MACsec

Encryption for the wired LAN

Networking Services Team, Red Hat

Sabrina Dubroca
sd@queasysnail.net
sdubroca@redhat.com

2016-02-07, DevConf.cz 2016

Outline

- Introduction to MACsec (architecture, protocol, related standards)
- Linux kernel implementation
- Use cases with configuration examples
- Future work

1 Introduction

redhat.

What is MACsec

- IEEE standard (802.1AE-2006) for encryption over Ethernet
- Encrypt and authenticate all traffic in a LAN with GCM-AES-128

redhat.

Why MACsec

- Security within LANs (layer 2) is pretty bad
  - rogue DHCP/router advertisements
  - ARP/ndisc spoofing
- IPsec is L3, cannot protect ARP/ndisc on untrusted links
- Cloud environment: VXLAN
  - Encrypted VXLAN: encryption on the tunnel endpoints, not in the VM $\Rightarrow$ Tenant has no control over the keys
  - MACsec over VXLAN: encryption in the VM, doesn't need to be aware of the underlay network

redhat.

MACsec concepts, architecture, and definitions

Secure channel (SC) unidirectional channel

- from one node to many
- sequence of successive, overlapping secure associations

Secure association (SA) within a SC

- every frame transmitted over MACsec belongs to one particular SA
- packet number and key are per-SA

Security Entity (SecY) instance of the MACsec implementation within a node

Uncontrolled port network interface providing insecure service

- MACsec is built on top of this

redhat.

Configuration and relation with IEEE 802.1X

- option 1: admin can configure SC/SA/keys manually
- option 2: use 802.1X with MACsec extensions
  - MKA (MACsec Key Agreement protocol)
  - discovery of other MACsec nodes
  - setup of SC/SA
  - key generation and distribution
  - synchronization of packet numbers

redhat.

Encryption and integrity

mandatory integrity+authenticity, optional encryption

- default crypto algorithm: GCM-AES
  - authenticated encryption with additional data
- the entire MACsec packet is always authenticated
- admin can choose whether to use encryption
  - no encryption, integrity/authenticity only: entire MACsec packet as additional data
  - encryption + integrity/authenticity: ethernet + MACsec header as additional data, original payload is encrypted and authenticated

redhat.

Strict validation

Three possible validation modes for incoming packets:

Strict Non-protected, invalid, or impossible to verify (no matching channel configured) frames are dropped

Check These frames are counted as "invalid" and accepted, if possible

Disabled Incoming frames are simply accepted, if possible

- Encrypted frames cannot be accepted without a matching channel and key

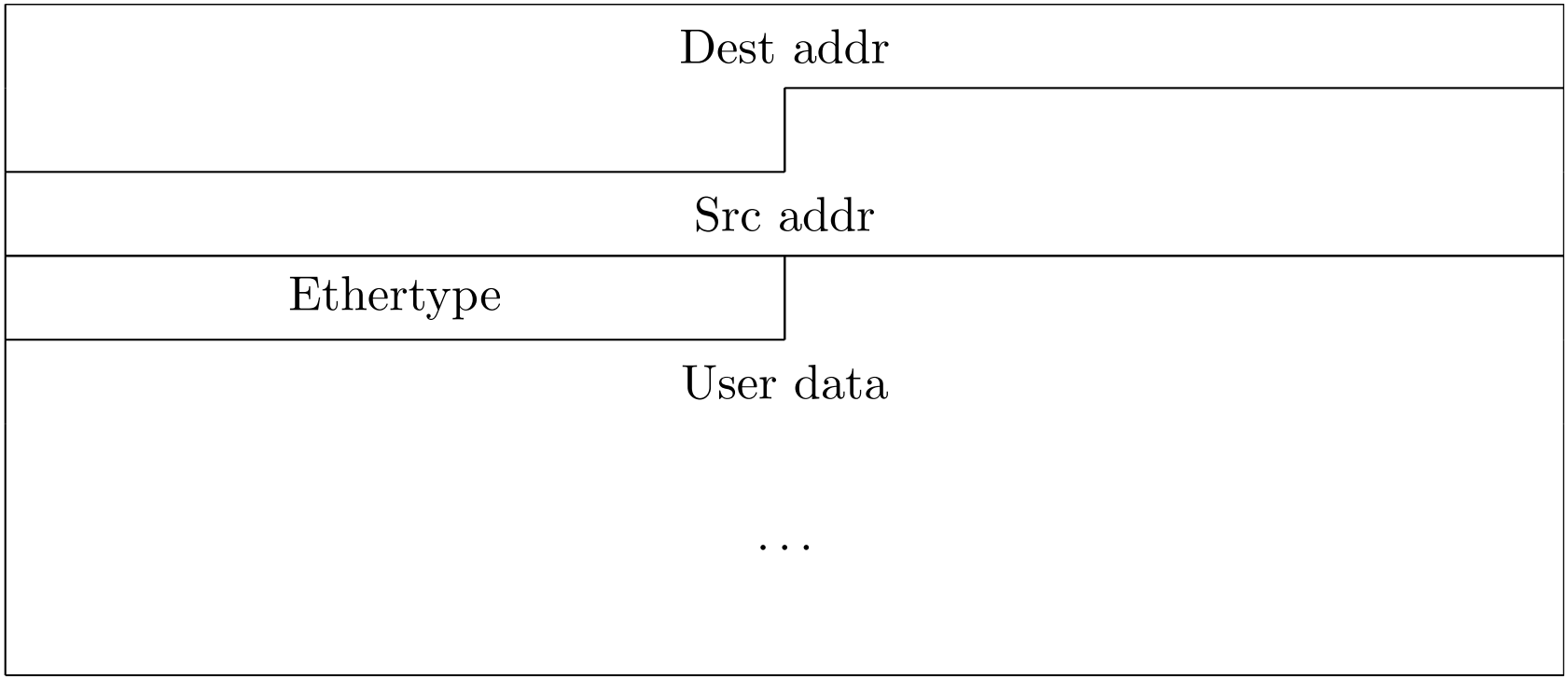
redhat.

Replay protection

- each frame has a 32-bit packet number
- on RX, the node may validate the PN against the lowest PN it expects to get
- configurable replay window
  - some amount of reordering is acceptable

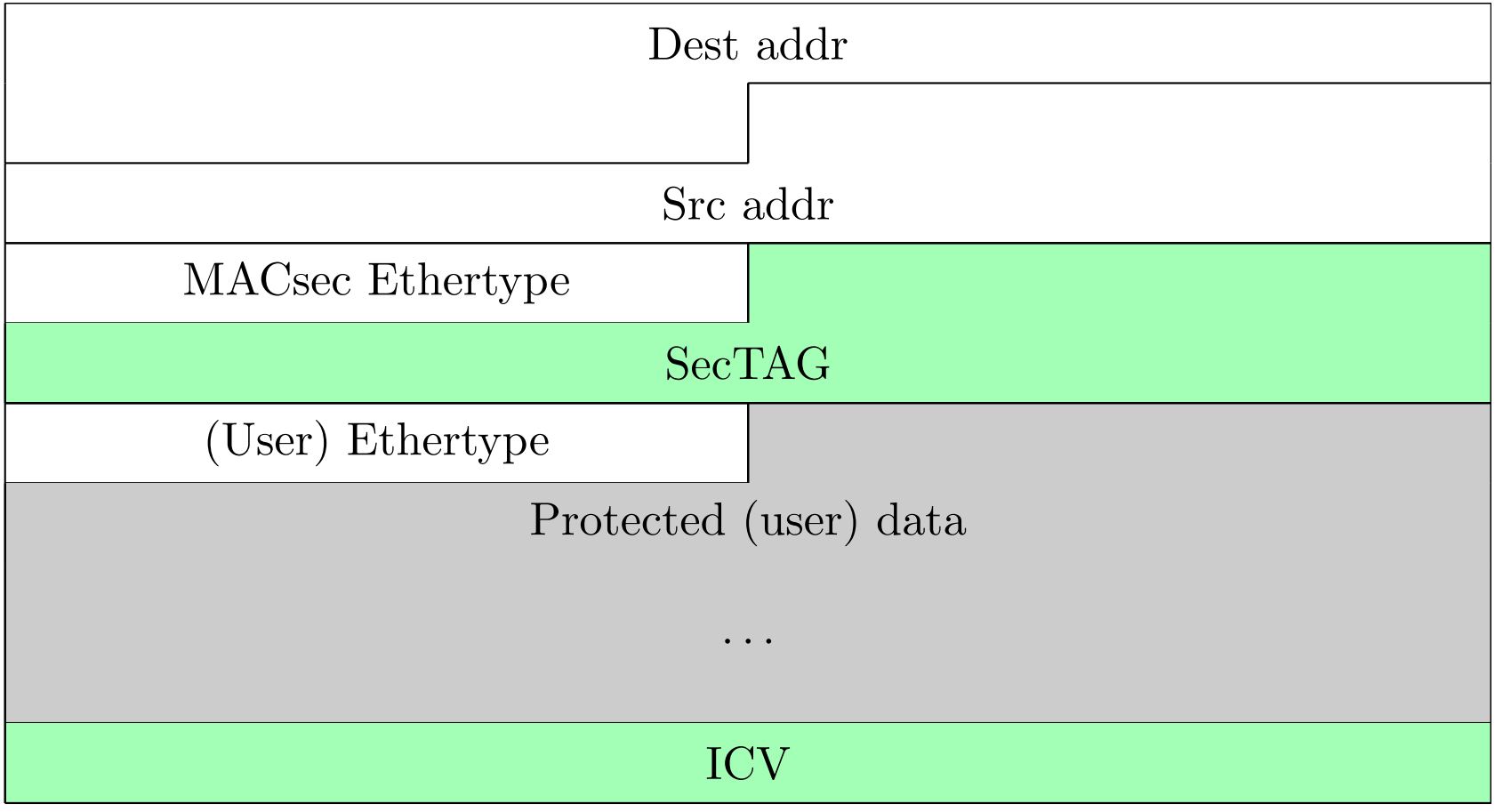
redhat.

Packet format (unprotected frame)



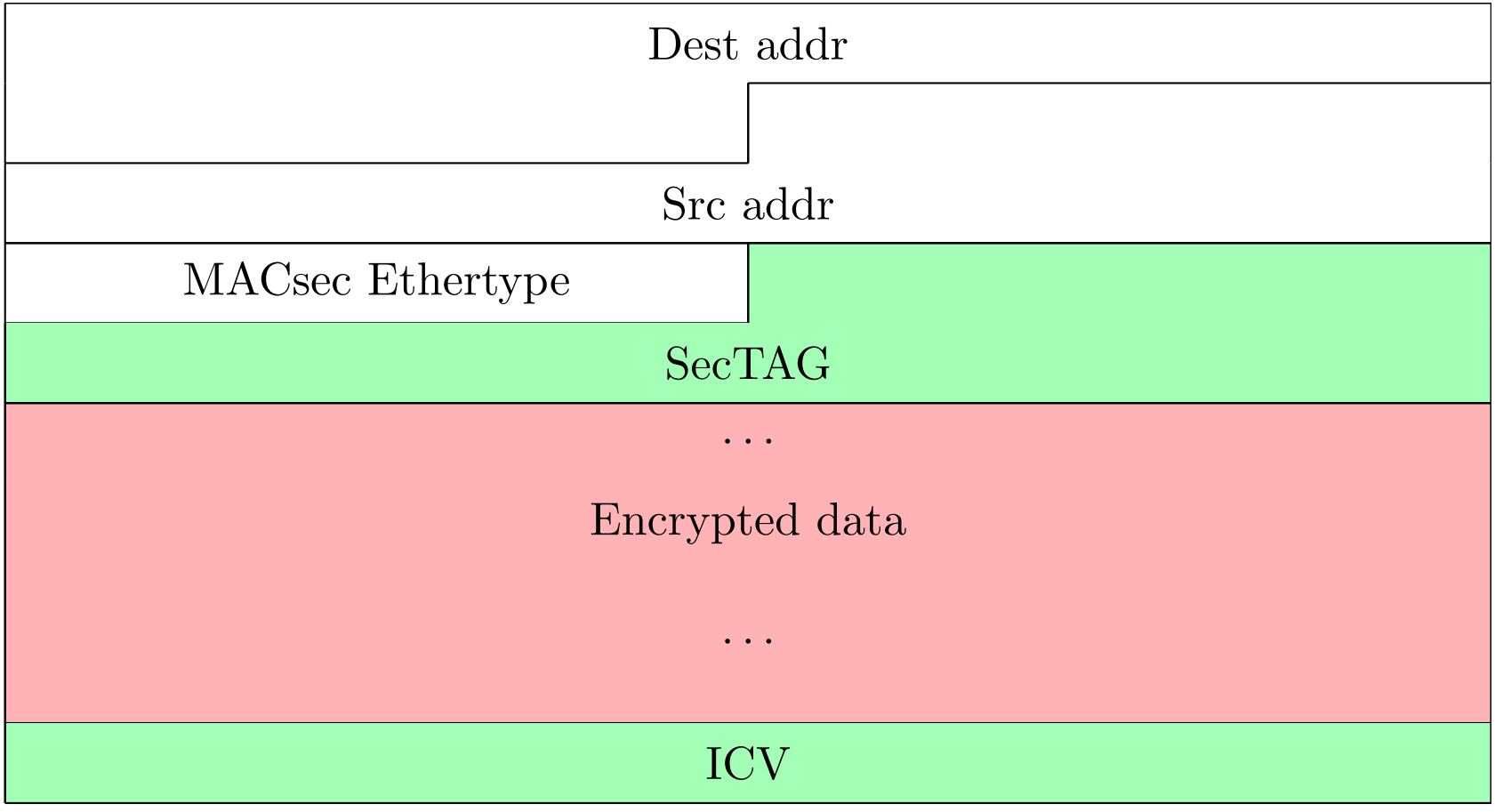
redhat.

Packet format (protected frame)



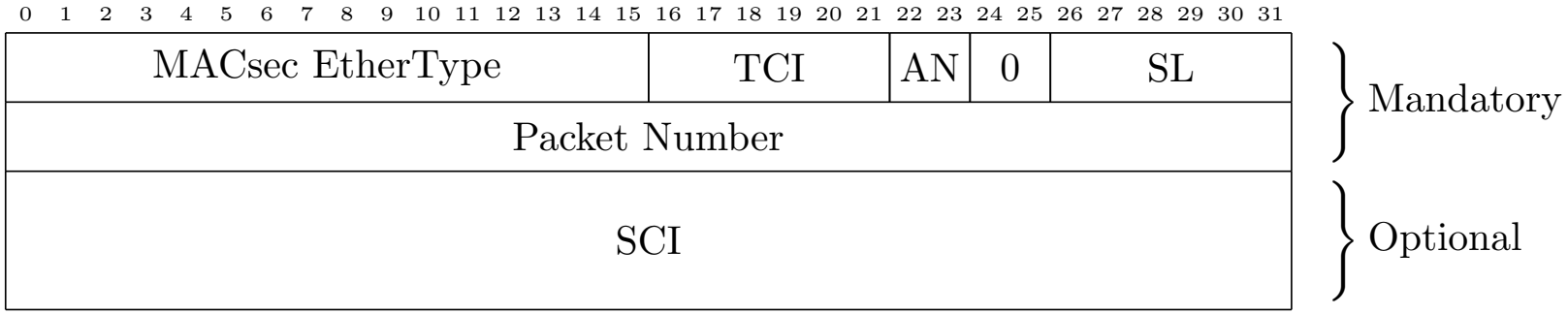
redhat.

Packet format (encrypted frame)



redhat.

SecTAG format



TCI tag control information

AN association number (SA identifier, 2 bits)

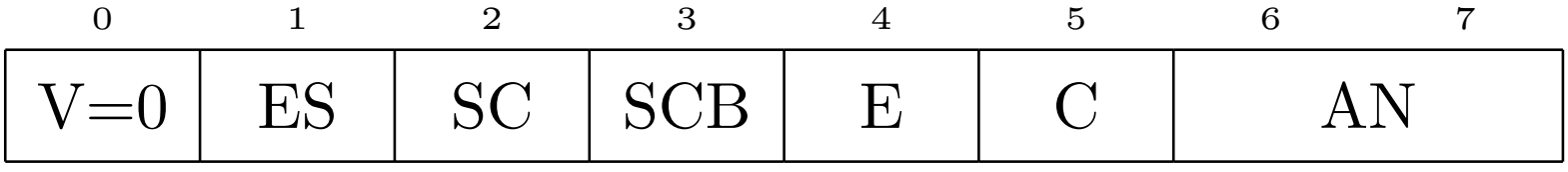
SL short length, non-zero for frame lengths under 64B

SCI secure channel identifier, 64 bits

- 48 bits “system identifier” (MAC address)
- 16 bits “port number”

redhat.

SecTAG format: TCI field



SC SCI present

E Encrypted payload

C Changed text

redhat.

Interaction with other protocols and layers

Figure: unprotected VLAN frame

Figure: MACsec-protected VLAN frame

- VLAN tag is part of the encrypted payload

redhat.

Packet handling: Transmit

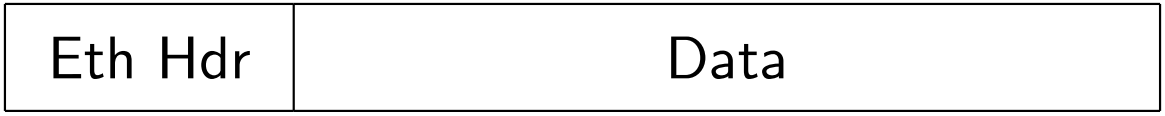


Figure: Packet coming from the stack

1. push SecTAG
2. compute and append ICV
3. pass down to the underlying device

Figure: Packet passed down to the network

redhat.

Packet handling: Receive

Figure: Packet coming from the network

1. verify packet/SecTAG format
2. check packet number (replay protection, optional)
   - just drop the packet, no feedback to a potential attacker
   - helps defend against DoS attacks: don't perform heavy computation on obviously wrong packets
3. decrypt/verify ICV
4. re-check packet number (replay protection after decryption)
5. remove ICV, pop SecTAG

Figure: Packet passed up the stack

redhat.

2 Linux kernel implementation

redhat.

Short description

- create a new netdevice for each TX channel on a specific device
  - similar to VLANs or macvlans
  - "master" device sees only the raw packets
    - ie, the encrypted/protected packets for all its slave MACsec devices
    - and all the non-protected traffic (802.1X, maybe also some normal LAN traffic)
  - good match for the uncontrolled/controlled port model in the IEEE standards
- uses `rx_handler` and `ndo_start_xmit`

redhat.

Crypto

- uses the kernel's crypto API for Authenticated Encryption with Additional Data (AEAD)
- can use HW acceleration (`aesni`) if available

redhat.

Configuration

- API split between rtnetlink and genetlink
- rtnetlink with MACsec-specific options to create the `net_device` and configure SecY attributes
- genetlink to configure TXSA, RXSC, RXSA
  - provides demux between the commands for the 3 kinds of objects
  - cleaner API design than if we had to configure everything over rtnetlink

3 Use cases

redhat.

MACsec LAN setup

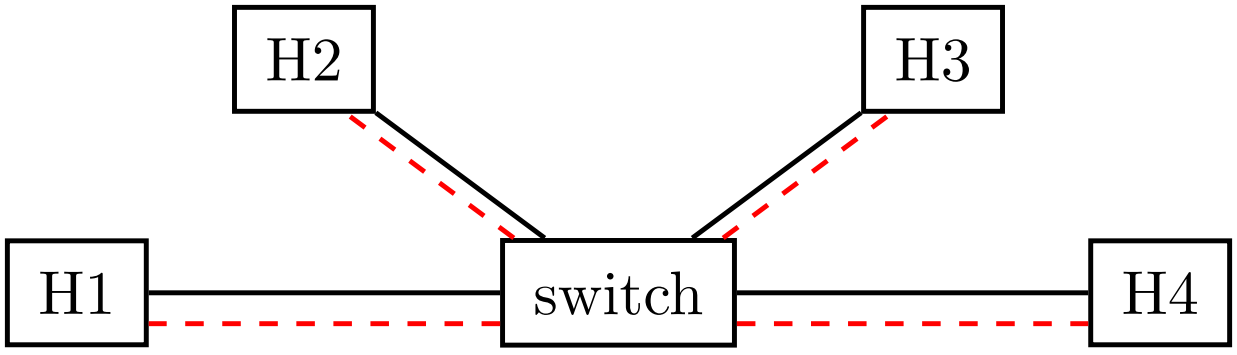


Figure: Example LAN setup

- configure MACsec on the hosts and on each switch port
  - need a switch with MACsec support
- configure MACsec only on the hosts
  - works with any switch
  - switch sees only MACsec-protected traffic

redhat.

MACsec LAN sample configuration

H1

```
ip link add link eth0 macsec0 type macsec
ip macsec add macsec0 tx sa 0 on pn 100 key 0 $KEY_0
ip macsec add macsec0 rx address $H2_ADDR port 1
ip macsec add macsec0 rx address $H2_ADDR port 1 \
    sa 0 pn 100 on key 1 $KEY_1
```

H2

```
ip link add link eth0 macsec0 type macsec
ip macsec add macsec0 tx sa 0 on pn 100 key 1 $KEY_1
ip macsec add macsec0 rx address $H1_ADDR port 1
ip macsec add macsec0 rx address $H1_ADDR port 1 \
    sa 0 pn 100 on key 0 $KEY_0
```

redhat.

Important configuration parameters

Changing the current active TXSA

```
ip link set macsec0 type macsec encoding 2
```

Enabling encryption (optional)

```
ip link add link eth0 macsec0 type macsec ...
# setup SA and RX ...

ip link set macsec0 type macsec encrypt on
```

Enabling replay protection (optional)

```
ip link add link eth0 macsec0 type macsec ...
# setup SA and RX ...

ip link set macsec0 type macsec replay on window 128
```

redhat.

MACsec LAN setup for multiple secure channels

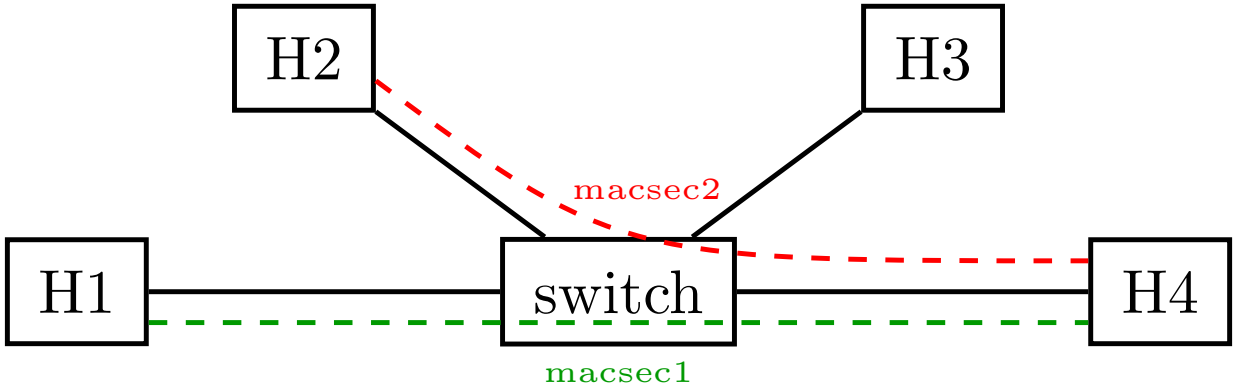


Figure: Example LAN setup with multiple channels

- Nodes H1 and H2 have only one secure channel
  - like in the previous example
- Node H4 has two secure channels
  - different crypto parameters and separate keys for each

redhat.

Multiple channels on an interface

H4

```
# channel to H1
ip link add link eth0 macsec0 type macsec
ip macsec add macsec0 tx sa 0 on pn 100 key 1 $KEY_1
ip macsec add macsec0 rx address $H1_ADDR port 1
ip macsec add macsec0 rx address $H1_ADDR port 1 \
    sa 0 pn 100 on key 0 $KEY_0

# channel to H2
ip link add link eth0 macsec1 type macsec port 2
ip macsec add macsec1 tx sa 0 on pn 400 key 2 $KEY_2
ip macsec add macsec1 rx address $H2_ADDR port 1
ip macsec add macsec1 rx address $H2_ADDR port 1 \
    sa 0 pn 100 on key 3 $KEY_3
```

redhat.

MACsec VLAN setup

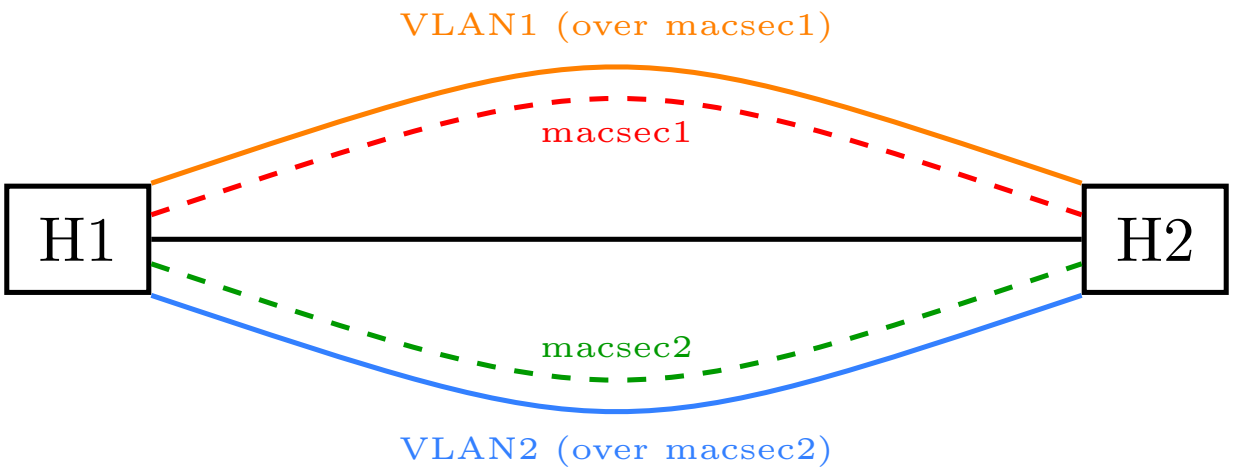


Figure: Example VLAN setup

redhat.

VLAN over MACsec configuration (VLAN1)

H1, VLAN1

```
ip link add link eth0 macsec0 type macsec
ip macsec add macsec0 tx sa 0 on pn 100 key 0 $KEY_0
ip macsec add macsec0 rx address $H2_ADDR port 1
ip macsec add macsec0 rx address $H2_ADDR port 1 \
    sa 0 pn 100 on key 1 $KEY_1

ip link add link macsec0 vlan0 type vlan id 42
```

H2, VLAN1

```
ip link add link eth0 macsec0 type macsec
ip macsec add macsec0 tx sa 0 on pn 100 key 1 $KEY_1
ip macsec add macsec0 rx address $H1_ADDR port 1
ip macsec add macsec0 rx address $H1_ADDR port 1 \
    sa 0 pn 100 on key 0 $KEY_0

ip link add link macsec0 vlan0 type vlan id 42
```

redhat.

VLAN over MACsec configuration (VLAN2)

H1, VLAN2

```
ip link add link eth0 macsec1 type macsec port 2
ip macsec add macsec1 tx sa 0 on pn 100 key 2 $KEY_2
ip macsec add macsec1 rx address $H2_ADDR port 2
ip macsec add macsec1 rx address $H2_ADDR port 2 \
    sa 0 pn 100 on key 3 $KEY_3

ip link add link macsec1 vlan0 type vlan id 10
```

H2, VLAN2

```
ip link add link eth0 macsec1 type macsec port 2
ip macsec add macsec1 tx sa 0 on pn 100 key 3 $KEY_3
ip macsec add macsec1 rx address $H1_ADDR port 2
ip macsec add macsec1 rx address $H1_ADDR port 2 \
    sa 0 pn 100 on key 2 $KEY_2

ip link add link macsec1 vlan0 type vlan id 10
```

redhat.

MACsec Bonding setup

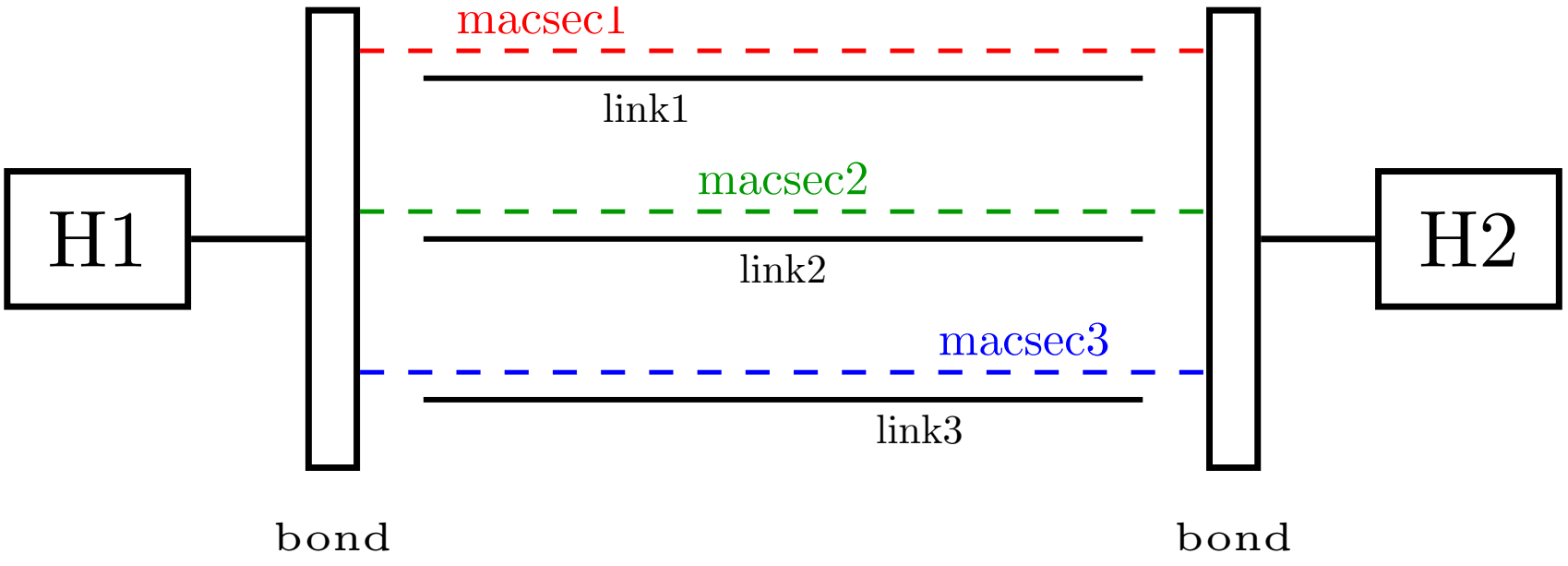


Figure: Example Bonding setup

- MACsec is configured separately on each underlying link
- MACsec netdevices are enslaved instead of the real links
- LACP/etc traffic is protected by MACsec

redhat.

MACsec bond configuration

Create bond

```
# modprobe bonding max_bonds=0
ip link add bond0 type bond [...]
ip link set bond0 up
```

Set up MACsec on each bonded link

```
ip link add link eth0 macsec0 type macsec ...
# setup SA and RX on macsec0 like before
ip link add link eth1 macsec1 type macsec ...
# setup SA and RX on macsec1 like before
```

Add the MACsec devices to the bond

```
ip link set macsec0 master bond0
ip link set macsec1 master bond0
```

redhat.

MACsec VXLAN setup

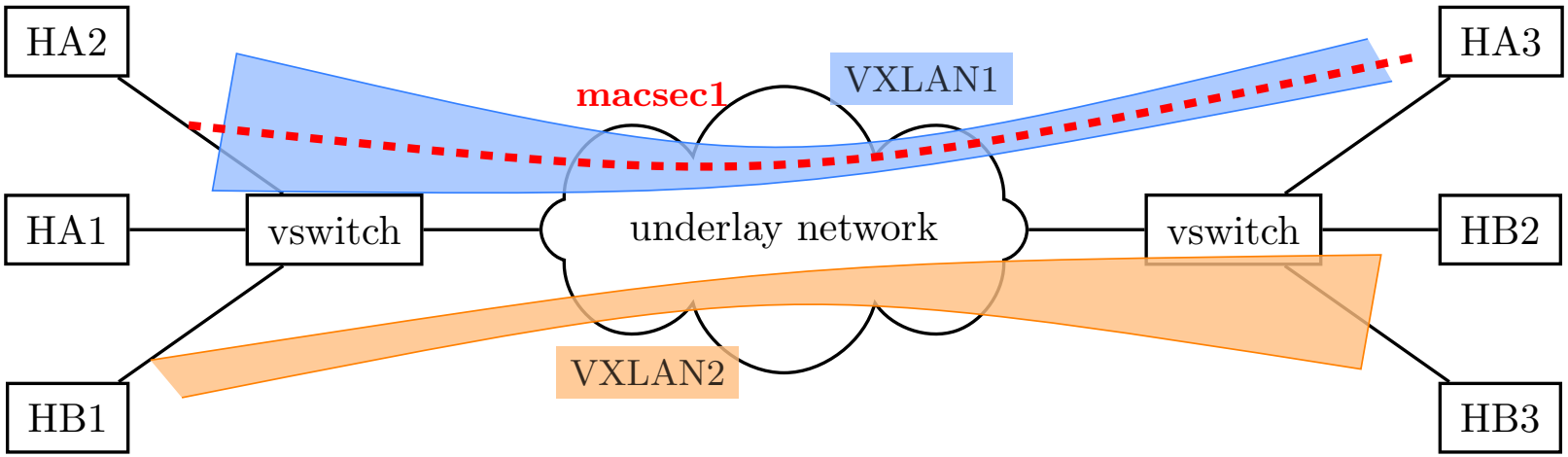


Figure: Example VXLAN setup

ETH	IP	UDP	VXLAN	ETH	SecTAG	Payload ...	ICV

Figure: Encapsulation for a MACsec over VXLAN packet

redhat.

MACsec VXLAN configuration

VXLAN

```
ip link add link vxlan0 type vxlan \
     id 10 group 239.0.0.10 ttl 5 dev eth0

ip link add link vxlan0 macsec0 type macsec ...
# setup SA and RX on macsec0 like before
```

4 Conclusion

redhat.

In the kernel

- optional features

  confidentiality offset the first 30 bytes of the packet are only integrity protected

  additional ciphersuite GCM-AES-256
- hardware offload (at least for some Intel ixgbe NICs)
- performance improvements

redhat.

In userspace

- NetworkManager support
- `wpa_supplicant` already has MKA support, need to hook up the netlink API
  - MKA support: commits `7baec808efb5`, `887d9d01abc7`, `dd10abccc86d`

redhat.

Questions?

- Feedback: `http://www.devconf.cz/feedback/374`

redhat.

More information

- IEEE 802.1AE-2006
  `http://standards.ieee.org/getieee802/download/802.1AE-2006.pdf`
- IEEE 802.1X-2010
  `http://standards.ieee.org/getieee802/download/802.1X-2010.pdf`
- Kernel submission (RFCv2 on netdev)
  `http://www.spinics.net/lists/netdev/msg362389.html`