Tunneling, IPSec, VRF

Matěj Grégr, mgregr@fit.vut.cz

Tunneling

What Is Tunneling?

- Many times it's useful to create "illusion" of the new network above the existing one. Here are some motivations:
  - *Existing network doesn't recognize protocol which we would need to transfer across it or service we would like to use*
  - *We would like to use existing network as transport tool but we want it to be completely invisible from point of view of internal network*
  - *We need to interconnect multiple sites with potentially private IP address space*
  - *We don't trust existing network and we want to securely transfer data across it*
- **Tunneling** = technique where packet is reencapsulated into the new packet
  - Former packet becomes payload of new packet and therefore its content (L3 header) is not in attention of routers

Protocols Used in Tunneling

- **Passenger protocol**
  - *We would like to transfer datagrams of this protocol inside tunnel*
  - E.g. IPX, AppleTalk, IPv4, IPv6
- **Encapsulating/Tunneling protocol**
  - Header of this protocol is prepended before passenger protocol
  - It's used to identify passenger protocol and secure transmission with authentication, encryption, etc.
  - E.g. GRE, IPsec, L2F, PPTP, L2TP
- **Carrier protocol**
  - Existing network uses this protocol for transport and inside it encapsulating protocol is carried wrapped around passenger protocol
  - E.g. Frame-relay, ATM, Ethernet

Encapsulating Protocols

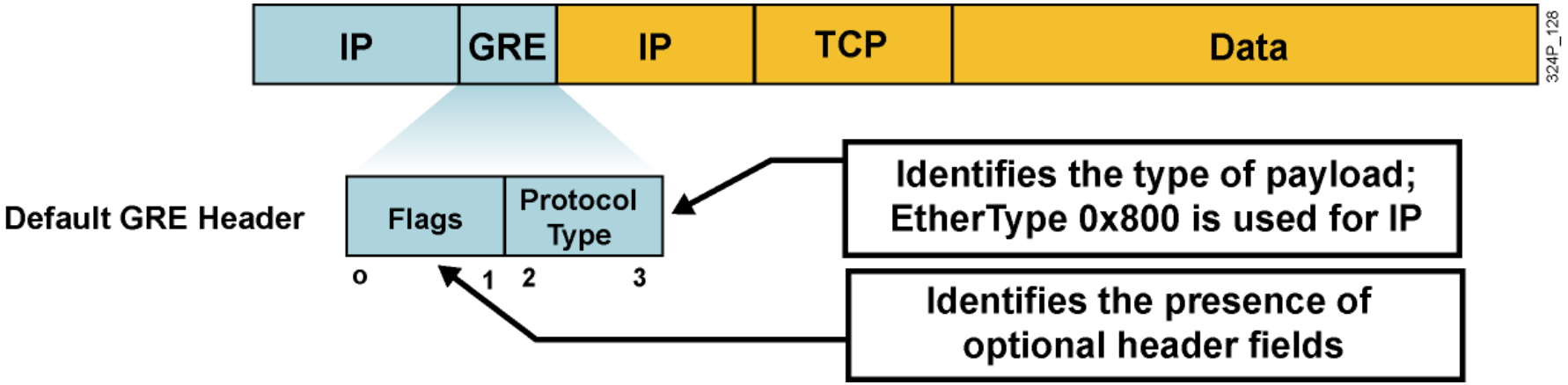
- Tunneling could be achieved with or even without support of encapsulating protocol
- **Tunneling WITH encapsulating protocol**
  - Encapsulating protocol wraps around original data and then is inserted into new packet in carrier protocol
  - Authentication support, multiple tunnels between same devices, encryption
  - More features means potentially more overhead
  - E.g. GRE, L2TP, PPTP
- **Tunneling WITHOUT encapsulating protocol**
  - Original packet is directly inserted into the new one
  - Limited support of advanced tunneling features
  - Minimal overhead
  - E.g. IP-in-IP, IPv6-in-IPv4

Generic Routing Encapsulation (GRE)

- **GRE** is encapsulating/tunneling protocol on L3
  - Supports multiple passenger protocols
  - Creates virtual point-to-point connection between pair of routers
  - Uses IP as carrier protocol
  - Allows transmission of multicast traffic (NBMA nature)
- GRE was originally invented by Cisco, but nowadays it's open standard specified in RFC 2784

GRE Header ①

- GRE is stateless without any signalization and traffic flow control
- GRE doesn't provide any security (no authentication, no encryption, no message integrity, no trustworthiness)
- Overhead of GRE tunnel is 24B (20B for new IPv4 header and 4B for GRE header)



GRE Header ②

- GRE **Flags** are stored in first 2B of header:
  - **Checksum Present (bit 0)**
  - **Key Present (bit 2)**
  - **Sequence Number Present (bit 3)**
  - **Version Number (bits 13–15)**: GRE has version 0, PPTP has version 1
- **Protocol Type** specify type of passenger protocol, usually it has same value as in field EtherType L2 Ethernet frame

Configuring GRE Tunnel

- GRE tunnels are represented by virtual **Tunnel interface** on router
- Tunnel interface must have:
  - Own IP address (just like any other interface)
  - IP address of sender and receiver of (carrier protocol) packets
  - Set proper tunneling mode
- Pair of Tunnel endpoint interfaces on opposite routers must met this criteria:
  - Tunnel endpoints own IP addresses must be in same network segment – *just like any other two directly interconnected interfaces*
  - IP addresses of sender and receiver must correspond on both endpoints – *IP address of receiver on one side must be IP address of sender on the opposite site and vice versa*
- Tunnel interface bandwidth is by default 9 Kbps
  - *Surprisingly it's recommended to change to reflect real situation* ☺

GRE Configuration


```
hostname Brno
!
interface Serial0/0
 ip address 192.3.4.5 255.255.255.0
 no shutdown
!
interface Tunnel0
 bandwidth 1000
 tunnel source s0/0
 tunnel destination 223.1.2.3
 tunnel mode gre ip ! OPTIONAL
 ip address 10.0.0.1 255.255.255.0
!
router ospf 1
 network 10.0.0.1 0.0.0.0 area 0
```

```
hostname Jesenik
!
interface Serial0/0
 ip address 223.1.2.3 255.255.255.0
 no shutdown
!
interface Tunnel7
 bandwidth 1000
 tunnel source s0/0
 tunnel destination 192.3.4.5
 tunnel mode gre ip ! OPTIONAL
 ip address 10.0.0.2 255.255.255.0
!
router ospf 1
 network 10.0.0.2 0.0.0.0 area 0
```

Tunnel Interface Status

- State „up, protocol up“ when using Tunnel interface for GRE is shown when following conditions are met:
  - Interface has defined source and destination addreses with commands **`tunnel source`** and **`tunnel destination`**
  - Actual interface with IP address specified in **`tunnel source`** is in state „up, protocol up“ – *source interface is working*
  - Route to address specified in **`tunnel destination`** must be present in routing table – *destination IP must be reachable according to router's RT*
  - Whenever GRE Keepalive feature is enabled then opposite side of tunnel should be able to response to Keepalive packets – transport network should be able to deliver packets between tunnel endpoints

IPsec for Dummies

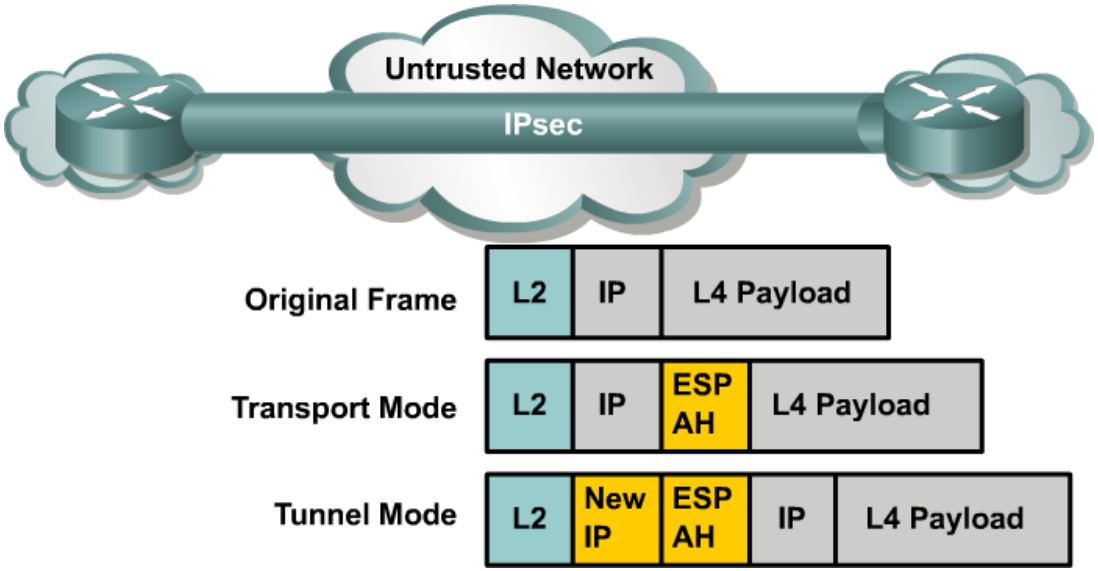
IP Security

- **IPsec** is series of IETF standards describing ways how to secure transmission of IP packet
- IPsec provides:
  - Data **confidentiality** – *Nobody can read it!*
  - Data **integrity** – *Nobody alter data as they traversed through network!*
  - Data **origin authentication** – *We know exactly and certainly who send it!*
  - **Anti-replay** protection
- IPsec uses 3 supporting protocols
  - **Internet Key Exchange (IKE)** for secure transfer of shared keys and NAT-T support (UDP ports 500 a 4500)
  - **Authentication Header (AH)** for sender authentication, data integrity and optional anti-replay protection
  - **Encapsulating Security Payload (ESP)** for data encryption, sender authentication, data integrity and optional anti-replay protection

AH and ESP

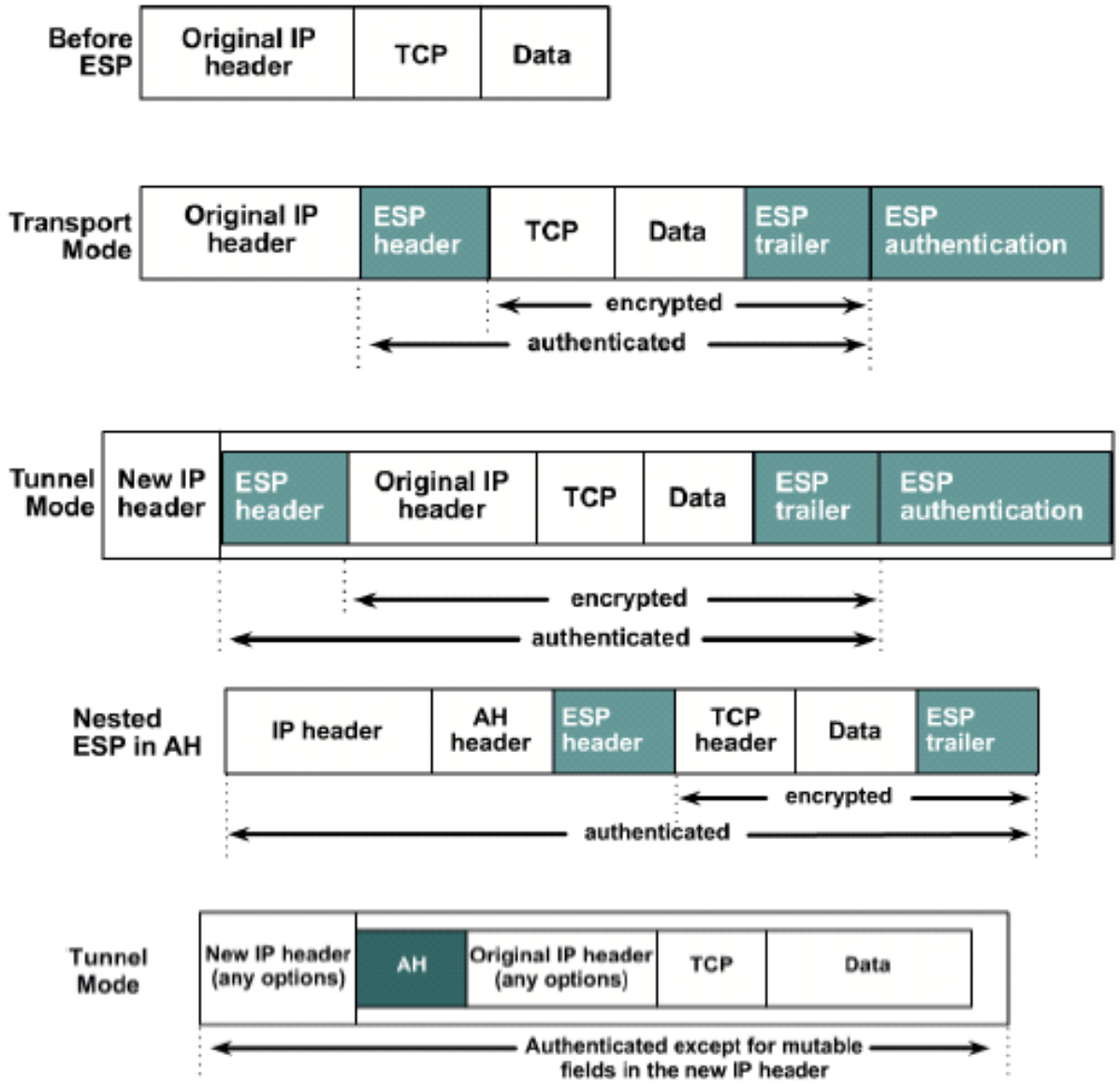
- AH protects packet payload and fixed IP header fields
  - Doesn’t provide encryption
  - Doesn’t like NAT (because NAT rewrites IP headers)
- ESP protects payload with encryption
  - Doesn’t secure IP header
  - Authentication is provided optionally
- AH is nowadays used rarely (even ASA doesn’t support AH), on the other hand ESP is used very often
- AH and ESP could be used simultaneously

IPsec Modes of Operation ①



- **Transport Mode**
  - Original header is used – routing is intact
  - Only the payload of the IP packet is usually encrypted and/or authenticated
  - Transport mode is used on Cisco router only when router itself is sender of packet
- **Tunnel Mode**
  - Entire IP packet is encrypted and/or authenticated
  - Adds new IP header

IPsec Modes of Operation ②



Security Association

- **Security Association (SA)** is a complete list of negotiated parameters between IPsec peers
- SA contains following operating information
  - *How is authentication of peers done?*
  - *In which operational mode should IPsec work?*
  - *Which algorithm and key is used for data encryption?*
  - *Which algorithm is used for data integrity?*
  - *How and how often should be keys replenished?*
- ISAKMP (IKE) is responsible for creation and maintenance SA between peers

IPsec Tunnel Creation

1. Host A sends interesting traffic to Host B.

2. Routers A and B negotiate an IKE Phase 1 session.

3. Routers A and B negotiate an IKE Phase 2 session.

4. Information is exchanged via the IPsec tunnel.

5. The IPsec tunnel is terminated.

IKE Phase 1 ①

- **IKE Phase 1** creates secure channel for IPsec peers authentication
- IKE Phase 1 has three steps
  1. Negotiating ISAKMP policies
  2. Diffie-Hellman key exchange
  3. Peers authentication

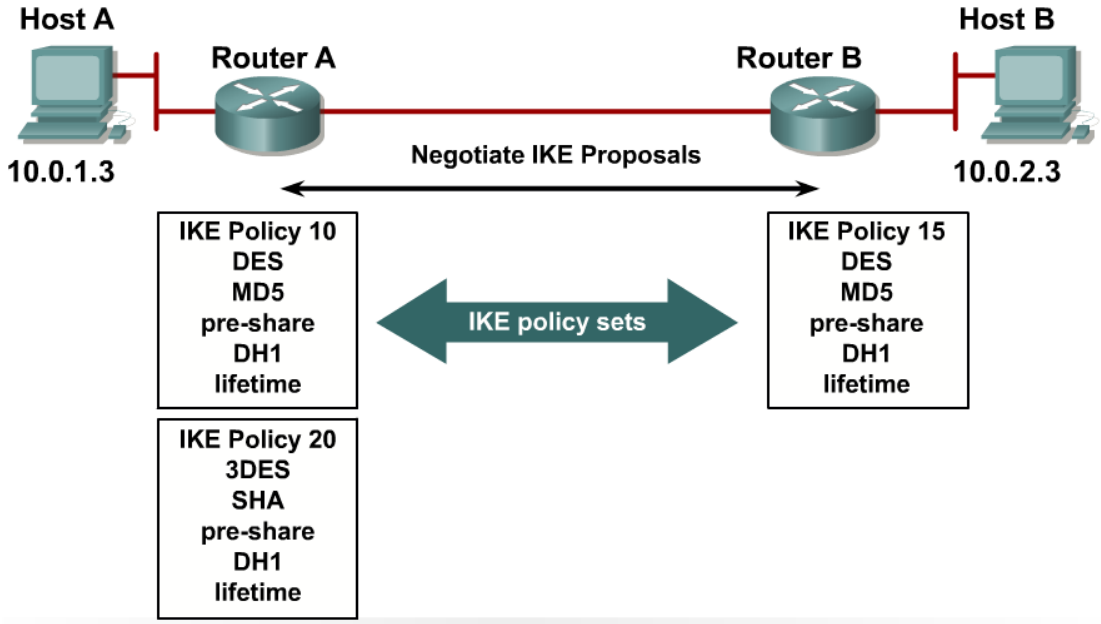
1. Negotiating ISAKMP policies
   - *Which encryption algorithm to use?*
   - *Which hashing algorithm to use?*
   - *Which Diffie-Hellman group?*
   - *How to authenticate peer?*

3. Peers authentication
   - According to way negotiated in Step 1 (pre-shared, RSA nonce, RSA signature)

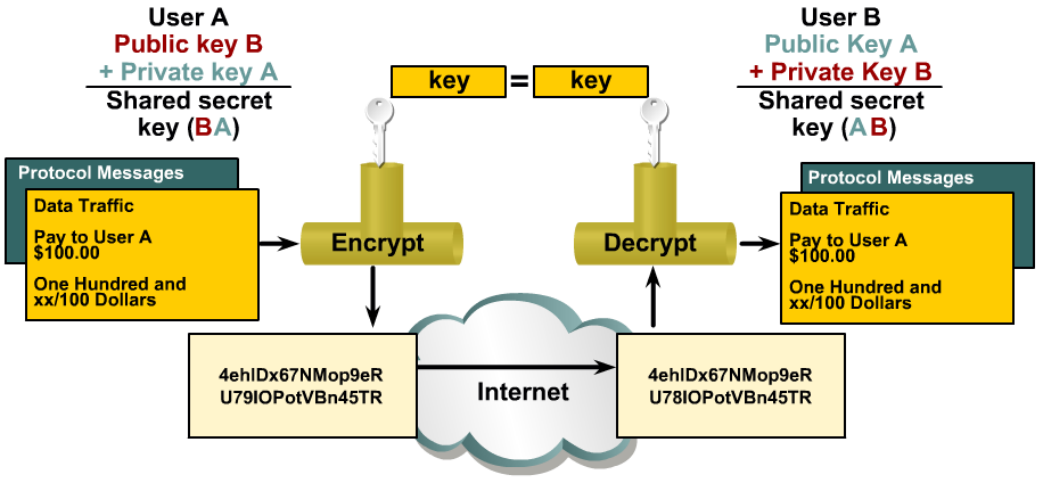
- Properties of IPsec tunnel itself are not negotiated, this is done in IKE Phase 2

IKE Phase 1 ②

1. Step



2. Step



IKE Phase 2

- In **IKE Phase 2** properties of IPsec tunnel between peers are negotiated
  - *Which protocol to use – AH, ESP, AH+ESP?*
  - *Which encryption algorithm to use?*
  - *Which hashing algorithm to use?*
  - *Which operational mode of IPsec to use?*
  - *What are keys used for encryption and decryption?*
  - *What is a lifetime of this negotiated properties?*
- First four properties are called **transform set (TS)**

Configuring of IPsec

- Successful implementation of IPsec consists of:
  - Create at least one ISAKMP policy for IKE Phase 1
  - Create at least one transform set for IKE Phase 2
  - Create crypto map and ACL describing interesting traffic
  - Apply crypto map on egress interface

Configuring ISAKMP Policy

```
crypto isakmp policy 1
 encr 3des
 hash md5
 authentication pre-share
 group 2
 lifetime 3600
 exit
crypto isakmp key 0 HESLO addr 192.0.2.254
```

- Encryption: 3DES
  - Alternatives: DES, AES
- Hash: MD5
  - Alternatives: SHA
- Authentication: pre-shared key
  - Alternatives: RSA based
- DH group: 2 (1024b)
  - Alternatives: 1 (768b), 5 (1536b)
- Lifetime: 3600s
- Pre-shared key for peer with address 192.0.2.254

Configuring Transform Set

- Transform set defines
  - Usage of AH and/or ESP
  - Encryption algorithm and length of key
  - Hashing algorithm
  - Operational mode – either transport or tunnel
- Transform sets are identified by their names
- Configuration snippet:

```
crypto ipsec transform-set AH-ESP-3DES-SHA ah-sha-hmac esp-3des
crypto ipsec transform-set ESP-AES-SHA esp-aes 256 esp-sha-hmac
```

Creation of Crypto Map

- Crypto map bonds together:
  - IPsec peers
  - ACL matching exact traffic to be encrypted between those peers
  - Target transform set to use
  - Way how to exchange keys (either manually or with support of ISAKMP)
  - Lifetime of SA and keys
- Crypto map must minimally consists of…
  - Peer definition
  - ACL reference
  - Transform set reference
- ACL defines which packets should be passed through IPsec tunnel
  - Usually statement in form "*from our network to peer's network*"
  - Stay away from usage statement with `any`!

Configuring Crypto Map

- Example:
  - Block 10: Lifetime is 10 000 KB or 1800 s, two alternative TS, newly generated keys with DH group 5
  - Block 20: Minimal configuration with one TS

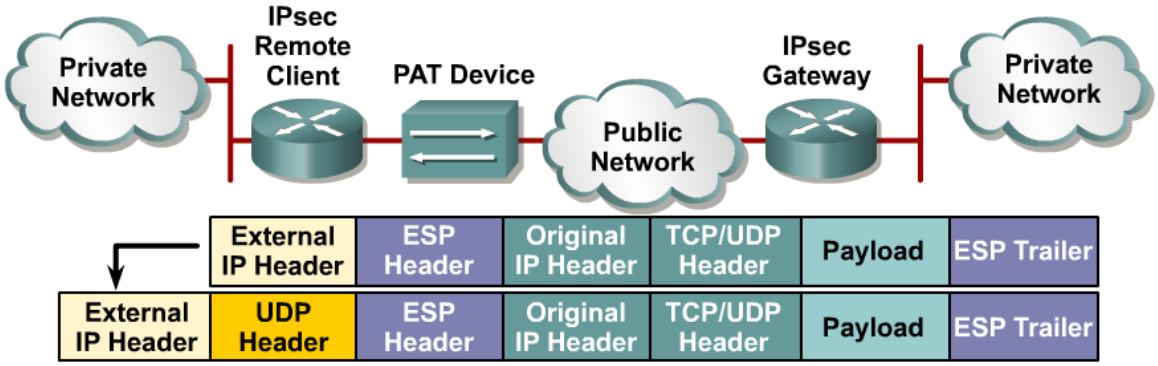
```
crypto map CM-S0/0 10 ipsec-isakmp ! Prvý blok, číslo 10
 set peer 1.2.3.4
 set security-association lifetime kilobytes 10000
 set security-association lifetime seconds 1800
 set transform-set ESP-AES-SHA AH-ESP-3DES-SHA
 set pfs group5
 match address 100

crypto map CM-S0/0 20 ipsec-isakmp ! Druhý blok, číslo 20
 set peer 5.6.7.8
 set transform-set ESP-AES-SHA
 match address 110

interface Serial 0/0
 crypto map CM-S0/0
```

Final IPsec Configuration

```
crypto isakmp policy 1
 encr 3des
 hash md5
 authentication pre-share
 group 2
 lifetime 3600

crypto isakmp key 0 prd31 address 192.0.2.254

crypto ipsec transform-set ESP-AES-SHA esp-aes 256 esp-sha-hmac

crypto map CM-S0/0 10 ipsec-isakmp
 set peer 192.0.2.254
 set transform-set ESP-AES-SHA
 match address 100

crypto map CM-S0/0 local-address Lo0 ! Loopback addresses peering

access-list 100 permit ip 10.0.0.0 0.255.255.255 192.168.10.0 0.0.0.255

int s0/0
 crypto map CM-S0/0
```

Final Notes

- ESP has huge issue with NAT hence **NAT-T (NAT Traversal)** was specified in RFC 3715
- NAT-T must be allowed through any firewall
  - UDP/500
  - UDP/4500



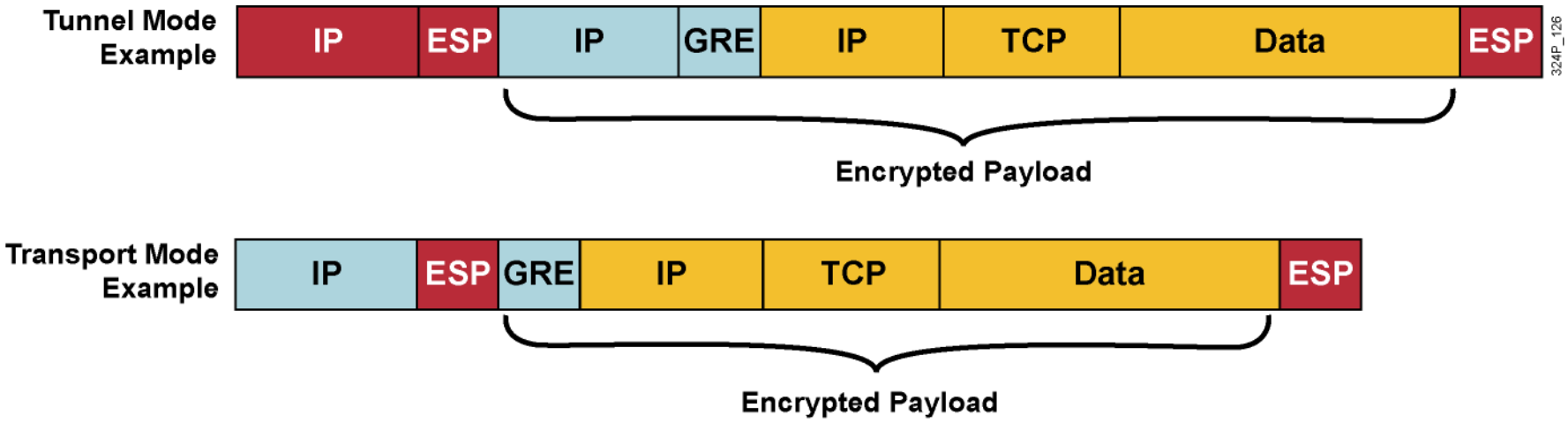
- For mobile clients is recommended new technology SSLVPN instead of robust and technologically complicated IPsec

Secure GRE Tunnels

IPsec + GRE?

- On the one hand IPsec is great for secure transfer of data, BUT…
  - Supports only IP
  - Older IOSes don't support IPsec + multicast
  - Until recent only way how to configure IPsec was with cryptomap on egress interface hence it wasn't possible to…
    - …create Tunnel interface representing IPsec tunnel
    - …activate routing protocol above this tunnel
- On the other hand GRE is great tunneling protocol, unfortunately its security features are terrible
- Solution = secure GRE tunnels with IPsec

IPsec + GRE!



- GRE could use IPsec in either tunnel or transport mode
  - Transport mode is efficient – saves 20 B on additional IP header per every packet
- Encapsulation order:
  - Passenger protocol → GRE → IPsec → IP

Configuration Ways ①

- Two existing ways
  - **Crypto maps** (in every IOS)
  - **IPsec profiles** applied directly on Tunnel interface (only in newer IOSes)
- Crypto map way for IPsec + GRE is similar to IPsec crypto map only...
  - ...however it's necessary to be aware of fact that egress interface transmits GRE packets instead of "plain IP packets"
  - Command `set peer` in crypto map must match with IP address in `tunnel destination` command on Tunnel interface
  - ACL in crypto map must match GRE packet type with source and destination address referred in `tunnel source` resp. `tunnel destination` commands
  - 12.2(13)T and older IOSes must have crypto map applied both on Tunnel interface and also egress interface (Document ID 14125)
  - All other configuration steps are similar

Crypto Map Configuration Example


```
hostname Jesenik
!
crypto map KRYPTUJ 1 ipsec-isakmp
 match address 150
 set transform-set TS
 set peer 223.1.2.3
!
interface Serial0/0
 ip address 192.3.4.5 255.255.255.0
 crypto map KRYPTUJ
!
interface Tunnel0
 tunnel source s0/0
 tunnel destination 223.1.2.3
 ip address 10.0.0.1 255.255.255.0
 crypto map KRYPTUJ ! Unnecessary on 12.2(13)T and newer
!
access-list 150 permit gre host 192.3.4.5 host 223.1.2.3
```

Configuration Ways ②

- Newer IOSese support more convenient IPsec profile way with use of:
  - IPsec profiles
  - Tunnel interface command **`tunnel protection`**
- IPsec profile is simplified version of crypto map
  - without **`match address`** for ACL
  - without **`set peer`**
- On Tunnel interface exists command **`tunnel protection`** that bonds tunnel with IPsec profile
- With this configuration way there's no need to create crypto map and ACL
  - All other configuration steps are still necessary – defining ISAKMP policies, pre-shared password, transform-sets
  - Be aware – GRE Keepalives feature isn't supported when using IPsec profiles (Document ID 64565)

IPsec Profiles Configuration Example


```
hostname Jesenik
!
crypto ipsec profile KRYPTUJ
 set transform-set TS
!
interface Serial0/0
 ip address 192.3.4.5 255.255.255.0
!
interface Tunnel0
 tunnel source s0/0
 tunnel destination 223.1.2.3
 tunnel protection ipsec profile KRYPTUJ
 ip address 10.0.0.1 255.255.255.0
```

Useful Commands

```
show crypto ipsec

show crypto session


clear crypto session


debug crypto isakmp

debug crypto ipsec
```

VRF

- VRF is a technology for creating separate virtual routers on a single physical router.
- VRF-Lite provides VRF without MPLS. Router interfaces, routing tables, and forwarding tables are isolated on an instance-by-instance basis and therefore prevent traffic from one VRF instance from interfering with another VRF instance.
- VRF is an essential component of the MPLS L3VPN architecture and provides increased router functionality through segmentation in lieu of using multiple devices.
- This section introduces you to VRF and demonstrates how you can configure and verify VRF-Lite in a Cisco network.

VRF-Lite Overview

- By default, all router interfaces, the routing table, and any forwarding tables are associated with the global VRF instance.
- What you've been calling your routing table is actually the routing table of the global VRF instance.
- If you need to divide your router up into multiple virtual routers, you can do so by creating additional VRF instances, which also creates additional routing and forwarding tables.

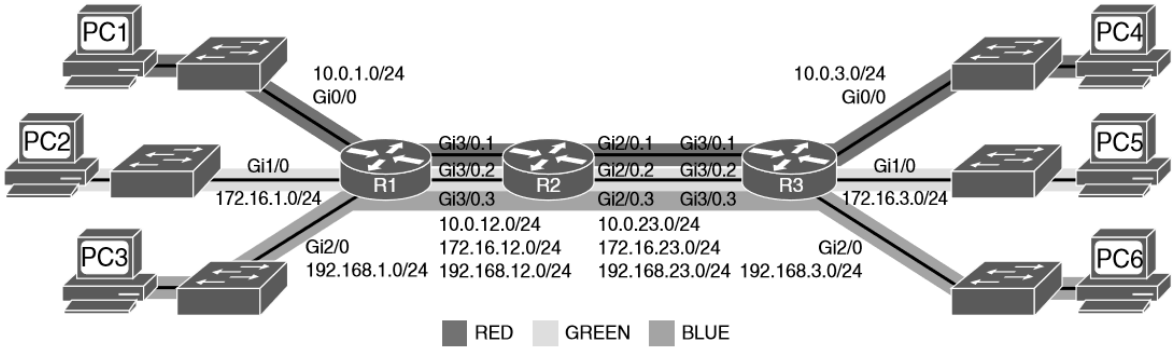


Figure 18-1 *An Example of Three VRF Instances (from top to bottom: red, green, and blue)*

Consider this scenario: for security reasons, you need to build three different networks so that traffic in each network is isolated from traffic in the other networks. However, you only want to build a single physical network to accomplish this. You can do this by using VRF. Figure 18-1 shows a single physical topology that is divided into three different logically isolated networks.

Implementing and Verifying VRF-Lite

Creating and Verifying VRF Instances

Example 18-1 shows how the **ip vrf** *vrf-name* command is used on each of the routers to create the VRF instances.

To verify that the VRF instances are created, use the **show ip vrf** command as shown in Example 18-2 for R1. Notice that the Interfaces column is empty. You need to assign interfaces to each of the VRF instances to separate and isolate the traffic.

Example 18-1 *Configuring VRF Instances on R1 with the* ***ip vrf*** *Command*

```
R1# configure terminal
R1(config)# ip vrf RED
R1(config-vrf)# exit
R1(config)# ip vrf GREEN
R1(config-vrf)# exit
R1(config)# ip vrf BLUE
```

Example 18-2 *Verifying That the VRF Instances Are Configured on R1*

```
R1# show ip vrf
  Name                              Default RD          Interfaces
  BLUE                              <not set>
  GREEN                             <not set>
  RED                               <not set>
```

Creating and Verifying VRF Instances (Cont.)

To assign an interface to a VRF, use the **ip vrf forwarding** *vrf-name* command in interface configuration mode, as shown in in Example 18-3.

Using the **show ip vrf** command again, you can verify that each interface has been assigned to the correct VRF instance, as shown in Example 18-4.

Example 18-3 *Assigning Interfaces to the VRF Instances with the* ***ip vrf forwarding*** *Command*

```
R1# configure terminal
R1(config)# interface gigabitEthernet 0/0
R1(config-if)# ip vrf forwarding RED
R1(config-if)# interface gigabitEthernet 1/0
R1(config-if)# ip vrf forwarding GREEN
R1(config-if)# interface gigabitEthernet 2/0
R1(config-if)# ip vrf forwarding BLUE
R1(config-if)# end
```

Example 18-4 *Verifying That the Interfaces Are Assigned to the Correct VRF Instances*

```
R1# show ip vrf
  Name                              Default RD          Interfaces
  BLUE                              <not set>           Gi2/0
  GREEN                             <not set>           Gi1/0
  RED                               <not set>           Gi0/0
```

Creating and Verifying VRF Instances (Cont.)

If a single physical interface supports multiple VRF instances, the physical interface needs to be broken into subinterfaces. Therefore, Gi3/0 needs to be broken into subinterfaces.

Example 18-5 shows how to create the subinterfaces and assign them to the correct VRF instances. It also shows the use of the **show ip vrf** command to verify that the interfaces are in the correct VRF instances.

Example 18-5 *Creating Subinterfaces on R1 and Assigning Them to the Correct VRF Instances*

```
R1# configure terminal
R1(config)# interface gigabitEthernet 3/0.1
R1(config-subif)# ip vrf forwarding RED
R1(config-vrf)# interface gigabitEthernet 3/0.2
R1(config-subif)# ip vrf forwarding GREEN
R1(config-vrf)# interface gigabitEthernet 3/0.3
R1(config-subif)# ip vrf forwarding BLUE
R1(config-vrf)# end
R1# show ip vrf
  Name                              Default RD          Interfaces
  BLUE                              <not set>           Gi2/0
                                                        Gi3/0.3
  GREEN                             <not set>           Gi1/0
                                                        Gi3/0.2
  RED                               <not set>           Gi0/0
                                                        Gi3/0.1
```

Creating and Verifying VRF Instances (Cont.)

Next, you can configure network addressing on R1. Example 18-6 shows the IP addressing configuration of each of the interfaces on R1, based on Figure 18-1.

Notice that the subinterfaces must be configured with dot1q encapsulation, or you can't assign an IP address to the interface. Also, note that when you configure R2's subinterfaces connecting to R1, they need to be configured with the same VLAN numbers.

Example 18-6 *Configuring R1's Interfaces and Subinterfaces with IP Addresses*

```
R1# configure terminal
R1(config)# int gig 0/0
R1(config-if)# ip address 10.0.1.1 255.255.255.0
R1(config-if)# int gig 1/0
R1(config-if)# ip address 172.16.1.1 255.255.255.0
R1(config-if)# int gig 2/0
R1(config-if)# ip address 192.168.1.1 255.255.255.0
R1(config-if)# int gig 3/0.1
R1(config-subif)# encapsulation dot1Q 100
R1(config-subif)# ip address 10.0.12.1 255.255.255.0
R1(config-subif)# int gig 3/0.2
R1(config-subif)# encapsulation dot1Q 200
R1(config-subif)# ip address 172.16.12.1 255.255.255.0
R1(config-subif)# int gig 3/0.3
R1(config-subif)# encapsulation dot1Q 300
R1(config-subif)# ip address 192.168.12.1 255.255.255.0
```

You can use the **show ip vrf interfaces** command to verify the IP address assigned to the interface, the VRF instance the interface is in, and whether the interface is up or down.

Creating and Verifying VRF Instances (Cont.)

As soon as you created the VRF instance with the **ip vrf** *vrf-name* command, the virtual routing table was created for the network. You can use the **show ip route** command to display the global routing table, as shown in Example 18-8.

To view the routing table for a VRF instance, use the show **ip route vrf** *vrf-name* command.

Example 18-8 *Verifying the Global Routing Table*

```
R1# show ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       + - replicated route, % - next hop override

Gateway of last resort is not set
R1#
```

Implementing and Verifying VRF-Lite

Creating and Verifying VRF Instances (Cont.)

Example 18-10 *Configuring R2 VRF Instances, Assigning Subinterfaces to VRF Instances, and Configuring IP Addresses on Subinterfaces*

```
R2# config terminal
R2(config)# ip vrf RED
R2(config-vrf)# ip vrf GREEN
R2(config-vrf)# ip vrf BLUE
R2(config-vrf)# int gig 3/0.1
R2(config-subif)# ip vrf forwarding RED
R2(config-subif)# encapsulation dot1Q 100
R2(config-subif)# ip address 10.0.12.2 255.255.255.0
R2(config-subif)# int gig 3/0.2
```

You can now configure R2. Example 18-10 shows the configuration required on R2.

Implementing and Verifying VRF-Lite

Creating and Verifying VRF Instances (Cont.)

Example 18-11 *Verifying R2's Configuration with the* ***show ip vrf interfaces*** *Command and the* ***show ip route vrf*** *Command*

```
R2# show ip vrf interfaces
Interface              IP-Address      VRF                              Protocol
Gi3/0.3                192.168.12.2    BLUE                             up
Gi2/0.3                192.168.23.2    BLUE                             up
Gi3/0.2                172.16.12.2     GREEN                            up
Gi2/0.2                172.16.23.2     GREEN                            up
Gi3/0.1                10.0.12.2       RED                              up
Gi2/0.1                10.0.23.2       RED                              up
R2# show ip route vrf RED

Routing Table: RED
...output omitted...
Gateway of last resort is not set

      10.0.0.0/8 is variably subnetted, 4 subnets, 2 masks
C        10.0.12.0/24 is directly connected, GigabitEthernet3/0.1
L        10.0.12.2/32 is directly connected, GigabitEthernet3/0.1
C        10.0.23.0/24 is directly connected, GigabitEthernet2/0.1
L        10.0.23.2/32 is directly connected, GigabitEthernet2/0.1
```

```
R2# show ip route vrf GREEN

Routing Table: GREEN
...output omitted...
Gateway of last resort is not set

      172.16.0.0/16 is variably subnetted, 4 subnets, 2 masks
C        172.16.12.0/24 is directly connected, GigabitEthernet3/0.2
L        172.16.12.2/32 is directly connected, GigabitEthernet3/0.2
C        172.16.23.0/24 is directly connected, GigabitEthernet2/0.2
L        172.16.23.2/32 is directly connected, GigabitEthernet2/0.2
R2# show ip route vrf BLUE

Routing Table: BLUE
...output omitted...
Gateway of last resort is not set

      192.168.12.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.12.0/24 is directly connected, GigabitEthernet3/0.3
L        192.168.12.2/32 is directly connected, GigabitEthernet3/0.3
      192.168.23.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.23.0/24 is directly connected, GigabitEthernet2/0.3
L        192.168.23.2/32 is directly connected, GigabitEthernet2/0.3
R2#
```

Example 18-11 displays the output of the **show ip vrf interfaces** command and the **show ip route vrf** *vrf_name* commands to verify that R2 has been configured correctly.

Creating and Verifying VRF Instances (Cont.)

- To verify connectivity when using VRF instances, you must specify the VRF instance with the **ping** command. If you do not, the global routing table is used instead of the VRF routing table.

Example 18-14 *Verifying Connecting with the ping Command*

```
R1# ping 10.0.12.2
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.12.2, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)
R1# ping vrf GREEN 10.0.12.2
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.12.2, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)
R1# ping vrf RED 10.0.12.2
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.12.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 44/49/60 ms
```

Creating and Verifying VRF Instances (Cont.)

Example 18-15 shows the output of the RED VRF routing table. At this point, you have only directly connected and local routes. For all the routers to learn about all the other networks, you can use static or dynamic routing. The following examples use EIGRP as the dynamic routing protocol to provide full connectivity for each of the VRF instances.

Example 18-15 *Using the show ip route vrf RED Command to Verify the Contents of the RED VRF Routing Table*

```
R1# show ip route vrf RED

Routing Table: RED
...output omitted...
Gateway of last resort is not set

      10.0.0.0/8 is variably subnetted, 4 subnets, 2 masks
C        10.0.1.0/24 is directly connected, GigabitEthernet0/0
L        10.0.1.1/32 is directly connected, GigabitEthernet0/0
C        10.0.12.0/24 is directly connected, GigabitEthernet3/0.1
L        10.0.12.1/32 is directly connected, GigabitEthernet3/0.1
```

Implementing and Verifying VRF-Lite

Creating and Verifying VRF Instances (Cont.)

To configure EIGRP for multiple VRF instances, you use EIGRP named configuration mode because it permits you to create multiple address families, as shown in Example 18-16.

- Enter the EIGRP named configuration mode by using the **router eigrp** *name* command in global configuration mode.
- Next, create an address family for each of the VRF instances. You accomplish this with the **address-family ipv4 vrf** *vrf-name* **autonomous-system** *as-number* command.
- Then specify any EIGRP configuration commands that are needed for your scenario. In this case, you are enabling the routing process on only certain interfaces.

Example 18-16 *Configuring EIGRP for Multiple VRF Instances*

```
R1# configure terminal
R1(config)# router eigrp VRFEXAMPLE
R1(config-router)# address-family ipv4 vrf RED autonomous-system 10
R1(config-router-af)# network 10.0.1.1 0.0.0.0
R1(config-router-af)# network 10.0.12.1 0.0.0.0
R1(config-router)# address-family ipv4 vrf GREEN autonomous-system 172
R1(config-router-af)# network 172.16.1.1 0.0.0.0
R1(config-router-af)# network 172.16.12.1 0.0.0.0
R1(config-router)# address-family ipv4 vrf BLUE autonomous-system 192
R1(config-router-af)# network 192.168.1.1 0.0.0.0
R1(config-router-af)# network 192.168.12.1 0.0.0.0
R1(config-router-af)# end
```

Implementing and Verifying VRF-Lite

Creating and Verifying VRF Instances (Cont.)

To verify that the interfaces are participating in the EIGRP process for the correct VRF instance, use the **show ip eigrp vrf** *vrf-name* **interfaces** command.

Example 18-17 *Verifying That the Interfaces Are Participating in the EIGRP Process for Each VRF*

```
R1# show ip eigrp vrf RED interfaces
EIGRP-IPv4 VR(VRFEXAMPLE) Address-Family Interfaces for AS(10)
           VRF(RED)
                      Xmit Queue   PeerQ        Mean   Pacing Time   Multicast   Pending
Interface     Peers  Un/Reliable  Un/Reliable  SRTT   Un/Reliable   Flow Timer   Routes
Gi0/0           0        0/0          0/0         0        0/0           0           0
Gi3/0.1         0        0/0          0/0         0        0/0           0           0
R1# show ip eigrp vrf GREEN interfaces
EIGRP-IPv4 VR(VRFEXAMPLE) Address-Family Interfaces for AS(172)
           VRF(GREEN)
                     Xmit Queue   PeerQ        Mean   Pacing Time   Multicast   Pending
Interface     Peers  Un/Reliable  Un/Reliable  SRTT   Un/Reliable   Flow Timer   Routes
Gi1/0           0        0/0          0/0         0        0/0           0           0
Gi3/0.2         0        0/0          0/0         0        0/0           0           0
R1# show ip eigrp vrf BLUE interfaces
EIGRP-IPv4 VR(VRFEXAMPLE) Address-Family Interfaces for AS(192)
           VRF(BLUE)
                     Xmit Queue   PeerQ        Mean   Pacing Time   Multicast   Pending
Interface     Peers  Un/Reliable  Un/Reliable  SRTT   Un/Reliable   Flow Timer   Routes
Gi2/0           0        0/0          0/0         0        0/0           0           0
Gi3/0.3         0        0/0          0/0         0        0/0           0           0
```

Creating and Verifying VRF Instances (Cont.)

When all the other routers have been configured for EIGRP, you can verify neighbor adjacencies by using the **show ip eigrp vrf** *vrf-name neighbors* command. As before, because you are dealing with multiple VRF instances, you will notice that each **show** command in Example 18-18 displays only the neighbors that are within that VRF instance.

By using the **ping vrf** *vrf-name ipv4-address* command, as shown in Example 18-20, you can verify that connectivity exists from R1 to R3.

Example 18-18 *Verifying EIGRP Neighbors for Each VRF Instance with the show ip eigrp vrf vrf-name neighbors Command*

```
R1# show ip eigrp vrf RED neighbors
EIGRP-IPv4 VR(VRFEXAMPLE) Address-Family Neighbors for AS(10)
           VRF(RED)
H   Address                 Interface              Hold Uptime   SRTT   RTO  Q  Seq
                                                   (sec)         (ms)       Cnt Num
0   10.0.12.2               Gi3/0.1                  13 00:02:31   48   288  0  7
R1# show ip eigrp vrf GREEN neighbors
EIGRP-IPv4 VR(VRFEXAMPLE) Address-Family Neighbors for AS(172)
           VRF(GREEN)
H   Address                 Interface              Hold Uptime   SRTT   RTO  Q  Seq
                                                   (sec)         (ms)       Cnt Num
```

Example 18-20 *Verifying VRF Connectivity from R1 to R3*

```
R1# ping vrf RED 10.0.3.3
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.3.3, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 68/91/112 ms
R1# ping vrf GREEN 172.16.3.3
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.3.3, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 64/71/80 ms
R1# ping vrf BLUE 192.168.3.3
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.3.3, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 64/68/72 ms
R1#
```

Slides adapted by Vladimír Veselý and Matěj Grégr
partially from official course materials
but the most of the credit goes to CCIE#23527 Ing. Peter Palúch, Ph.D.

Last update: 2023-06-12