization Encapsulation) are three widely adopted tunneling technologies used to encapsulate Layer 2 traffic in a Layer 3 (IP) networks. To ensure that this Layer 2 traffic remains secure as it traverses untrusted WAN infrastructures, MACsec can be applied before the encapsulation process, adding a layer of encryption and authentication to Ethernet frames before they are tunneled.

Taking as an example MACsec over VXLAN:

- MACsec is deployed between switches or routers acting as VXLAN Tunnel Endpoints (VTEPs). These devices encrypt Ethernet frames before encapsulating them into VXLAN packets.
- The encrypted MACsec frames are transmitted across the IP-based WAN through VXLAN tunnels, providing end-to-end Layer 2 encryption.
- At the remote site, the VXLAN packet is decapsulated, and the MACsec-encrypted frame is delivered to the destination.

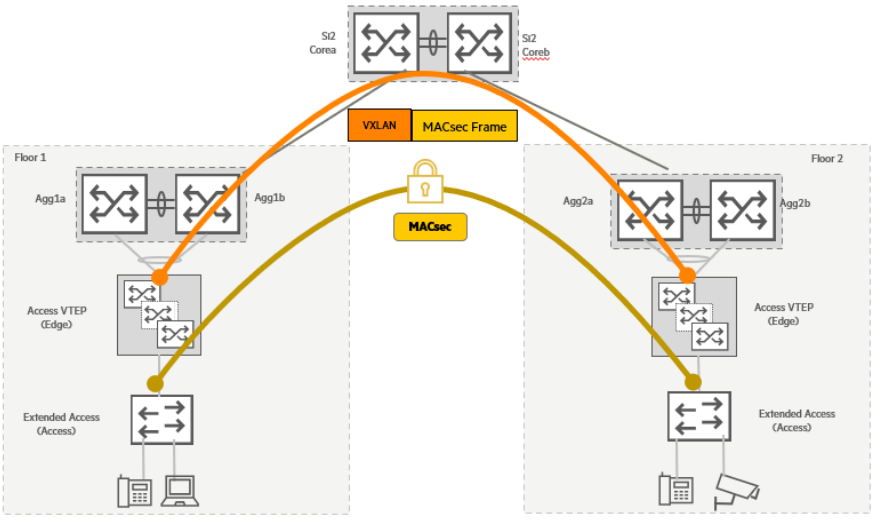

Figure 1.9: MACsec over VXLAN (HPE Aruba Networking ©)

MACsec over MPLS Pseudowire

Many enterprises use **MPLS** (Multiprotocol Label Switching) to interconnect multiple sites over a service provider network. In such cases, a **pseudowire** is often used to create a virtual Layer 2 connection between sites. In this scenario:

- MACsec is applied to the Layer 2 traffic at the enterprise's edge device. The MACsec-encrypted frame is then sent across the MPLS network through the pseudowire.
- The service provider forwards the encrypted frame without decrypting it, preserving the confidentiality of the traffic as it traverses the MPLS cloud.
- At the destination site, the original MACsec-encrypted frame is delivered and decrypted by the MACsec-enabled device.

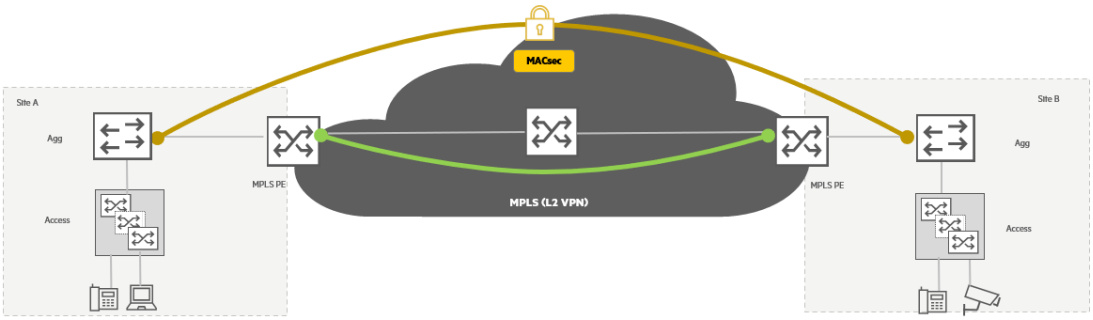

Figure 1.10: MACsec over MPLS (HPE Aruba Networking ©)

Dual Encryption: MACsec over IPsec

In high-security environments, organizations often use IPsec to encrypt WAN traffic at Layer 3. For enhanced security, MACsec can be layered on top of IPsec to provide encryption at both Layer 2 and Layer 3. In this scenario:

- MACsec is applied within the LAN, encrypting Ethernet frames at the Layer 2 level. These encrypted frames are then encapsulated inside an IPsec tunnel.
- The IPsec tunnel provides encryption for the traffic across the WAN, ensuring that the data is encrypted both at Layer 2 (MACsec) and Layer 3 (IPsec).
- This dual-layer encryption ensures that even if an attacker compromises one layer, the other layer remains intact, offering an additional security buffer.

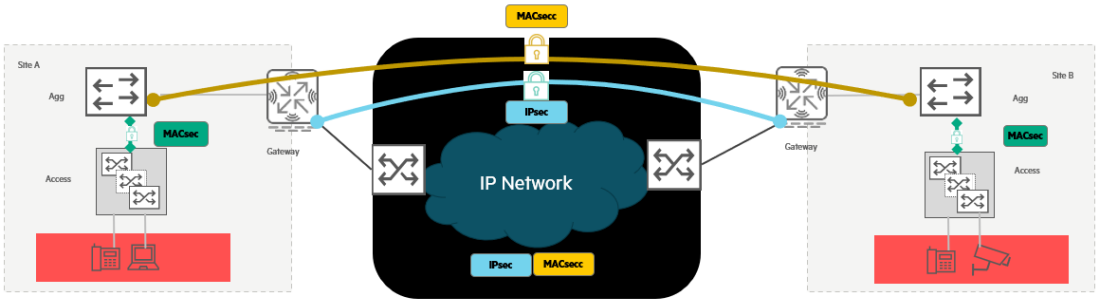

Figure 1.11: MACsec over IPsec (HPE Aruba Networking ©)

Chapter 2

MACsec LAN Simulation

This chapter proposes a simulation environment designed to emulate secure communication over a Local Area Network using MACsec. To simulate this secure LAN scenario, we use Linux network **namespaces**, which provide lightweight, isolated network environments on a single physical machine. Each network namespace behaves like a separate host, with its own network interfaces, routing tables, and processes. By creating multiple namespaces, we can replicate the behavior of distinct devices within a LAN, allowing us to configure and test MACsec in a controlled environment without the need for physical hardware or complex virtualization technologies.

In our setup, virtual Ethernet interfaces are created and bridged to form a virtual LAN, where MACsec-protected communication can take place between namespaces. This simulation requires the creation of namespaces, the configuration of network interfaces, and the setup of **wpa_supplicant**, a tool used used to manage MACsec connections.

wpa_supplicant

`wpa_supplicant` is a software tool that implements the IEEE 802.1X authentication protocol, primarily used for secure access in both wired and wireless networks. In the context of wired networks, such as MACsec-enabled environments, `wpa_supplicant` is essential for managing the authentication, encryption, and key management processes.

For MACsec devices, 802.1X is particularly essential because it governs the establishment of the MACsec Key Agreement (MKA). Thus, `wpa_supplicant` plays a dual role: it acts as the supplicant in the 802.1X framework for initial device authentication and manages the MKA process to enable secure MACsec connections ensuring that your associations remain up to date and perform new key exchange when necessary.

For this lab, the setup chosen is based on a **GNU/Linux** system (Debian-based).

Requirements

In order to make this simulation work, the following packages must be installed and up to date:

- Linux kernel $\geq$ 4.7.0
- Netlink library libnl $\geq$ 3.2.29
- wpa_supplicant $\geq$ 2.6
- ip-route2

2.1 Scenario

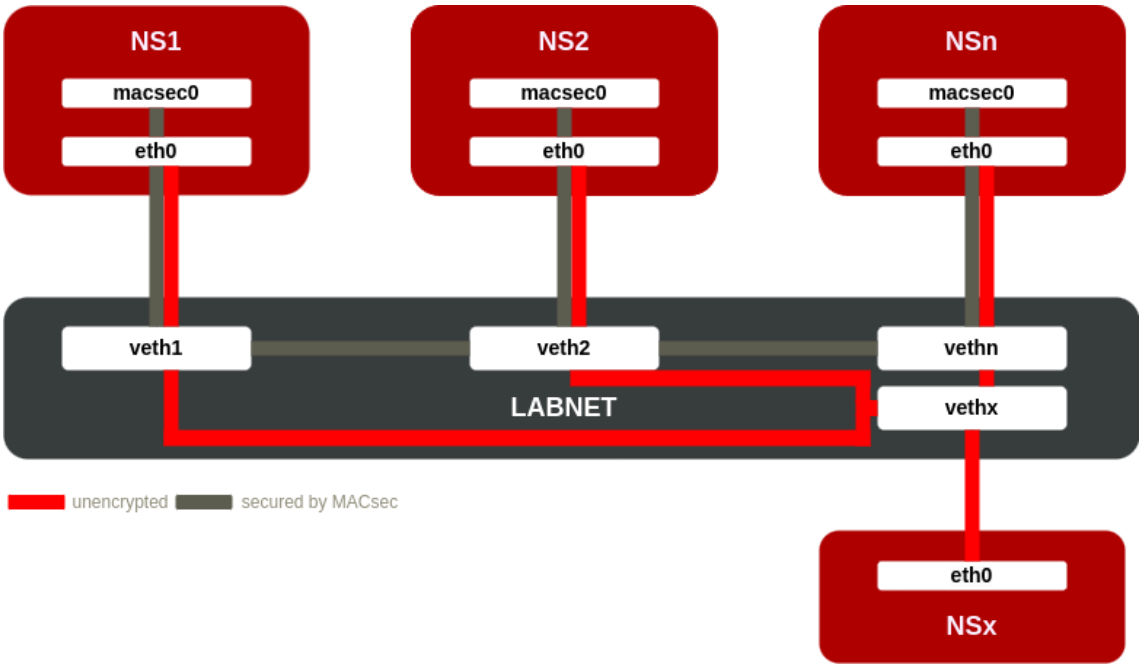

Figure 2.1: MACsec LAN simulation scenario

In this simulation, multiple **network namespaces** (e.g., ns1, ns2) are created to isolate network environments, allowing each namespace to operate indipendently.

A **bridge** named `labnet` is established in the root namespace to facilitate communication between these isolated namespaces.

Each namespace contains a **virtual Ethernet interface** (veth pairs) that connect to the bridge. For example, `veth1` connects `ns1` to `labnet`, while `veth2` connects `ns2`.
Within each namespace, interfaces are defined as `macsec0` and `eth0`, where `macsec0` secures the data transmission and eth0 connects to the respective veth pair.

To test MACsec-protected nodes alongside unprotected ones, an additional namespace (NSx) has been created to demonstrate the interactions between the entities involved in the communication.

2.2 Initial Setup

As first step, create 3 network namespaces. Two of them will act as MACsec-protected nodes, instead the last one will act as an unprotected node.

Create network namespaces

```
$ sudo ip netns add ns1
$ sudo ip netns add ns2
$ sudo ip netns add ns3 #non-MACsec node
```

Create a pair of virtual Ethernet interfaces where one end stays in the root network namespace and the other is moved to each respective namespace. The `lo` and `eth0` interfaces inside each namespace are also brought up.

Set up virtual ethernet interfaces

```
$ sudo ip link add veth1 type veth peer name eth0 netns ns1
$ sudo ip link add veth2 type veth peer name eth0 netns ns2
$ sudo ip link add veth3 type veth peer name eth0 netns ns3
```

Bring up interfaces

```
$ sudo ip netns exec ns1 ip link set dev eth0 up
$ sudo ip netns exec ns2 ip link set dev eth0 up
$ sudo ip netns exec ns3 ip link set dev eth0 up

$ sudo ip netns exec ns1 ip link set dev lo up
$ sudo ip netns exec ns2 ip link set dev lo up
$ sudo ip netns exec ns3 ip link set dev lo up
```

Assign now the MAC addresses for each interface in the different namespaces.

Bring up interfaces

```
$ sudo ip netns exec ns1 ip link set dev eth0 address 02:00:00:00:01:01
$ sudo ip netns exec ns2 ip link set dev eth0 address 02:00:00:00:01:02
$ sudo ip netns exec ns3 ip link set dev eth0 address 02:00:00:00:01:03
```

In the next step, create a network bridge called `labnet` to interconnect the virtual Ethernet interfaces. The bridge is then brought up and the forwarding of 802.1X PAE frames is enabled by setting the `group_fwd_mask` to `8`.

Create labnet bridge

```
$ sudo ip link add name labnet type bridge
$ sudo ip link set veth1 master labnet
$ sudo ip link set veth1 up
$ sudo ip link set veth2 master labnet
$ sudo ip link set veth2 up
$ sudo ip link set veth3 master labnet
$ sudo ip link set veth3 up
```

Bring up the bridge

```
$ sudo ip link set labnet up
```

The `group_fwd_mask` is a bitmask that controls the types of multicast and broadcast traffic that the bridge is allowed to forward.

The bit `8` specifically enables forwarding of PAE (Port Access Entity) group address frames, which includes EAPoL (**0x888e**) and MACsec MKA (**EtherType 0x88e5**) frames.

This is essential for 802.1X authentication and MACsec traffic to traverse the bridge correctly. Without setting this `group_fwd_mask` value, the bridge would drop or block those special frames, which would prevent MACsec from functioning properly in your simulated network environment.

Enable 802.1X PAE forwarding

```
$ sudo sh -c 'echo "8" > /sys/devices/virtual/net/labnet/bridge/group_fwd_mask'
```

2.3 WPA Supplicant Configuration

Before running `wpa_supplicant` on each node, it is necessary to create a proper configuration file to enable MACsec and EAP 802.1X. To do this, you need to create a new configuration file with the required parameters. For more details about `wpa_supplicant` configuration file follow this link.

For the MKA keys, you'll have to choose a **CAK/CKN** pair. You can, for example, generate a 16-byte key in hexadecimal notation this way:

Random hex bytes

```
$ dd if=/dev/urandom count=16 bs=1 2> /dev/null | hexdump -e '1/2 "%02x"'
```

Apply the same for a 32-byte hexadecimal key by just changing the `count` value to `32`.

Now create the following `wpa_supplicant` configuration file in the current working directory.

wpa_supplicant.conf

```
eapol_version=3
ap_scan=0
fast_reauth=1

network={
        key_mgmt=NONE
        eapol_flags=0
        macsec_policy=1
        #macsec_integ_only=1
        #macsec_replay_protect=1
        #macsec_replay_window=16
        #macsec_offload=0
        #macsec_port=1
        mka_cak=0011... # 16 bytes hexadecimal
        mka_ckn=2233... # 32 bytes hexadecimal
        mka_priority=128
}
```

Additionally, set up a directory to store logs, which will be useful for analyzing `wpa_supplicant` operations.

Setup logging

```
# Create the log directory
$ mkdir -p log

# Create an empty log file for namespace 1
$ touch log/wpa_supplicant_ns1.log

# Create an empty log file for namespace 2
$ touch log/wpa_supplicant_ns2.log
```

Now it's the time to actually spawn all the supplicant nodes. Each command starts a `wpa_supplicant` process inside the respective namespace (ns1, ns2). This simulates the MACsec protocol using the `macsec_linux` driver, with logs being saved for each namespace in the log directory.

Spawn supplicants

```
$ sudo ip netns exec ns1 wpa_supplicant -B -D macsec_linux -c wpa_supplicant.conf
-i eth0 -f log/wpa_supplicant_ns1.log -dd -K -t

$ sudo ip netns exec ns1 wpa_supplicant -B -D macsec_linux -c wpa_supplicant.conf
-i eth0 -f log/wpa_supplicant_ns2.log -dd -K -t
```

Explanation of the `wpa_supplicant` options:

- **-D macsec_linux**: Specifies the use of the MACsec driver to establish a MACsec connection.
- **-i eth0**: Tells wpa_supplicant to operate on the eth0 interface inside the namespace.
- **-c wpa_supplicant.conf**: Path to the configuration file where the MKA keying and MACsec settings are defined.
- **-f log/wpa_supplicant_ns*.log -dd -K -t**: Enables detailed debugging, logs the output to a specific log file, and includes timestamps for troubleshooting.

After the MKA protocol successfully establishes a secure session, the **macsec0** interface is created automatically by `wpa_supplicant` as a child interface of eth0. The macsec0 interface will be responsible for handling encrypted traffic over the physical eth0 link.

As the last step, you'll need to assign IP addresses to the **macsec0** interface in each namespace, except for the last namespace (the non-MACsec node) where the IP address will be assigned to the **eth0** interface. For the sake of the simulation, assigning only IPv4 addresses is enough (e.g., 10.0.0.0/16).

Assign IPv4 addresses

```
$ sudo ip netns exec ns1 ip address add 10.0.0.1/16 dev macsec0
$ sudo ip netns exec ns2 ip address add 10.0.0.2/16 dev macsec0
$ sudo ip netns exec ns3 ip address add 10.0.0.3/16 dev eth0
```

2.4 Analysis

After configuring the namespaces and setting up the interfaces, you may want to access each namespace to verify that the `macsec0` (or `eth0` in the case of the non-MACsec node) interfaces are set up correctly and running.

To access a specific namespace, you can use the following command:

Access namespaces

```
$ sudo ip netns exec <namespace_name> bash
```

Replace <namespace_name> with the actual namespace name (e.g., ns1, ns2, etc.). For example, to enter ns1:

Access ns1

```
$ sudo ip netns exec ns1 bash
```

This command starts a new bash session within the specified namespace, giving you access to run network commands as if you were in that environment.

This command starts a new bash session within the specified namespace, giving you access to run network commands as if you were in that environment.

List interfaces and their status:

```
$ ip link show
```

Check assigned IP addresses

```
$ ip addr show
```

For namespaces with MACsec configured, you can view additional details about the macsec0 interface:

View MACsec details

```
$ ip macsec show
```

This command will display MACsec configuration details, including any active secure associations or secure channels. You should see an output like the following:

```
3: macsec0: protect on validate strict sc off sa off encrypt on send_sci on end_station off scb off replay off
    cipher suite: GCM-AES-128, using ICV length 16
    TXSC: 0200009c32090001 on SA 0
        0: PN 16, state on, key 7b87b6edbf48050e81e6c08201000000
    RXSC: 02000025b8a40001, state on
        0: PN 16, state on, key 7b87b6edbf48050e81e6c08201000000
    offload: off
```

Figure 2.2: MACsec details

2.4.1 Wireshark

To test network connectivity between namespaces and capture traffic, you can use Wireshark to monitor packets in ns2 while sending ping requests from ns1. Here's a step-by-step example of how to do this.

Step 1: Start Wireshark in ns2 to Capture Traffic

First, open a new terminal window, start a bash session in the ns2 namespace and start Wireshark on the `macsec0` interface to monitor it.

Run Wireshark on ns2

```
$ sudo ip netns exec ns2 bash
$ wireshark -i eth0 -k
```

- The **-i eth0** option specifies the interface to capture on.
- The **-k** option tells Wireshark to start capturing immediately.

Step 2: Ping ns2 from ns1

In a separate terminal, enter ns1. Then, assuming that ns2's eth0 interface has been assigned 10.0.0.2, send a ping request from ns1 to this IP address:

Ping ns2

```
$ ping 10.0.0.2
```

You should see output similar to this, indicating successful packet transmission:

```
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.065 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.069 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.075 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.067 ms
```

Figure 2.3: ns1 pinging ns2

Step 3: Observe the traffic in Wireshark

Go back to the Wireshark window in ns2 and look for the ICMP packets coming from ns1. Wireshark should display a sequence of **ICMP echo requests** and replies between ns1 and ns2. To have a better picture, here is the captured traffic with Wireshark:

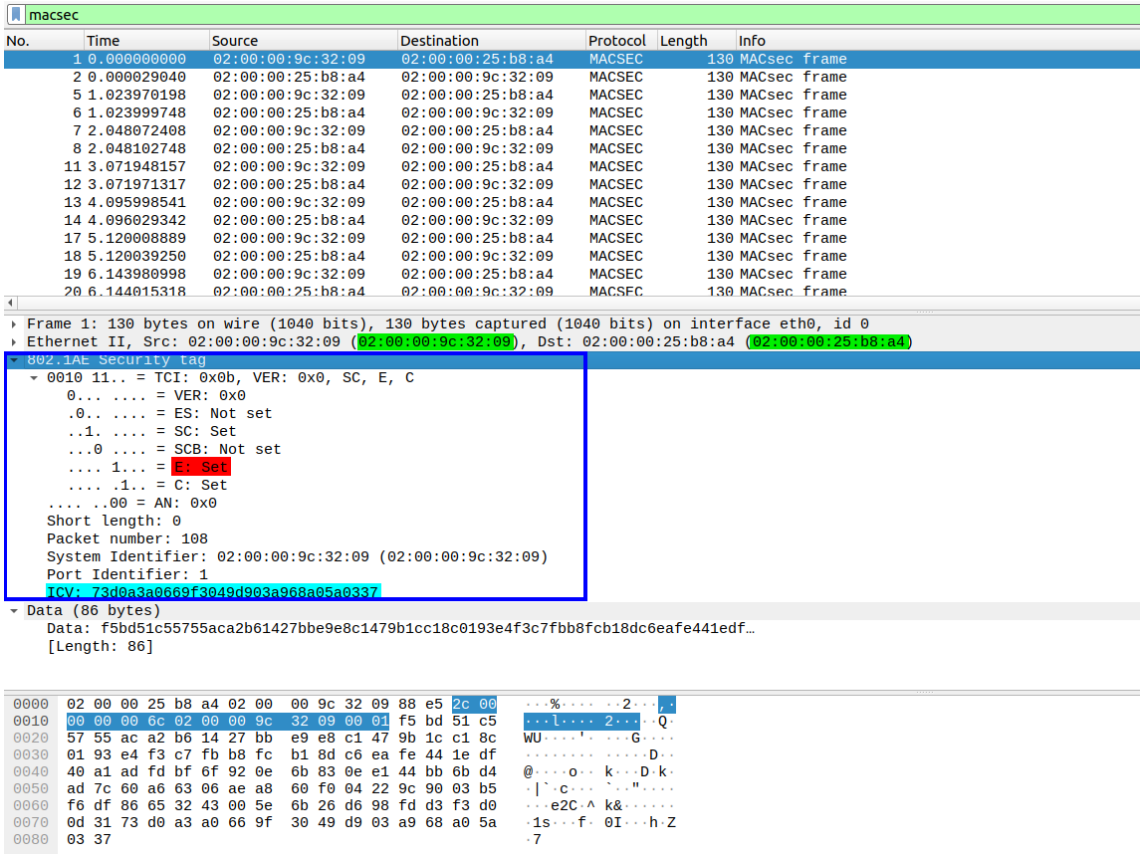

Figure 2.4: Traffic captured on ns2

Based on the capture, it's worth noting that only MACsec frames are present and they contain, as expected, additional fields:

- **802.1AE Security Tag (SecTAG)**
  - **TCI (Tag Control Information)**:
    - **T (Type of packet)**: `0b` (Indicates a MACsec frame)
    - **VER (Version)**: `0x0` (Initial MACsec version as per IEEE 802.1AE standards)
    - **ES (End Station bit)**: `0` (Not set)
    - **SC (Single Copy bit)**: `0` (Not set)
    - **SCB (Single-copy Broadcast)**: `0` (Not set)
    - **E (Encrypted bit)**: `1` (Indicates encrypted payload)
    - **C (Confidentiality offset)**: `1` (Indicates confidentiality is enabled)
  - **AN (Association Number)**: `0x0` (Identifier for the secure association key)
- **Packet Number**: `108` (Increments with each frame, helps prevent replay attacks)
- **System Identifier**: `02:00:00:9c:32:09` (MAC address of the source)
- **Port Identifier**: `1` (Indicates the port through which the packet was sent)
- **ICV (Integrity Check Value)**:
  - The cryptographic hash ensuring packet integrity: `73d0a3a0a66f30d99d03a968a05a037e`
- **Data Field**:
  - Contains 86 bytes of encrypted data.

As we can see the length of the frame is 130 bytes due to:

- **Ethernet Header**: 14 bytes
- **802.1AE MACsec Header (SecTAG)**: 16 bytes (though default is 8 bytes, here it includes the optional Secure Channel Identifier (SCI) encoding which adds an extra 8 bytes).
  Beware: the size of the Security TAG here is displayed as 14-bit field and the reason can be found in the note below!
- **ICV**: 16 bytes
- **Payload**: 86 bytes. It includes the IPv4 header (20 bytes), the ICMP header (8 bytes) and the ICMP payload (56 bytes on Linux), totaling 84 bytes. **But why is the size displayed as 86 bytes?**

  Note

  In a MACsec-protected frame, the 2 bytes corresponding to the MACsec EtherType (0x88e5) are taken from the Security TAG and placed in the Ethernet II frame header, replacing the usual EtherType carrying an IPv4 packet (0x0800). This reduces the Security TAG size from **16 bytes to 14 bytes**. Then, the 2 bytes that would normally hold the original EtherType are prepended to the MACsec-protected payload, which is then encrypted.

  After decryption, these 2 bytes are restored to the Ethernet frame as the original EtherType (0x0800), while the decrypted payload now consists of the expected 84 bytes carrying the ICMP echo request (20 bytes for the IP header, 8 bytes for the ICMP header, and 56 bytes for the ICMP payload).

To view the unencrypted traffic, we need to directly inspect the `macsec0` interface. By examining `macsec0`, we can observe the unencrypted payloads before they are processed through the encryption layer for secure transmission. This is essential for verifying and troubleshooting traffic in its original, unencrypted form.

The latest version of Wireshark now allows decryption of the encrypted MACsec payload, enabling further dissection of the traffic as if it were unencrypted. To enable this feature, go to `Edit > Preferences > Protocols > MACsec`, then enter the MACsec Pre-shared Key (the encryption key used to secure the traffic).
This key, known as the **SAK**, can be found in the `wpa_supplicant.log` file. Use the following command to locate it:

Get SAK

```
$ cat wpa_supplicant_ns1.log | grep "AES Key Unwrap of SAK"
```

The key will appear in hexadecimal format with spaces. Remove these spaces before pasting it into Wireshark. Once entered, you'll be able to view the decrypted payload, including the ICMP Echo Request.

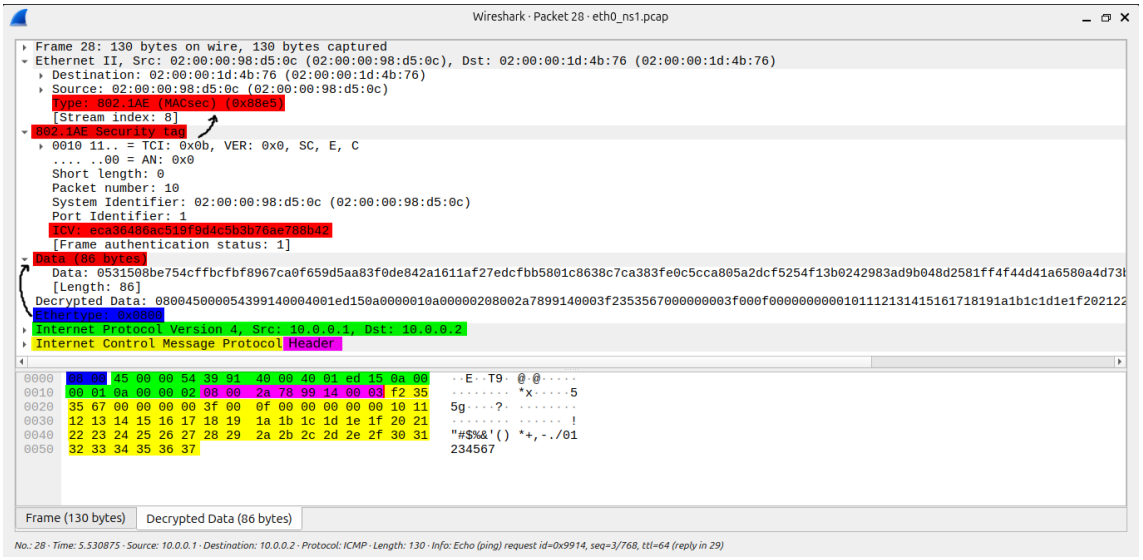

Figure 2.5: Decrypted MACsec payload in Wireshark

If the assumptions are correct, then when inspecting the `macsec0` interface (where the traffic is decrypted), we should find an Ethernet frame (14 bytes) plus the ICMP payload (84 bytes), totaling 98 bytes. As you can see in the image, this mateches the expected result.

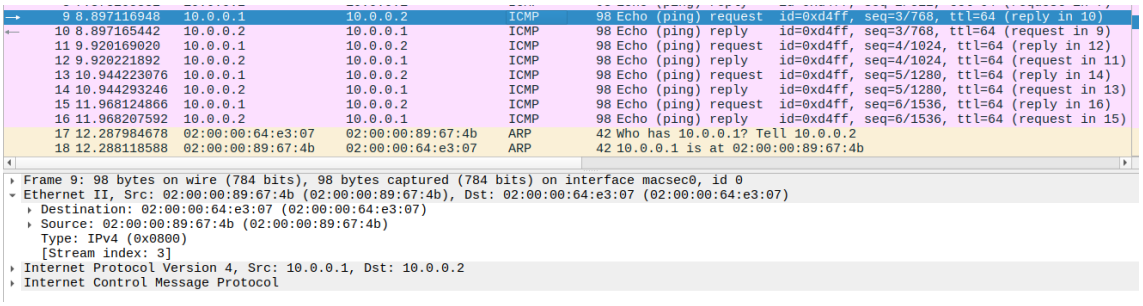

Figure 2.6: Inspection on macsec0 interface

Note: There is no need to manually set the proper MTU on the `macsec0` interface since it is already handled by `wpa_supplicant`.

2.4.2 Traffic from non-MACsec node towards a MACsec node

When a non-MACsec node tries to communicate with a MACsec-enabled node, the MACsec security mechanisms cause any traffic that does not follow the MACsec security mechanisms to be dropped.

Let's suppose that the non-MACsec node `ns3` sends an ICMP Echo Request toward the MACsec node `ns1`, assuming normal and unencrypted communication. The ICMP Echo Request is received by the `eth0` interface of the MACsec node. Because the MACsec node has MACsec enabled, it expects all inboud traffic to be compliant with MACsec standards. Specifically, each frame should have a MACsec Security Tag (SecTAG), which indicates that the frame is encrypted (optionally) and authenticated. Since each frame received does not contain a SecTAG, then it is identified as "non-secure" traffic. The `macsec0` interface is configured to drop any packet that does not have a valid SecTAG. Therefore, all the frames coming from `ns3` are dropped without reaching any higher layers.

By running the following command on the MACsec-enabled namespace `ns2` you can monitor live statistics for the `macsec0` interface, updating the output in real time.

Monitor macsec0 interface

```
$ watch ip -s macsec show
```

```
3: macsec0: protect on validate strict sc off sa off encrypt on send_sci on end_station off scb off replay off
    cipher suite: GCM-AES-128, using ICV length 16
    TXSC: 020000ec23910001 on SA 0
    stats: OutPktsUntagged InPktsUntagged OutPktsTooLong InPktsNoTag InPktsBadTag InPktsUnknownSCI InPktsNoSCI InPktsOverrun
                        0              0              0        1198              0              0           0            0
    stats: OutPktsProtected OutPktsEncrypted OutOctetsProtected OutOctetsEncrypted
                        0             27              0             1326
        0: PN 28, state on, key 57186a0b5987522eebb3591501000000
    stats: OutPktsProtected OutPktsEncrypted
                        0             27
    RXSC: 020000a8e4fc0001, state on
    stats: InOctetsValidated InOctetsDecrypted InPktsUnchecked InPktsDelayed InPktsOK InPktsInvalid InPktsLate InPktsNotValid InPktsNotUsingSA InPktsUnusedSA
                        0            966              0            0       15            0          0             0                0              0
        0: PN 16, state on, key 57186a0b5987522eebb3591501000000
    stats: InPktsOK InPktsInvalid InPktsNotValid InPktsNotUsingSA InPktsUnusedSA
                15            0              0               0              0
    offload: off
```

Figure 2.7: Inbound Packets with No Security Tag

When the non-MACsec node, `ns3`, attempts to reach the MACsec-enabled node, `ns2`, it is immediately noticeable that the **InPktsNoTag** field increments. This field, standing for "Inbound Packets with No Security Tag" is useful for troubleshooting unprotected traffic that appears on a secure network.

By initiating a Wireshark capture on `eth0`, you can observe unencrypted traffic arriving at the interface. However, this traffic does not reach the `macsec0` interface, indicating that packets lacking MACsec protection are filtered out before reaching the secure MACsec layer.

Note

A similar behavior can be noticed in the opposite direction. When the MACsec node initiates an ICMP Echo Request toward the non-MACsec node, the request is wrapped with a SecTAG and encrypted as per MACsec standards. Since the frame is encrypted and contains a SecTAG, the non-MACsec node has no means to decrypt or authenticate it, as it lacks MACsec capabilities and the necessary keys. As a result, the MACsec frames will be dropped and the communication will fail.

2.5 Teardown

After completing the simulation, use the following commands to gracefully bring down the network interfaces and remove all namespaces created for this simulation. These commands ensure that all network resources are released and that the environment is reset to its original state.

Stop wpa_supplicant

```
$ sudo systemctl stop wpa_supplicant
```

Bring down interfaces and delete namespaces

```
# Bring down loopbacks and veths interfaces for all the namespaces
$ sudo ip netns exec ns_n ip link set dev lo down
$ sudo ip link set dev veth_n down
$ sudo ip netns exec ns_n ip link set dev eth0 down

# Delete the namespaces
$ sudo ip netns delete ns_n
```

Substitute *_n with the desired number of the namespace/veth to teardown.

Bring down labnet bridge

```
$ sudo ip link set dev labnet down
$ sudo ip link set veth1 nomaster
$ sudo ip link set veth2 nomaster
$ sudo ip link set veth3 nomaster
$ sudo ip link delete labnet type bridge
```

2.6 Automated bash script for LAN simulation

This part of the document provides instructions on downloading, setting up, and running a Bash script that creates network namespaces with MACsec protocol enabled. The script automates namespace and interface configuration, MACsec protocol setup, logging, and provides options for teardown and namespace access.

Downloading the script

To begin, download the MACsec setup script from GitHub. Clone the repository and navigate to the script directory:

Clone Repository

```
$ git clone https://github.com/eferollo/MACsec.git
$ cd MACsec/src
```

Ensure the script is executable. If it is not, change its permissions:

Make Script Executable

```
$ chmod +x ./macsec.sh
```

Running the Script

To execute the script, use the following command:

Run Script

```
$ sudo ./macsec.sh
```

Specify the Number of Namespaces

After starting the script, you will be prompted to enter the number of namespaces to create. Input the desired number (e.g., 2 for two namespaces):

First Script Input

```
Enter the number of network namespaces: 2
```

Open Shell in Each Namespace

The script will then ask if you would like a `konsole` terminal window for each namespace. This allows direct interaction within each namespace:

Example Console Prompt

```
Do you want to open a shell in each namespace? (y/n): y
```

Entering `y` opens a `konsole` window for each namespace. Selecting `n` will skip this, but instructions to manually access each namespace are provided.
Note: `konsole` package is required to be installed on your machine.

Starting MACsec and MKA

Once the namespaces and interfaces are set up, the script will prompt you to press `Enter` to initialize the MACsec and MKA (Key Agreement) protocols:

Start MACsec Prompt

```
Simulating MACsec environment, press enter to start MKA and MACsec...
```

Viewing Logs and Namespace Details

The script includes a menu for managing namespaces and logs. Upon running the script, a `log` directory is automatically created in the current working directory. This directory stores various logs generated during the setup and operation of the network namespaces and MACsec configuration. Below is a description of each log type you will find:

- **WPA Supplicant Logs** (`wpa_supplicant_nsX.log`): These logs are generated by `wpa_supplicant` for each namespace (where `X` is the namespace number). They provide detailed debug information about the MACsec Key Agreement (MKA) protocol operations, security association setups, and any issues encountered by `wpa_supplicant`.
- **Interface Capture Logs** (`eth0_nsX.pcap`): For each namespace, a `.pcap` file is generated to capture packets on the `eth0` interface. These captures include traffic using MACsec encryption protocols, specifically focusing on frames with EtherTypes `0x888E` (EAPOL frames) and `0x88E5` (MACsec secured data). You can analyze these with tools such as Wireshark or `tcpdump`.
- **RTNETLINK Log** (`rtnetlink.log`): This log tracks changes to network interfaces and addresses in the system using `rtnetlink`. It monitors all network events across namespaces, providing insight into link, address, and route changes crucial for network troubleshooting.

The logs are automatically rotated to prevent excessive disk usage. Each log type is limited to a maximum of 10 rotated files, ensuring older logs are archived while keeping recent information readily available.
Note: `logrotate` package is required to be installed on your machine.

To view any of these logs, navigate to the `log` directory created within the directory where the script is executed:

List Namespaces and Show MACsec Context

To list all namespaces and view MACsec information, use the options below:

List and Show MACsec Context

```
Select an option: list
Select an option: show
```

Teardown the Setup

When done, select `shutdown` from the menu. This will:

- Stop all running `wpa_supplicant` processes.
- Remove all network interfaces.
- Delete namespaces and the bridge.

Note: Manual Namespace Access

If you skipped the console option, you can manually access each namespace as follows:

Manual Namespace Access

```
$ sudo ip netns exec ns1 bash
```

Chapter 3

MACsec WAN Simulation

In this chapter, you will explore the process of simulating a MACsec deployment over a **WAN** (Wide Area Netowork) environment. Although MACsec is designed to secure communication on Local Area Networks, it can also extend security benefits to WANs, allowing secure Ethernet communication across geographically separated networks. This simulation will demonstrate how to configure MACsec over a WAN tunnel using Linux network **namespaces**. You will set up a secure **Layer 2 GRE tunnel** protecting the traffic between two remote sites, over WAN or Internet, like a **site-to-site VPN at Layer 2**.

Why MACsec over a GRETAP tunnel?

Adding MACsec to a **GRETAP** tunnel is particularly useful when different sites need to be securely connected over Layer 2 VPNs. With the increasing use of geographically distributed data centers and remote sites, many organizations rely on Layer 2 VPNs to extend network connectivity over WANs, often using protocols like GRETAP to encapsulate Ethernet frames for seamless Layer 2 communication across IP networks.

However, traditional Layer 2 VPN solutions, like GRE alone, lack encryption and leave data vulnerable to interception over the WAN. MACsec adds an essential layer of security by encrypting Ethernet frames at Layer 2, which ensures data confidentiality, integrity and data origin authentication even as it traverses untrusted WAN links. By enabling MACsec over GRETAP, organizations can achieve secure, high-throughput Layer 2 communication without compromising on latency or scalability.

For this lab, the setup chosen is based on a **GNU/Linux** system (Debian-based) and the requirements are identical to those of the previous simulation.

Requirements

In order to make this simulation work, the following packages must be installed and up to date:

- Linux kernel $\geq$ 4.7.0
- Netlink library libnl $\geq$ 3.2.29
- wpa_supplicant $\geq$ 2.6
- ip-route2

3.1 Scenario

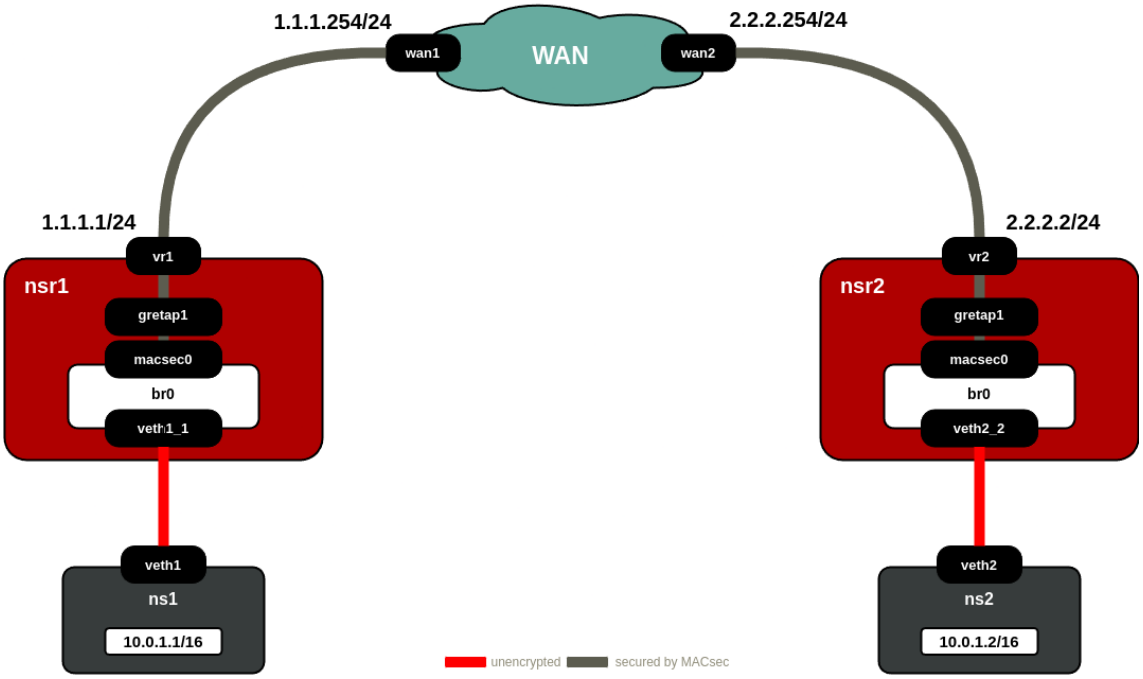

Figure 3.1: MACsec WAN simulation scenario

In this simulation, multiple namespaces and interfaces are configured to create a secure, bridged environment over a WAN. Namespaces `ns1` and `ns2` act as two endpoint **hosts**, connected to their respective **router** namespaces (`nsr1` and `nsr2`) via virtual Ethernet pairs (`veth1` ↔ `veth1_1` and `veth2` ↔ `veth2_2`). The router namespaces connect to the `wan` namespace (simulating the **WAN**) through additional virtual Ethernet pairs (`vr1` ↔ `wan1` and `vr2` ↔ `wan2`). Within each router namespace, bridge interfaces (`br0`) connect the host interfaces and MACsec-protected GRE tunnel (`macsec0` over `gretap1`), allowing secure Layer 2 traffic to flow between `host1` and `host2` across the WAN.

This setup uses GRETAP to encapsulate Ethernet frames and MACsec to encrypt them, bridging traffic securely between both endpoints. The `vr1` and `vr2` interfaces, connecting `nsr1` and `nsr2` to `wan1` and `wan2` in the `wan` namespace, allow these routers to route and forward encrypted data across the WAN, creating a seamless, secure Layer 2 link between sites.

3.2 Initial Setup

As first step, create 2 network namespaces that will act as endpoint **hosts** and connect them to the simulated routers through their virtual Ethernet pairs.

Create host namespaces and virtual ethernet pairs

```
$ sudo ip netns add ns1
$ sudo ip netns add ns2
$ sudo ip link add veth1 type veth peer name veth1_1
$ sudo ip link add veth2 type veth peer name veth2_2
$ sudo ip link set veth1 netns ns1
$ sudo ip link set veth2 netns ns2
```

Now set specific MAC addresses and IP addresses for `veth1` and `veth2` by ensuring predictable addresses for both hosts. Then bring up all the interfaces.

Assign IP and MAC addresses for hosts

```
$ sudo ip netns exec ns1 ip link set veth1 address 02:00:00:00:01:01
$ sudo ip netns exec ns1 ip address add 10.0.1.1/16 dev veth1
$ sudo ip netns exec ns1 ip link set veth1 up
$ sudo ip netns exec ns2 ip link set veth2 address 02:00:00:00:01:02
$ sudo ip netns exec ns2 ip address add 10.0.1.2/16 dev veth2
$ sudo ip netns exec ns2 ip link set veth2 up
```

In the next step create the **router** namespaces, move the virtual ethernet pair into the router namespaces and bring the interfaces up.

Create router namespaces

```
$ sudo ip netns add nsr1
$ sudo ip netns add nsr2
```

Assign and bring up interfaces within each router

```
$ sudo ip link set veth1_1 netns nsr1
$ sudo ip link set veth2_2 netns nsr2
$ sudo ip netns exec nsr1 ip link set veth1_1 up
$ sudo ip netns exec nsr2 ip link set veth2_2 up
```

Now it's the time to set up the WAN namespace, assign the `wan1` and `wan2` interfaces that connect each router, and enable IP forwarding.

Setup WAN and enable routing

```
# Create namespace and configure veth pairs
$ sudo ip netns add wan
$ sudo ip link add vr1 type veth peer name wan1
$ sudo ip link add vr2 type veth peer name wan2
$ sudo ip link set vr1 netns nsr1
$ sudo ip link set vr2 netns nsr2

# Assign interfaces
$ sudo ip link set wan1 netns wan
$ sudo ip link set wan2 netns wan

# Enable routing
$ sudo ip netns exec wan sysctl -w net.ipv4.ip_forward=1
```

The following commands create a GRETAP tunnel, allowing Layer 2 traffic between `nsr1` and `nsr2`. This setup encapsulates Ethernet frames in GRE headers, which the WAN can forward transparently.

Configure GRETAP tunnel

```
$ sudo ip netns exec nsr1 ip link add gretap1 type gretap local 1.1.1.1 remote
2.2.2.2
$ sudo ip netns exec nsr2 ip link add gretap1 type gretap local 2.2.2.2 remote
1.1.1.1

$ sudo ip netns exec nsr1 ip link set gretap1 address 02:00:00:11:11:11
$ sudo ip netns exec nsr1 ip link set gretap1 up

$ sudo ip netns exec nsr2 ip link set gretap1 address 02:00:00:22:22:22
$ sudo ip netns exec nsr2 ip link set gretap1 up
```

By layering MACsec on top of GRE tunnel, we enable all the security properties provided by MACsec on layer 2 traffic.

Enable MACsec on the GRETAP tunnel

```
$ sudo ip netns exec nsr1 ip link add gretap1 type gretap local 1.1.1.1 remote
2.2.2.2
$ sudo ip netns exec nsr2 ip link add gretap1 type gretap local 2.2.2.2 remote
1.1.1.1

$ sudo ip netns exec nsr1 ip link set gretap1 address 02:00:00:11:11:11
$ sudo ip netns exec nsr1 ip link set gretap1 up

$ sudo ip netns exec nsr2 ip link set gretap1 address 02:00:00:22:22:22
$ sudo ip netns exec nsr2 ip link set gretap1 up
```

Now, you need to configure the Security Associations (SAs). These commands set up transmit (tx) and receive (rx) security associations with keys on both routers, ensuring bidirectional encryption over the GRETAP tunnel.

First, let's generate some keys. We need a key for each router's transmit channel. That same key must be configured on the peer's matching receive channel.

This command will generate 16 bytes keys, in a hexadecimal format suitable to configure with the ip command.

Random hex bytes

```
$ dd if=/dev/urandom count=16 bs=1 2>/dev/null | hexdump | cut -c 9- | tr -d '\n '
```

On the router of the first site:

Configure MACsec SA on nsr1

```
# Add a transmit (tx) Secure Association (SA) for outgoing packets
$ sudo ip netns exec nsr1 ip macsec add macsec0 tx sa 0 pn 1 on key 01
6564e1d555c6db6656bea4aa5af2c6f6

# Add a receive (rx) association, identifying the peer MAC address that nsr1 will accept
packets from
$ sudo ip netns exec nsr1 ip macsec add macsec0 rx address 02:00:00:22:22:22 port 1

# Configure an SA for receiving packets from the specified address with a unique rx key
$ sudo ip netns exec nsr1 ip macsec add macsec0 rx address 02:00:00:22:22:22 port 1 sa 0
pn 1 on key 02 08b0f764ee2dfaa992e02e410d7eca66
```

Similarly on the router of the second site:

Configure MACsec SA on nsr2

```
# Set a transmit SA on nsr2's macsec0 interface for outgoing packets, with a matching key to nsr1's receive SA
$ sudo ip netns exec nsr2 ip macsec add macsec0 tx sa 0 pn 1 on key 02 08b0f764ee2dfaa992e02e410d7eca66

# Define an rx association on nsr2 for accepting packets from nsr1
$ sudo ip netns exec nsr2 ip macsec add macsec0 rx address 02:00:00:11:11:11 port 1

# Configure an SA for nsr2 to decrypt packets from nsr1 using the correct rx key
$ sudo ip netns exec nsr2 ip macsec add macsec0 rx address 02:00:00:11:11:11 port 1 sa 0 pn 1 on key 01 6564e1d555c6db6656bea4aa5af2c6f6
```

As final step you need to set up the **bridge** for the hosts. Each router namespace (`nsr1` and `nsr2`) creates a bridge interface (`br0`) to link their host connections (`veth1_1` and `veth2_2`) with the MACsec interface.

Set up Bridge

```
$ sudo ip netns exec nsr1 ip link add br0 type bridge
$ sudo ip netns exec nsr1 ip link set veth1_1 master br0
$ sudo ip netns exec nsr1 ip link set macsec0 master br0
$ sudo ip netns exec nsr1 ip link set br0 up

$ sudo ip netns exec nsr2 ip link add br0 type bridge
$ sudo ip netns exec nsr2 ip link set veth2_2 master br0
$ sudo ip netns exec nsr2 ip link set macsec0 master br0
$ sudo ip netns exec nsr2 ip link set br0 up
```

This bridge setup enables seamless Layer 2 connectivity, so host `ns1` and host `ns2` can communicate as if they are on the same network, secured by MACsec over the GRETAP tunnel.

3.3 Analysis

Once the testing environment has been successfully configured, you may want to have a quick look at the traffic protected by MACsec and make sure that everything is working as expected. By following the same instructions explained in the previous simulation, you can verify the status of each namespace and its interface and perform a packet capture with `tcpdump` or `Wireshark` in the **wan** network namespace, as if someone in the Internet is trying to sniff the traffic.

Capture traffic on WAN namespace

```
# Access WAN namespace
$ sudo ip netns exec wan bash

# Start capture on wan1 interface
$ sudo wireshark -i wan1 -k
```

Generate some traffic from the host `ns1` toward the host `ns2`. For semplicity, access the host `ns1` namespace (10.0.1.1) and send some ICMP Echo Requests to the host `ns2` (10.0.1.2).

Ping ns2

```
# Access ns1 namespace
$ sudo ip netns exec ns1 bash

# Ping ns2
$ ping 10.0.1.2
```

```
PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data.
64 bytes from 10.0.1.2: icmp_seq=1 ttl=64 time=0.233 ms
64 bytes from 10.0.1.2: icmp_seq=2 ttl=64 time=0.151 ms
64 bytes from 10.0.1.2: icmp_seq=3 ttl=64 time=0.138 ms
64 bytes from 10.0.1.2: icmp_seq=4 ttl=64 time=0.148 ms
64 bytes from 10.0.1.2: icmp_seq=5 ttl=64 time=0.152 ms
64 bytes from 10.0.1.2: icmp_seq=6 ttl=64 time=0.121 ms
64 bytes from 10.0.1.2: icmp_seq=7 ttl=64 time=0.121 ms
64 bytes from 10.0.1.2: icmp_seq=8 ttl=64 time=0.148 ms
^C
--- 10.0.1.2 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7200ms
rtt min/avg/max/mdev = 0.121/0.151/0.233/0.033 ms
```

Figure 3.2: ns1 pinging ns2

3.3.1 Wireshark

After pinging `ns2`, go back to the Wireshark window in `ns2` and look for the ICMP packets coming from `ns1`. Wireshark would show multiple MACsec packets carrying both ARP and ICMP data. Remembeer that MACsec protects all Layer 2 traffic, including broadcast.

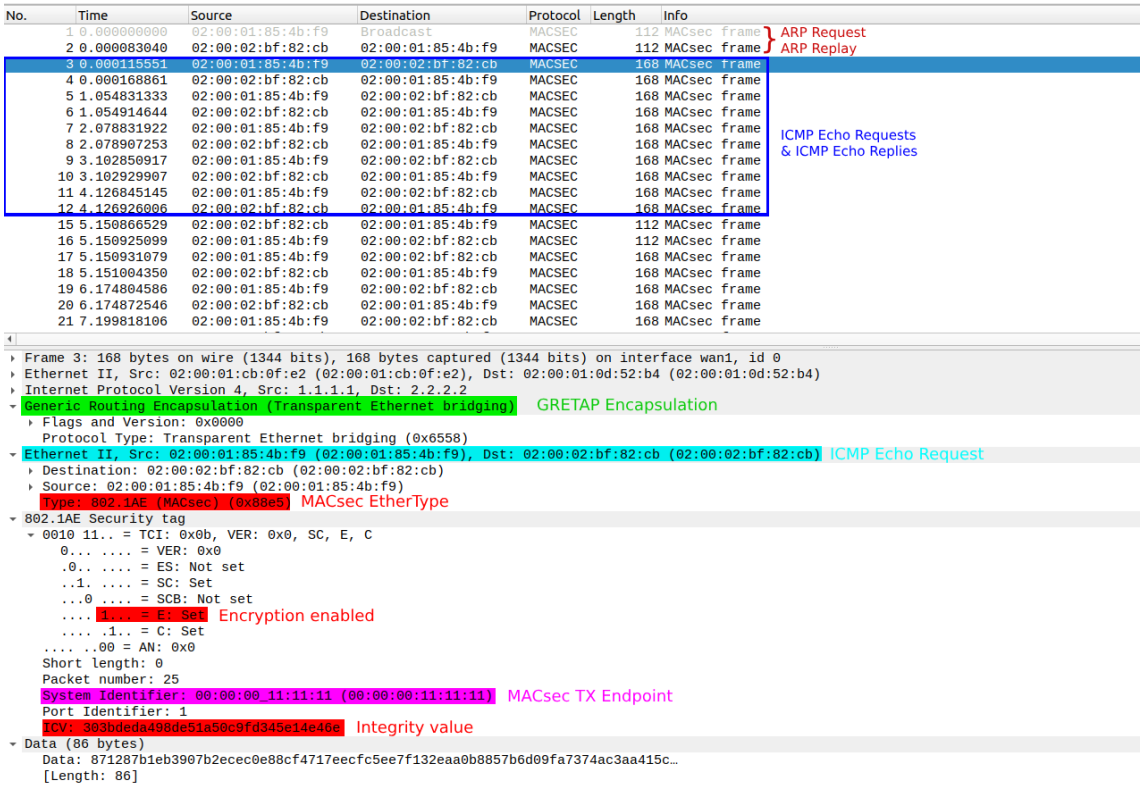

Figure 3.3: Traffic captured on ns2

Looking at the capture you can immediately notice that all the traffic coming from the host `ns1` is MACsec protected.

Taken the ICMP Echo Request frame as an example we can see additional fields between the original Ethernet header and the payload.

- **Outer Ethernet II Frame**. This is the encapsulating Ethernet frame that contains the MACsec-protected payload.
  - **Source MAC Address:** 02:00:01:cb:0f:e2
  - **Destination MAC Address:** 02:00:01:0d:52:b4
  - **Type:** 0x0800 (IPv4)
- **IPv4 Packet**. This IPv4 packet is encapsulated within the Ethernet frame and acts as the transport layer for the GRETAP (Generic Routing Encapsulation for Transparent Ethernet Bridging) traffic.
  - **Source IP Address:** 1.1.1.1
  - **Destination IP Address:** 2.2.2.2
- **Generic Routing Encapsulation (GRE) - Transparent Ethernet Bridging**. This GRE layer enables encapsulation of Ethernet frames, allowing for Layer 2 forwarding across Layer 3 networks. In this case, GRE is used for Transparent Ethernet Bridging, meaning it can carry Ethernet frames as its payload.
  - **Flags and Version:** 0x0000 (default value with no special flags)
  - **Protocol Type:** 0x6558 (Transparent Ethernet Bridging), indicating Transparent Ethernet Bridging, meaning that an Ethernet frame follows within the GRE payload.

- **Inner Ethernet Frame (Encapsulated in GRE)**. This Ethernet frame, encapsulated within GRE, includes the actual MACsec-protected traffic.
  - **Source MAC Address:** 02:00:01:85:4b:f9
  - **Destination MAC Address:** 02:00:01:bf:82:cb
  - **Type:** 0x88E5 (MACsec)
- **MACsec Layer (802.1AE)**. The MACsec (802.1AE) layer includes the already known (seen before) security fields:
  - **802.1AE Security Tag:**
    - **TCI (Tag Control Information):** 0x0b
      - **SC (Secure Channel):** Unique identifier for the secure communication channel
      - **E (End):** Indicates end of a protected data unit
      - **C (Confidentiality):** Confidentiality protection is enabled
    - **AN (Association Number):** 0x0
    - **Packet Number:** 25 (provides replay protection)
    - **System Identifier:** 00:00:00:11:11:11
    - **Port Identifier:** 1
  - **Integrity Check Value (ICV):** 303bdeda498de51a506370345e14e46e (16 bytes)
- **Protected Data (Encrypted Payload)**. The actual encrypted payload data, representing the content protected by MACsec (potentially containing the original ICMP echo request data). The data length and content appear encrypted and are protected for confidentiality.
  - **Data (86 bytes):** 87128701eb3907b2ecec0e88cf4717eecf...

3.3.2 MTU

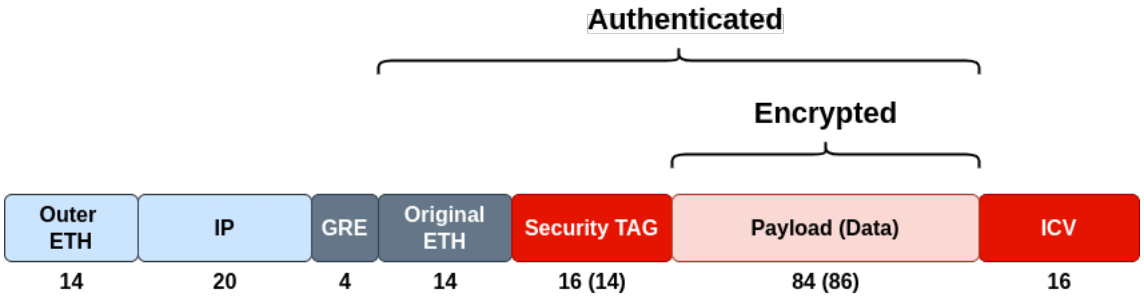

Figure 3.4: MACsec over GRETAP frame overview

When tunneling is in use, it's crucial to account for the MTU. In this scenario, multiple encapsulation layers add extra overhead, requiring a reduction in the MTU on the hosts. Let's calculate the overhead introduced by each layer:

- **Original Ethernet**: 14 bytes (since it is now transparently bridged over the WAN)
- **MACsec**: 32 bytes
- **GRETAP**: 4 bytes
- **IP Header**: 20 bytes

Total Overhead:

70 bytes

Starting with an original MTU of 1500, subtracting this overhead gives:

$$\text{MTU} = 1500 - 70 = 1430$$

To verify the Path MTU, we can send an ICMP packet from host `ns1` with the DF (Don't Fragment) bit set. If our calculations are correct, the largest ICMP payload that can pass without fragmentation is:

$$1430 - 20 \text{ (IP header)} - 8 \text{ (ICMP header)} = 1402$$

Thus, any packet larger than 1402 bytes will be dropped.

In order to test this, run the following commands on the `ns1` namespace.

Test MTU size

```
$ sudo ip netns exec ns1 ping -M do -c4 10.0.1.2 -s 1402

$ sudo ip netns exec ns1 ping -M do -c4 10.0.1.2 -s 1403
```

Note

When analyzing the packet size, you'll notice that even with a size of 1514 bytes, no fragmentation occurs. This is due to Ethernet's handling of frame sizes and the specifics of Layer-2 tunneling with MACsec and GRE. Here's a breakdown:

1. **Ethernet MTU and Frame Size**: Ethernet typically allows a maximum MTU of 1500 bytes for payload data. However, the total Ethernet frame size (payload + headers) can extend up to 1518 bytes, accounting for:
   - 14 bytes for the Ethernet header (Source MAC, Destination MAC, and EtherType).
   - 1500 bytes for the actual payload.
   - 4 bytes for the Frame Check Sequence (FCS) trailer, which is often omitted in packet captures.

   This is why a frame size of up to 1518 bytes can pass without fragmentation in a standard Ethernet network.
2. **Impact of MACsec Over GRE (GRETAP) Overhead**: In this setup, the additional MACsec, GRE, and IP encapsulation add 70 bytes of overhead. This reduces the effective MTU for the payload to 1430 bytes. To avoid fragmentation, the ICMP payload must be adjusted to fit within this reduced MTU. With 70 bytes of overhead and 28 bytes for the IP and ICMP headers, the maximum ICMP data size you can send is 1402.
3. **Layer-2 Tunneling Behavior**: Since this setup uses Layer-2 tunneling (GRETAP), packets remain encapsulated at Layer-2 without involving Layer-3 IP routing within the tunnel. This means:
   - Oversized frames are simply dropped within the tunnel without sending back a "Message too long" (ICMP Fragmentation Needed) response, which typically comes from a Layer-3 device.
   - Frames exceeding the MTU limit are silently discarded, providing no feedback to the sender.
4. **MTU Adjustment Options**:
   - **Decrease MTU on the Client Side**: One approach is to reduce the MTU on client devices, ensuring that the packets they send fit within the encapsulated MTU limit, avoiding oversized frames that would be dropped in the tunnel.
   - **Increase MTU on the WAN Path**: If you control the WAN infrastructure, you could increase the MTU on the WAN links to accommodate the extra encapsulation overhead. This would allow for a larger packet size to pass through without fragmentation. However, if the WAN path traverses the public internet, increasing the MTU isn't feasible, as you don't control the underlying infrastructure.

As you can see below every packet with an MTU size bigger than 1402 bytes is dropped.

```
           :~/MACsec/src$ sudo ip netns exec ns1 ping -M do -c4 10.0.1.2 -s 1402
PING 10.0.1.2 (10.0.1.2) 1402(1430) bytes of data.
1410 bytes from 10.0.1.2: icmp_seq=1 ttl=64 time=0.388 ms
1410 bytes from 10.0.1.2: icmp_seq=2 ttl=64 time=0.278 ms
1410 bytes from 10.0.1.2: icmp_seq=3 ttl=64 time=0.334 ms
1410 bytes from 10.0.1.2: icmp_seq=4 ttl=64 time=0.272 ms

--- 10.0.1.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3098ms
rtt min/avg/max/mdev = 0.272/0.318/0.388/0.047 ms
           :~/MACsec/src$ sudo ip netns exec ns1 ping -M do -c4 10.0.1.2 -s 1403
PING 10.0.1.2 (10.0.1.2) 1403(1431) bytes of data.

--- 10.0.1.2 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3106ms
```

Figure 3.5: Test MTU overhead

Taking in mind the overhead given by MACsec over GRETAP, let's set the proper MTU (1430) on the host namespaces.

Change MTU

```
$ sudo ip netns exec ns1 ip link set veth1 mtu 1430
$ sudo ip netns exec ns2 ip link set veth2 mtu 1430
```

3.4 Teardown

After completing the simulation, paste the following commands to gracefully bring down all the network interfaces and remove all the namespaces configured before. These commands ensure that all network resources are released and that the environment is reset to its original state.

Bring down interfaces

```
$ sudo ip netns exec ns1 ip link set veth1 down
$ sudo ip netns exec ns2 ip link set veth2 down
$ sudo ip netns exec nsr1 ip link set veth1_1 down
$ sudo ip netns exec nsr2 ip link set veth2_2 down
```

Bring down virtual router and WAN interfaces

```
$ sudo ip netns exec nsr1 ip link set vr1 down
$ sudo ip netns exec nsr2 ip link set vr2 down
$ sudo ip netns exec wan ip link set wan1 down
$ sudo ip netns exec wan ip link set wan2 down
```

Disables GRE and MACsec interfaces to stop secure tunneling and encapsulation connections.

Bring down GRETAP and MACsec interfaces

```
$ sudo ip netns exec nsr1 ip link set gretap1 down
$ sudo ip netns exec nsr2 ip link set gretap1 down
$ sudo ip netns exec nsr1 ip link set macsec0 down
$ sudo ip netns exec nsr2 ip link set macsec0 down
```

Bring down and delete the bridges

```
$ sudo ip netns exec nsr1 ip link set br0 down
$ sudo ip netns exec nsr2 ip link set br0 down
$ sudo ip netns exec nsr1 ip link del br0
$ sudo ip netns exec nsr2 ip link del br0
```

Delete MACsec and GRETAP

```
$ sudo ip netns exec nsr1 ip link del macsec0
$ sudo ip netns exec nsr2 ip link del macsec0
$ sudo ip netns exec nsr1 ip link del gretap1
$ sudo ip netns exec nsr2 ip link del gretap1
```

Delete network namespaces

```
$ sudo ip netns del ns1
$ sudo ip netns del ns2
$ sudo ip netns del nsr1
$ sudo ip netns del nsr2
$ sudo ip netns del wan
```

3.5 Automated bash script for WAN simulation

For the WAN simulation, as with the LAN simulation, it's possible to run an automated Bash script to create the environment for MACsec over WAN, replicating the setup that was previously deployed manually. This script automates the configuration of namespaces and interfaces, sets up the MACsec protocol and GRETAP, enables logging, and includes options for environment teardown and direct namespace access. The script allows you to run the setup directly with plain GRETAP tunneling without a MACsec context.

Downloading the script

To begin, download the MACsec setup script from GitHub. Clone the repository and navigate to the source directory.

Clone Repository

```
$ git clone https://github.com/eferollo/MACsec.git
$ cd MACsec/src
```

Once inside the src directory, ensure it is executable. If it is not, change its permissions:

Make Script Executable

```
$ sudo ./wan_macsec.sh
```

Running the Script and Choosing Options

After starting the script, you'll be prompted to make two key choices:

1. **Open a Shell in Each Namespace**: If you select "yes," the script opens a shell for each namespace in separate terminals. This allows you to interact directly with each namespace. **Note:** `konsole` package is required to be installed on your machine.
2. **Enable MACsec Encryption**: You'll have the option to toggle MACsec encryption. If enabled, MACsec will secure network communications between namespaces using randomly generated keys.

Logging and Log Files

The script creates detailed logs for each network interface within the namespaces. All log files are saved in the `wan_logs` directory, with specific subdirectories based on the namespace type:

- `wan_logs/hosts`: Logs for host namespaces `ns1` and `ns2`, which are created for each veth (virtual Ethernet) interface.
- `wan_logs/routers`: Logs for router namespaces `nsr1` and `nsr2`. Each router has logs for multiple interfaces, including veth, GRE, and (if enabled) MACsec.
- `wan_logs/wan`: Logs for WAN interfaces (`wan1` and `wan2`) in the WAN namespace.

Each subdirectory contains `.pcap` (packet capture) files for individual interfaces, which can be analyzed using tools like `tcpdump` or `Wireshark`.

Example Log Locations

- **Host Namespace Logs**:
  - `wan_logs/hosts/ns1_veth1.pcap` for the `veth1` interface of `ns1`
  - `wan_logs/hosts/ns2_veth2.pcap` for the `veth2` interface of `ns2`
- **Router Namespace Logs**:
  - `wan_logs/routers/nsr1_veth1_1.pcap` for the `veth1_1` interface of `nsr1`
  - `wan_logs/routers/nsr2_macsec.pcap` (if MACsec is enabled)
- **WAN Namespace Logs**:
  - `wan_logs/wan/wan1.pcap` for the `wan1` interface in the WAN namespace

The script includes a **log rotation** function that prevents the log files from growing indefinitely. The rotation policy retains the last 10 logs for each interface. You can configure log rotation settings in the configuration files generated by the script in each logging directory.
Note: `logrotate` package is required to be installed on your machine.

Monitoring and Analyzing Traffic

You can use `tcpdump` or `Wireshark` to analyze `.pcap` files. This analysis provides insight into:

- Traffic between hosts and routers.
- Encapsulated GRE tunnel packets.
- Encrypted traffic (if MACsec is enabled).

List Namespaces and Show MACsec Context

To list all namespaces and view MACsec information, use the options below:

List and Show MACsec Context

```
Select an option: list
Select an option: show
```

Teardown the Setup

When done, select `shutdown` from the menu. This will:

- Bring down the GRETAP tunnel and MACsec context.
- Remove all network interfaces.
- Delete namespaces and the bridge.

Note: Manual Namespace Access

If you skipped the console option, you can manually access each namespace as follows:

Manual Namespace Access

```
$ sudo ip netns exec ns1 bash
```

Chapter 4

Performance

This chapter evaluates the performance of MACsec over LAN and WAN scenarios. Performance testing was conducted with MACsec enabled and disabled. The evaluation metrics primarily include bandwidth over time, measured in Gbit/s, for the different configurations. This analysis reveals the impact of enabling MACsec on network performance and provides insights into its performance degradation and trade-offs.

4.1 Overview

All benchmark tests were conducted using `iperf3`, and each test was repeated multiple times to ensure the consistency and reliability of the results.

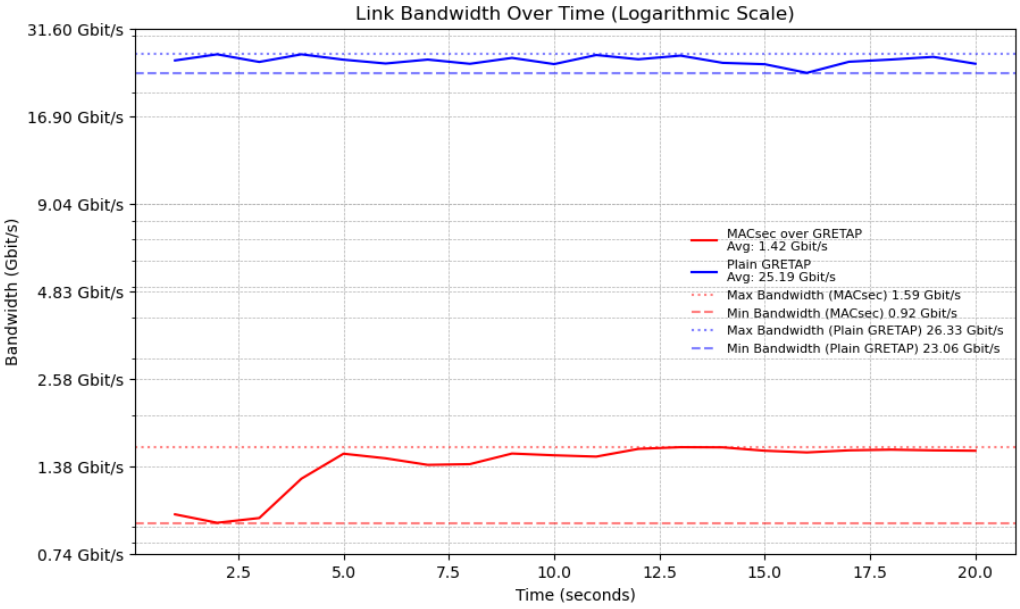

Figure 4.1: WAN performances

Plain GRE over WAN:

- **Average Bandwidth:** ~25.19 Gbit/s
- **Bandwidth Range:** 23.06–26.33 Gbit/s

MACsec over GRETAP:

- **Average Bandwidth:** ~1.42 Gbit/s
- **Bandwidth Range:** 0.92–1.59 Gbit/s

Key Observations:

- There is a significant bandwidth reduction when MACsec is enabled.
- Plain GRE achieves high and stable throughput in the emulated WAN environment, while MACsec introduces substantial cryptographic overhead, reducing bandwidth by approximately 94%.

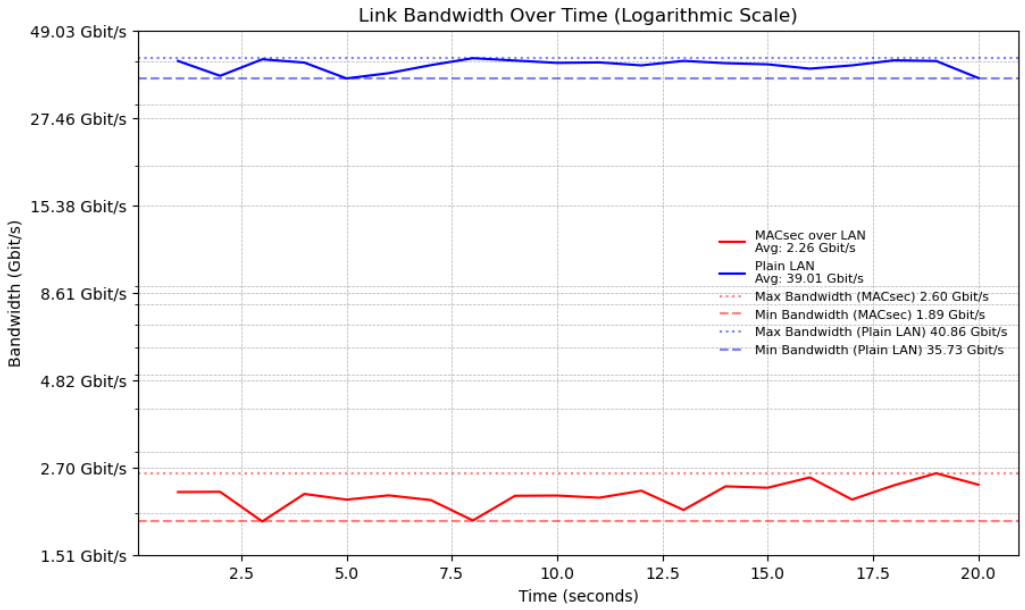

Figure 4.2: LAN performances

Plain LAN:

- **Average Bandwidth:** ∼39.01 Gbit/s
- **Bandwidth Range:** 35.73–40.86 Gbit/s

MACsec over LAN:

- **Average Bandwidth:** ∼2.26 Gbit/s
- **Bandwidth Range:** 1.89–2.60 Gbit/s

Key Observations:

- Plain LAN provides high throughput due to minimal overhead.
- With MACsec enabled, bandwidth decreases by a similar margin to the WAN case (∼94% reduction), though the raw throughput is slightly higher than the WAN configuration due to the absence of the additional layer of encapsulation given by GRE.

4.2 Considerations

Linux namespaces create virtualized environments where network interfaces, routing, and security protocols can be isolated and tested without external interference. While namespaces effectively emulate real-world conditions, the cryptographic operations for MACsec (e.g., encryption and integrity checks) in these virtualized setups likely rely on software-based processing. This results in high CPU usage and a significant performance hit, as seen in both WAN and LAN scenarios.

Generally speaking the LAN environment, compared to the WAN environment, exhibits higher baseline throughput due to reduced latency in the namespace configuration. GRE encapsulation combined with MACsec adds overhead in terms of both packet size and processing requirements.

However, the relative impact of MACsec remains consistent, suggesting that the performance bottleneck lies in the software implementation of MACsec, not the underlying network medium.

Within Linux namespaces, cryptographic operations are typically handled by the host CPU unless explicitly configured to use hardware acceleration. The steep drop in bandwidth with MACsec enabled implies that no **hardware offloading** (which MACsec is capable of) is utilized, resulting in significant performance degradation.

Overall MACsec demonstated to be a worth competitor of other technologies, such as **IPsec**, in terms of throughput and computational overhead when encrypting network traffic. Unlike IPsec, which operates at Layer 3 and typically involves significant overhead due to tunneling and complex protocols, MACsec operates at Layer 2 and this **lower-level implementation** results in minimal latency and higher throughput since it encrypts traffic directly on the link rather than across an entire end-to-end connection.

Bibliography

[1] "Ieee standard for local and metropolitan area networks-media access control (mac) security," *IEEE Std 802.1AE-2018 (Revision of IEEE Std 802.1AE-2006)*, pp. 1–239, 2018.

[2] "Ieee standard for local and metropolitan area networks–port-based network access control," *IEEE Std 802.1X-2010 (Revision of IEEE Std 802.1X-2004)*, pp. 1–205, 2010.