Lecture 5.1:

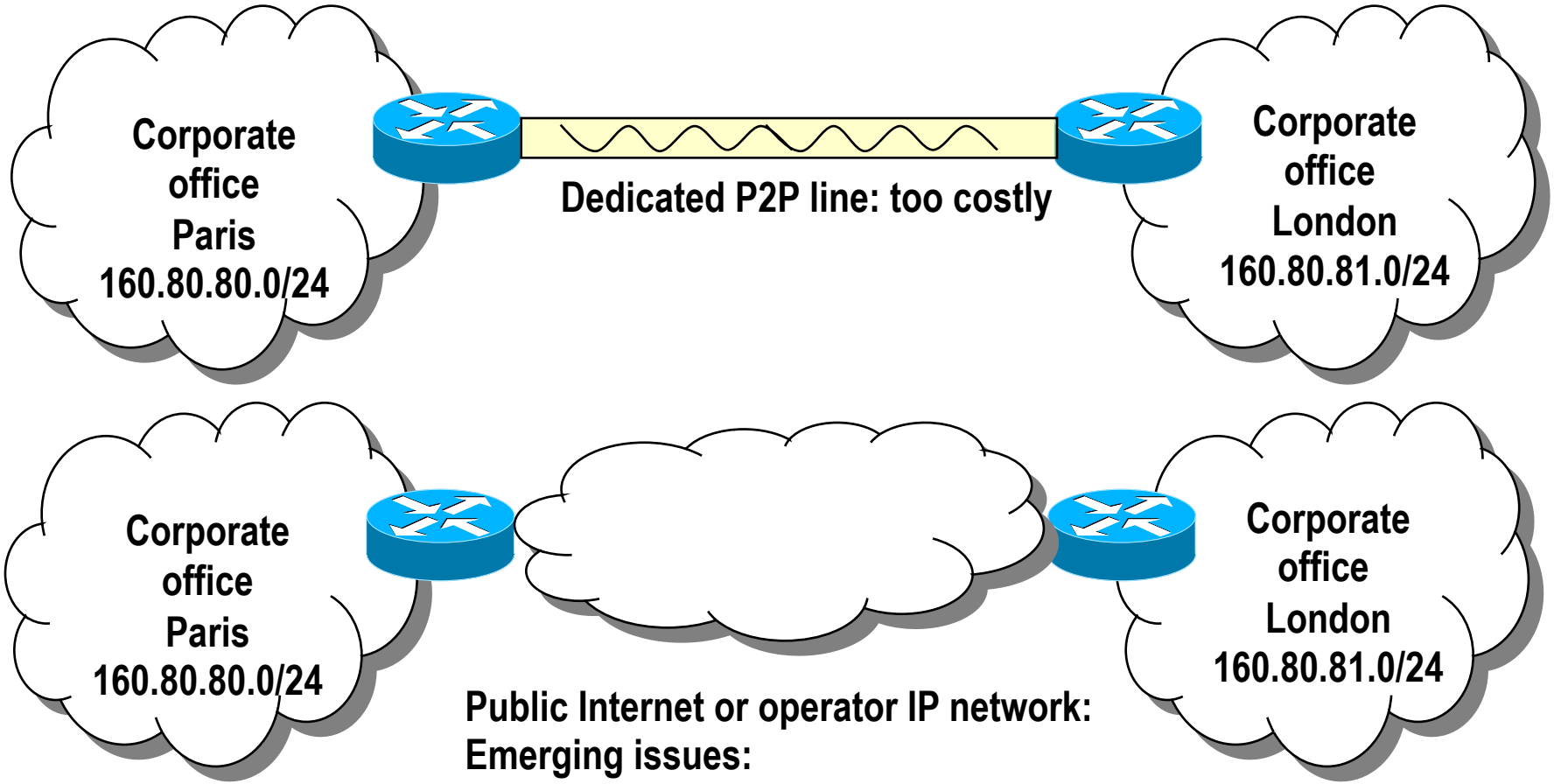
IPsec Basics

Recommended reading: Stallings, Chapter 16
(RFCs are perhaps a bit too complex and extensive for our class – use as extra reading material)

A parenthesis
VPNs: what they are

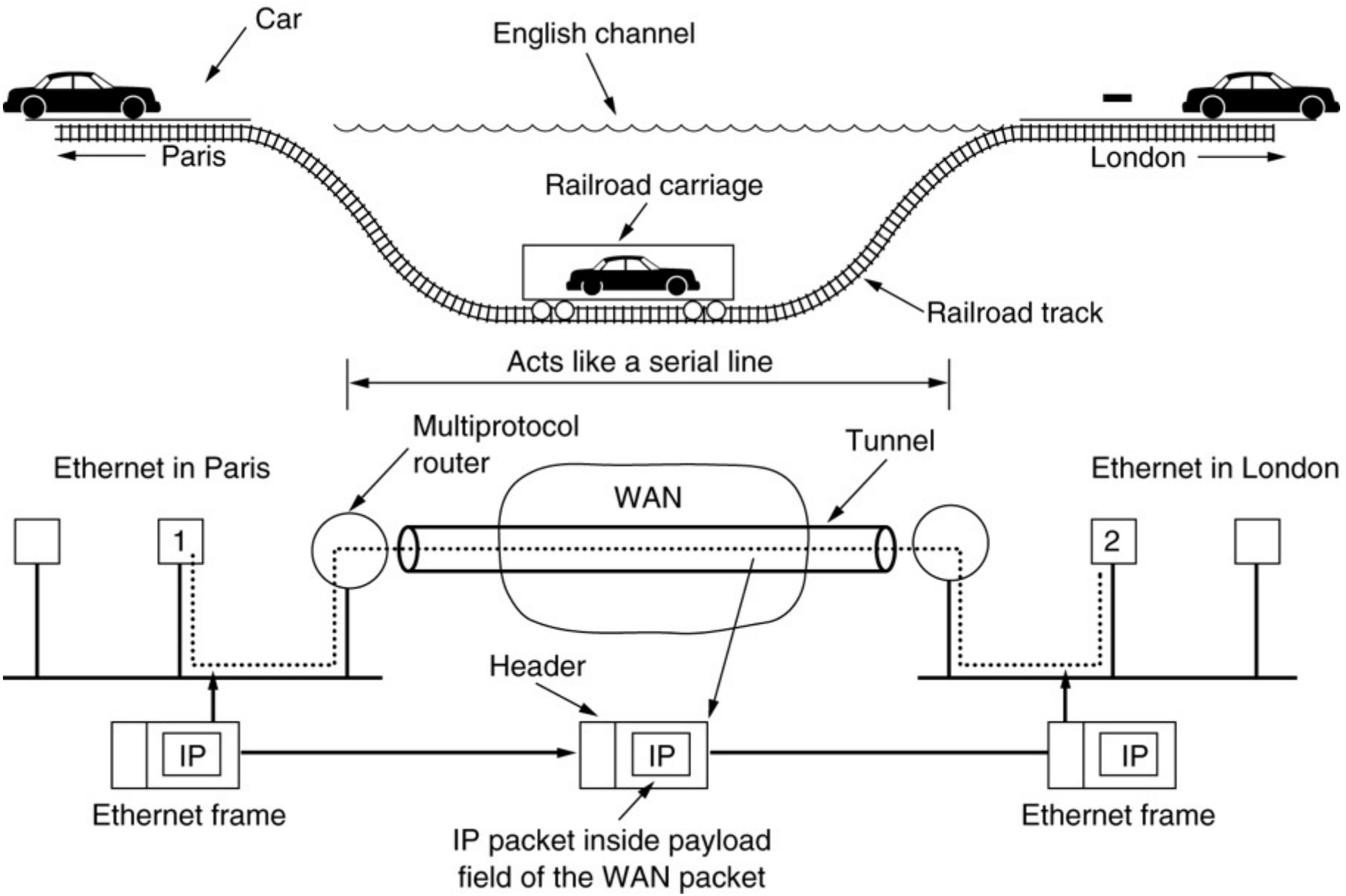
Perhaps out of scope, here, as VPN and IPsec are NOT the same – more later

Virtual Private Networks: why?

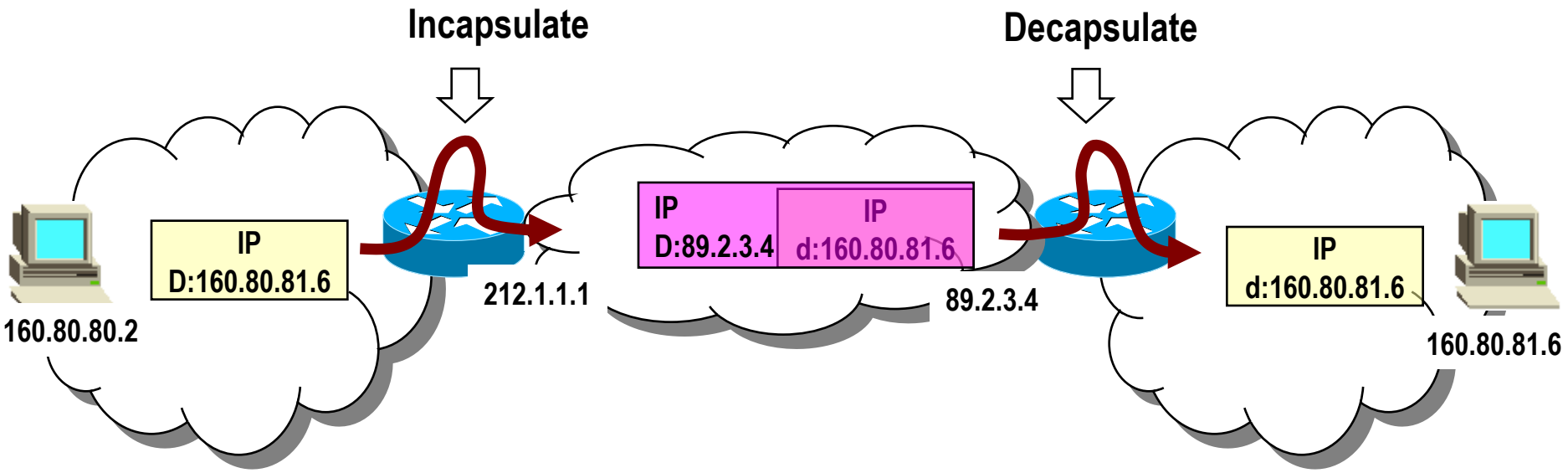


Public Internet or operator IP network:
Emerging issues:
- How to manage private address space across distributed sites?
- How to protect data in transit (especially if public Internet)?

Virtual Networks → tunnels



Virtual Networks over IP



➔ **IP in IP tunnels**

⇨ Not the most effective approach!

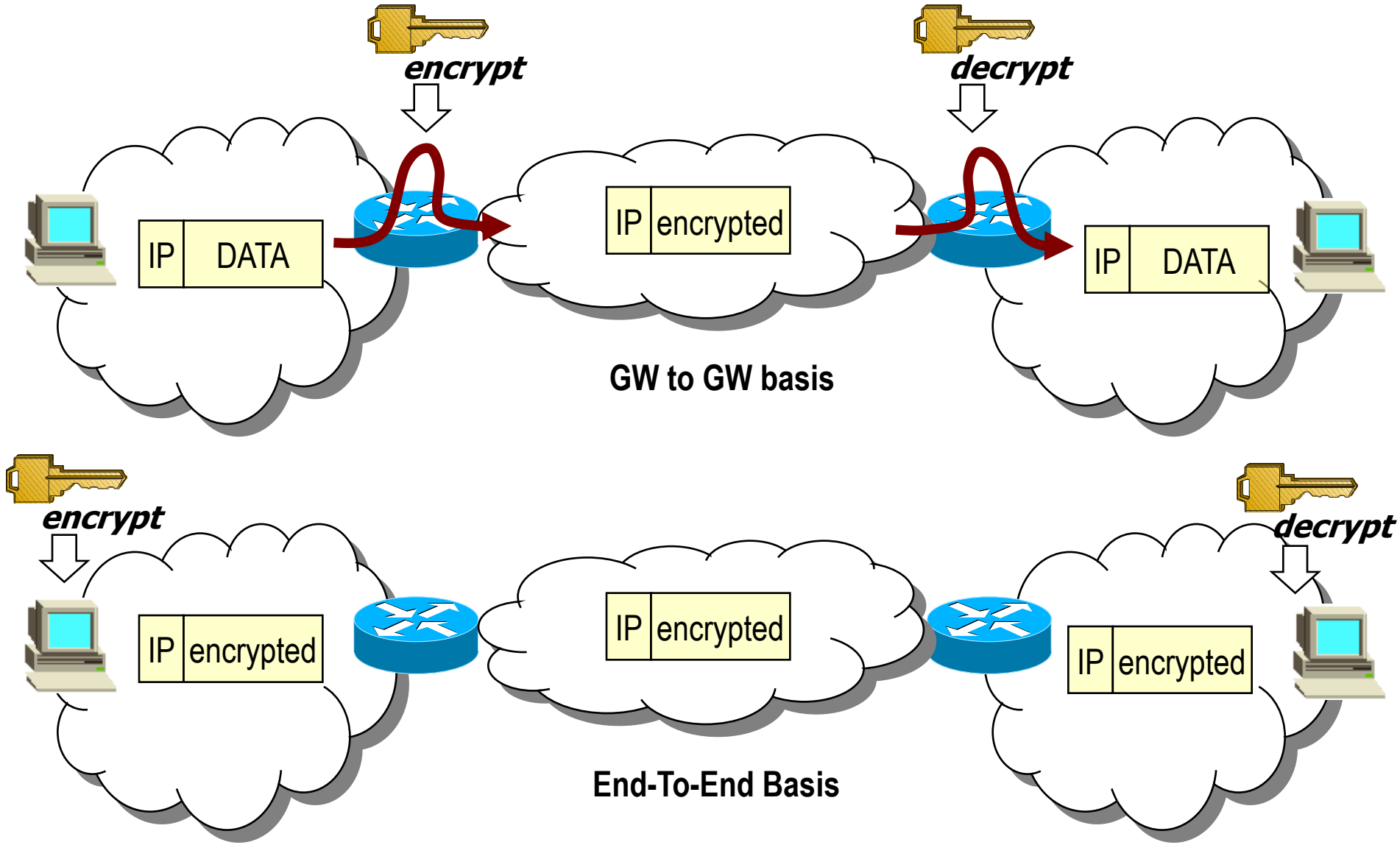
➔ **MPLS tunnels by far more performance effective**

⇨ Typical VPN offer from today operators

» MPLS tunnels alone = VPN without the "P" ☺

» However customer may trust operator (the only one with "hands on" the net)

Private Networks → encryption



Virtual + Private Networks

➔ **VPN =**

- ⇨ Virtual Networks (tunnels)

  \+

- ⇨ Private Networks (authentication,encryption)

➔ **IPsec: a POSSIBLE tool for building VPN**

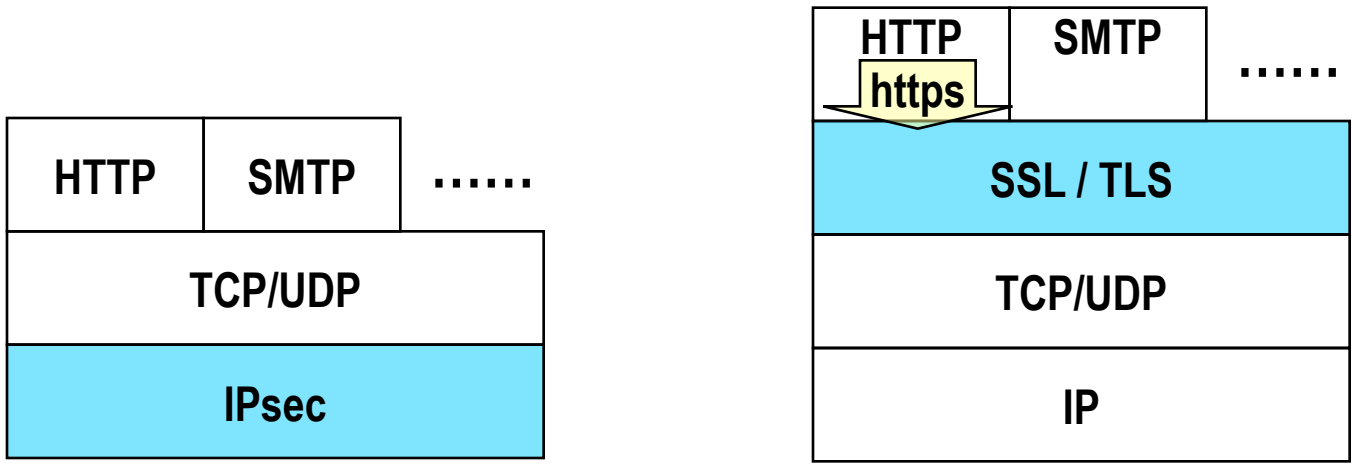
- ⇨ But IPsec and VPNs are NOT synonymous
  - ➔ as some beginner might think
  - ➔ IPsec VPNs not viable when non-IP traffic must be transported!
- ⇨ IPsec: not only tunnels; also e2e encrypted/authenticated transport

➔ **VPN alternatives:**

- ➔ Layer 2: GRE/PPTP, L2TP
- ➔ Layer 3 (actually 3-): MPLS
- ➔ Layer 4 (actually between 4 and 7): SSL tunnels
- ➔ Layer 7: SSH tunnels

IPsec Components

IPsec: layered view



Network layer security

Transport layer security

➔ **IPsec operates with & within IP, at layer 3**

⇨ IPsec & unprotected IP packets do coexist (of course)

➔ **Applications/terminals unaware of IPsec**

⇨ IPsec protects all protocols that rely on IP

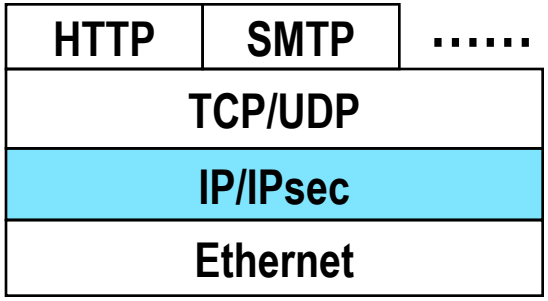
➔ **Protection on a per-host (IP) basis**

⇨ cannot protect SPECIFIC users/applications

IPsec implementation approaches

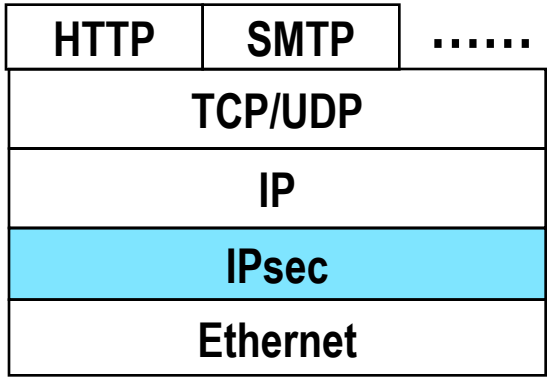
➔ Inside the native IP code

- ⇨ Best approach
- ⇨ But hard to deploy as requires to access and modify IP source code



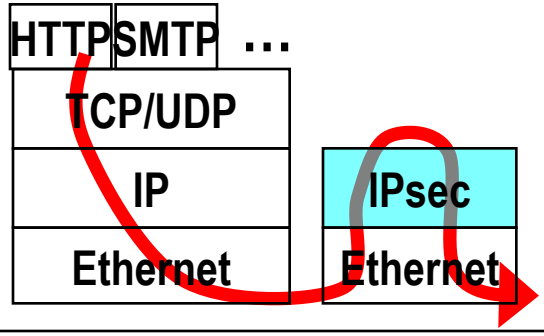
➔ Bump in the Stack (BITS)

- ⇨ Between native IP code and device driver
- ⇨ Deployed in legacy systems



➔ Bump in the Wire (BITW)

- ⇨ Implemented in dedicated hardware
- ⇨ External security processor, acts as gw

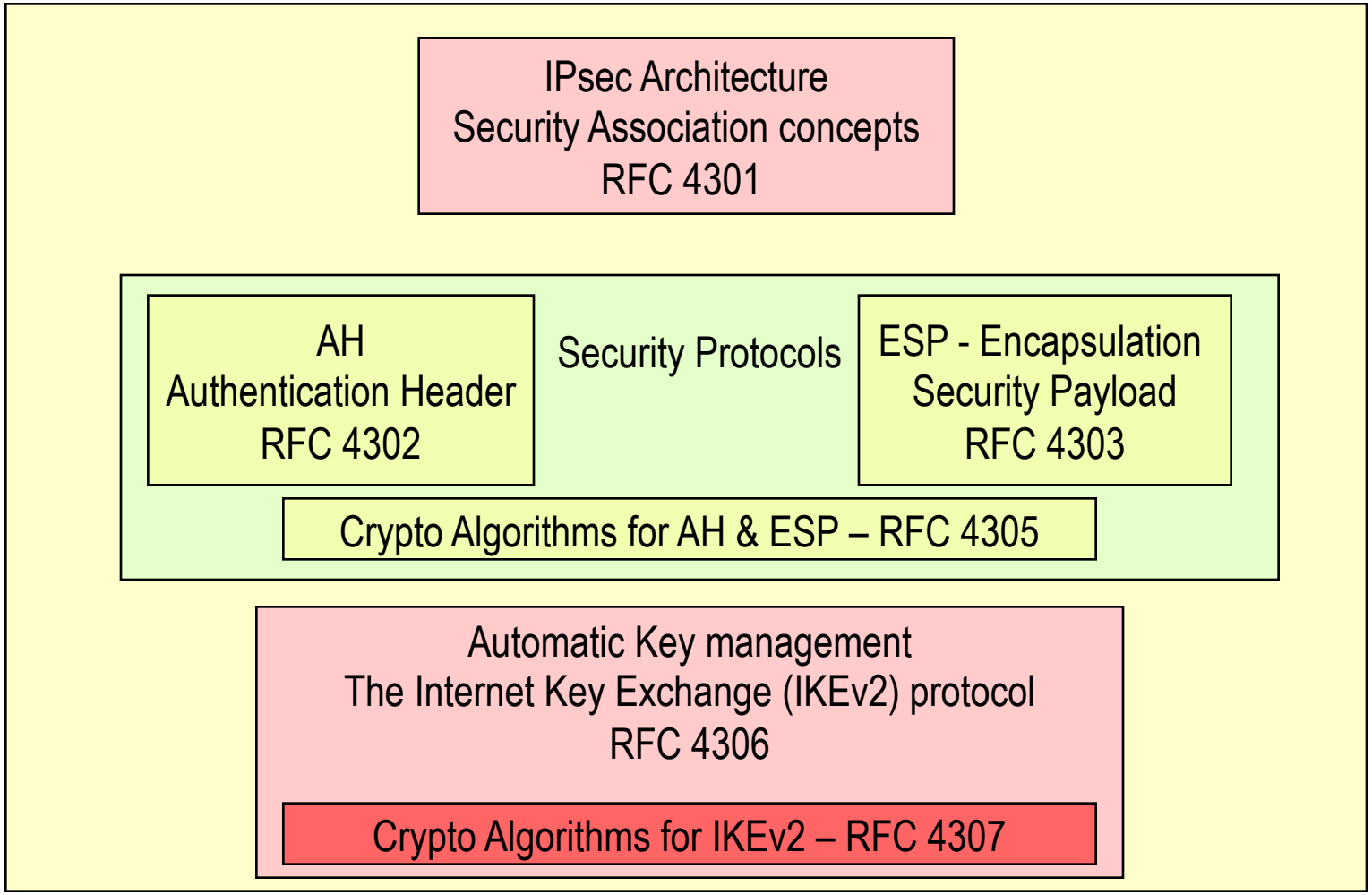


IPsec standardization History

➔Three major "series" of RFCs

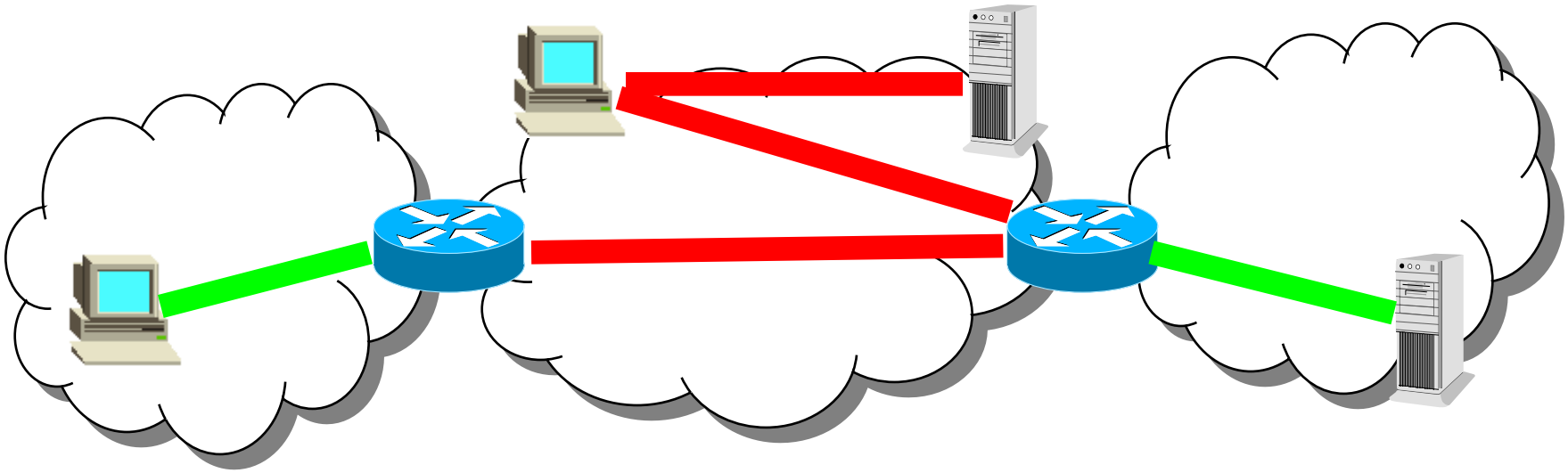
- ⇨Serie 1: RFC 1825-1827 (august 1995)
  - ➔IPsec concepts first drafted
- ⇨Serie 2: RFCs 2401-2412 (november 1998)
  - ➔Significant revision of ALL the IPsec architecture
  - ➔Describes IPsec as we know it today
- ⇨Serie 3: RFC 4301-4307 (december 2005)
  - ➔Born after long discussion in WG (almost 5 years)
  - ➔basically touches/extends all the IPsec architecture
  - ➔Most important: major revision of IKE (Internet Key Exchange protocol)
    - » IKEv2 now simplifies and glues several protocols (ISAKMP, IKE, Oakley) into one

IPsec RFCs

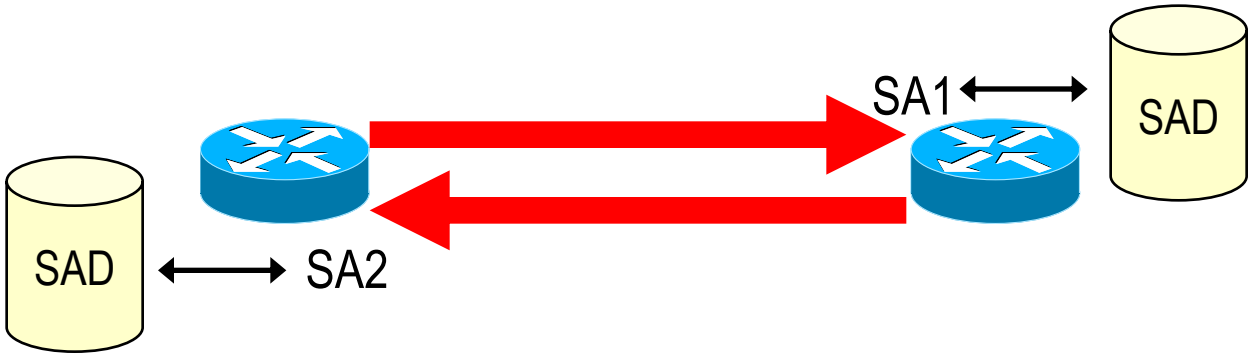

IPsec Architecture
Security Association concepts
RFC 4301
AH
Authentication Header
RFC 4302
Security Protocols
ESP - Encapsulation
Security Payload
RFC 4303
Crypto Algorithms for AH & ESP – RFC 4305
Automatic Key management
The Internet Key Exchange (IKEv2) protocol
RFC 4306
Crypto Algorithms for IKEv2 – RFC 4307

Security Association

- **Fundamental concept in IPsec**
- **May involve:**
  - Host to host
  - Host to intermediate router (security gateways)
  - Security gateway to security gateway
- **Defines the boundaries for IP packets authentication/encryption**
  - A “connection” with security services active



SA: monodirectional!



➔ **SPI = Security Parameters Index**

⇨ The (somewhat) unique "name" of an SA

➔ **Destination Address based**

⇨ Security Association managed at the receiver side

➔ **SAD = Security Associations Database**

⇨ SPI = search key (at least)

⇨ Stores set of security services per each SA, and related parameters

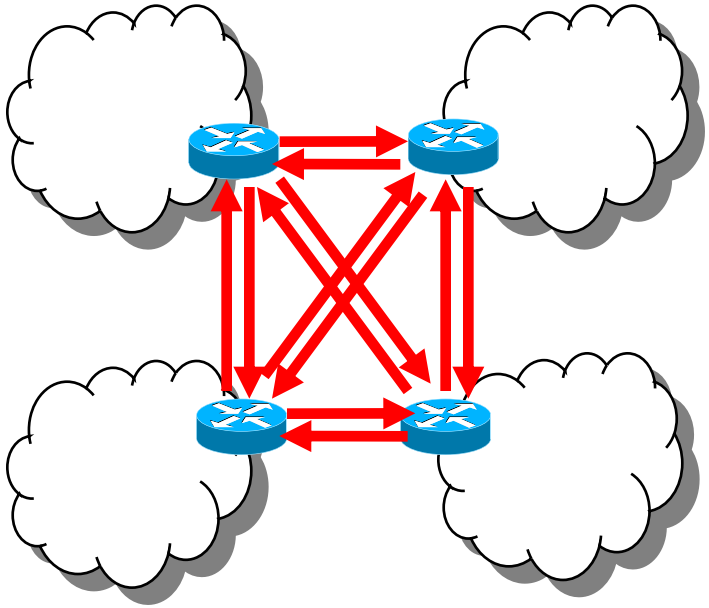
➔ E.g. which encryption algorithm; shared key for encryption, SA lifetime, Sequence number counter, etc

SPI and SAD lookup

- **Security Parameters Index = 32 bit index**
- **Identifies a Security Association**
  - Uniquely, but in conjunction with destination address and possibly source address
    - Note: in multicast destination address differs from receiver address!!
  - Used to lookup into Security Association Database and retrieve security parameters
    - E.g. AH: Authentication algorithm, shared key,
  - Lookup rule: "longest match"–like
    - First look up SPI, Dest Addr, Src Addr
    - Then look up SPI, Dest Addr
    - Then Look up SPI
      - » Implementations may also specify AH/ESP during lookup

Security Association and Key management

- **Manual**
  - Manually configure each SA and related crypto keys
    - static, symmetric
  - Typical in small-scale VPNs
    - Few security gateways, e.g. one per site
    - Meshed SA connections
- **Automatic**
  - SA management through IKEv2
    - In the past, through the combined operation of several protocols
      - IKE+ISAKMP+others
  - On-demand SA creation
  - Session-oriented keying/rekeying



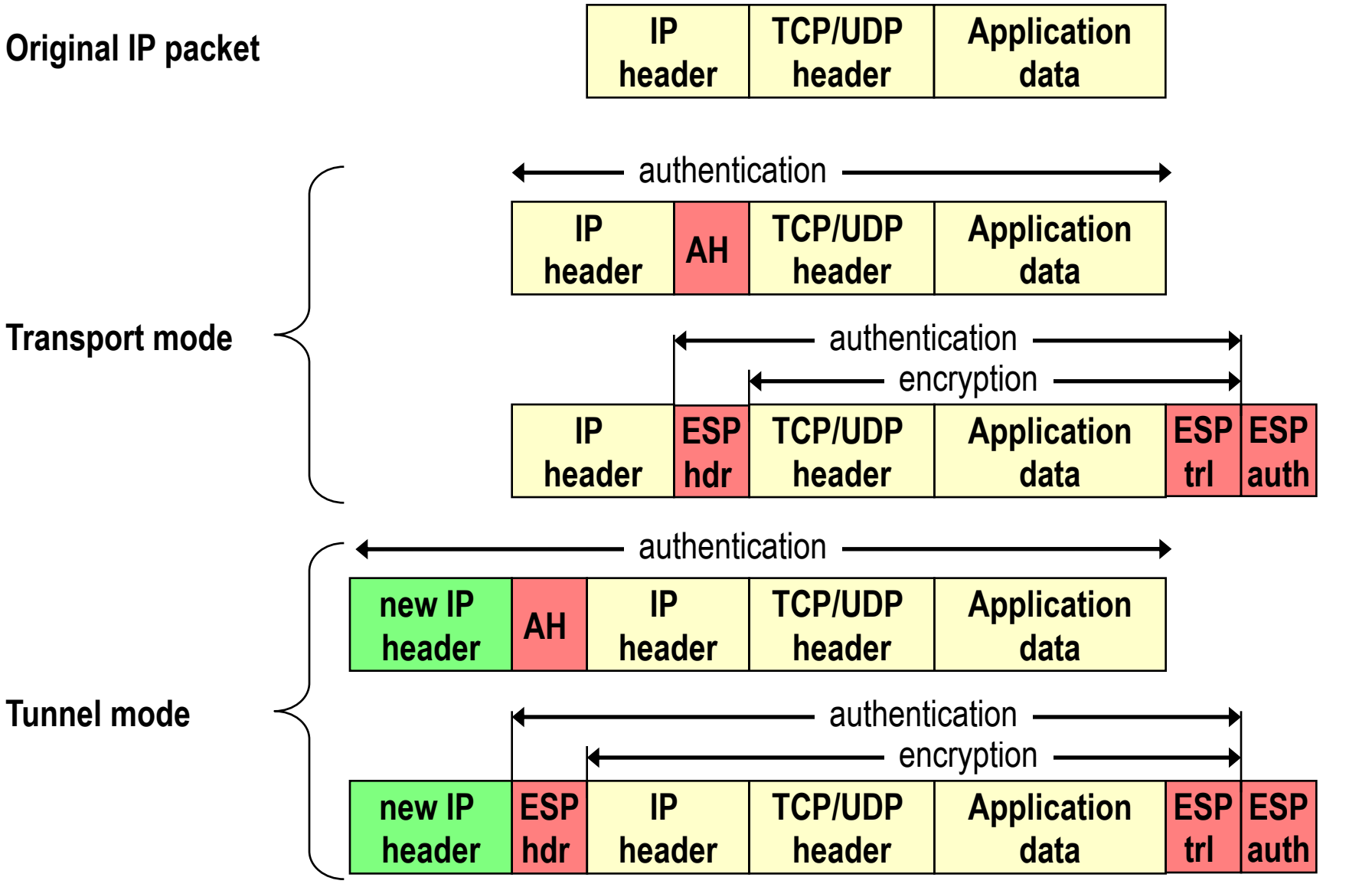
IPsec Security protocols

- **AH (Authentication Header)**
  - Authentication (whole packet) only
- **ESP (Encapsulated Security Payload)**
  - Encryption, Authentication, both
    - Unlike AH, payload only authentication, no header
    - Not an issue, when tunnel mode used

- **ESP in most cases is the only one needed**
  - Mandatorily supported in any IPsec implementation, unlike AH
    - RFC 4301: AH downgraded: from MUST to MAY (be supported)
      - Quoting from RFC 4301: *"Experience has shown that there are very few contexts in which ESP cannot provide the requisite security services. Note that ESP can be used to provide only integrity, without confidentiality, making it comparable to AH in most contexts."*
  - AH and ESP may be combined together, if needed (rarely)

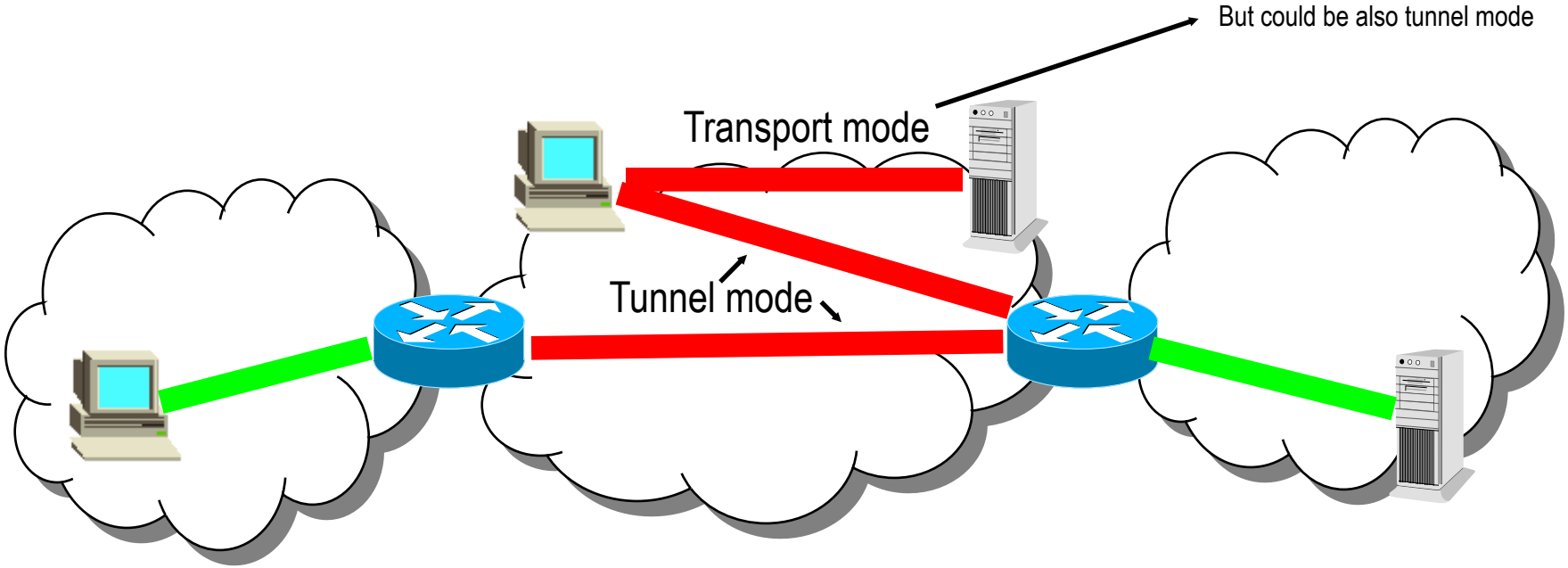
- **Transport mode and tunnel mode for both**

Transport vs Tunnel – AH and ESP

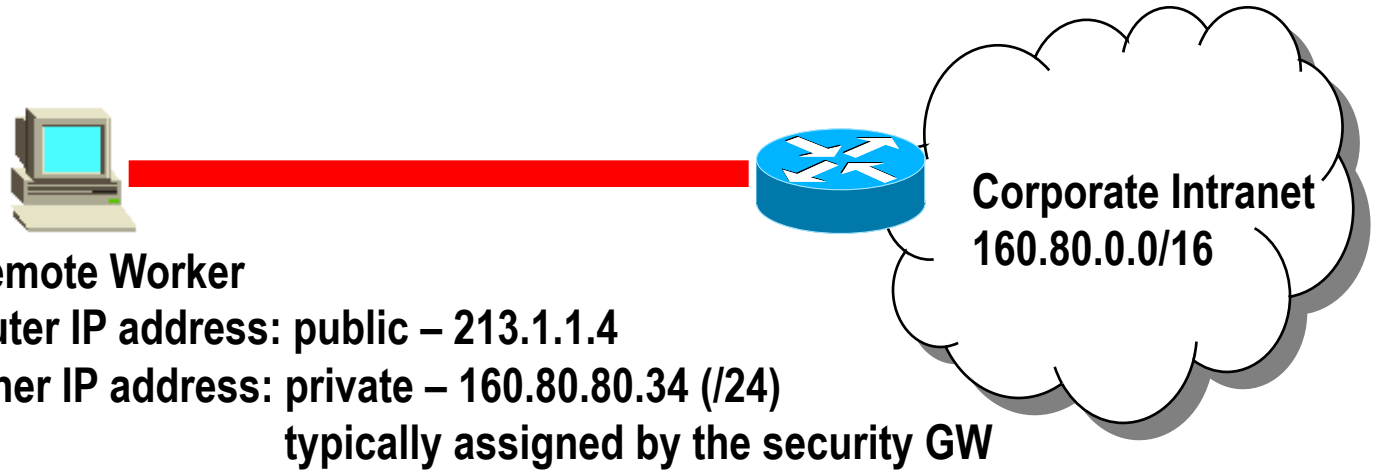


When transport? When Tunnel?

- **Transport mode: end-to-end**
- **Security Gateways can use transport mode only for connections originating/terminating there**
  - ⇨ Not when they are intermediary between host and server!
  - ⇨ Tunnel mode used in almost all the cases



A note on host-to-gw tunnels



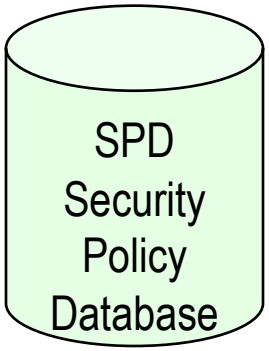
Remote Worker
Outer IP address: public – 213.1.1.4
Inner IP address: private – 160.80.80.34 (/24)
typically assigned by the security GW

➔Using a private IP address inside the tunnel:

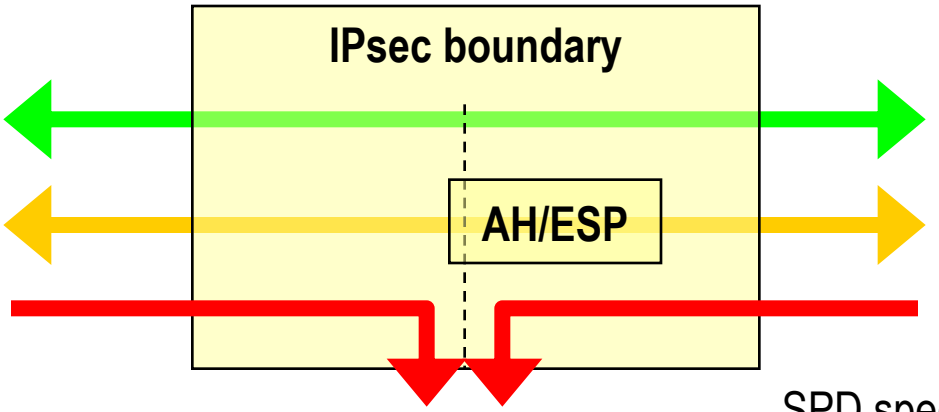
- ⇨Allows to access to all services provided in the intranet, exactly like in the case the worker is connected inside the corporate
- ⇨Allows to be protected by the corporate firewall (all traffic destined to 160.80.80.34 MUST be first routed to the corporate subnet and then routed to the end user in a protected fashion

IPsec protection & access control

Protected Interface
- **Internal Network with IPsec Security GW;**
- **Host OS with IPsec Host**



Unprotected Interface
- **The external network**



BYPASS

PROTECT

DISCARD

Three rules

Outbound Traffic

Inbound Traffic

SPD specifies:
- Which security protocols (AH, ESP)
- On which packets/traffic relations
- Which mode (tunnel, transport)
- Which granularity
  - single shared tunnel
  - different tunnels per TCP conn

Lecture 5.2:

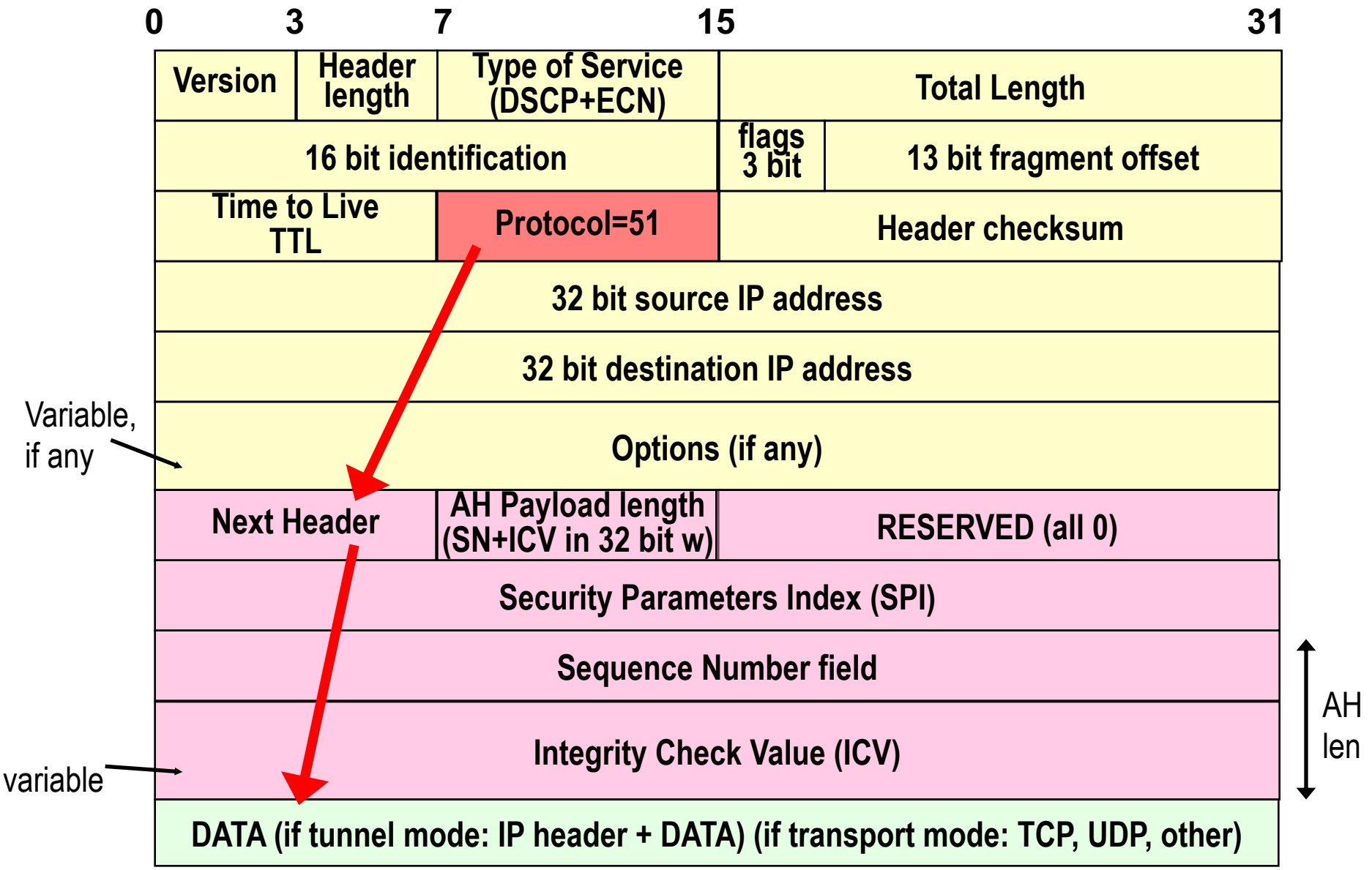
IPsec Security Protocols AH/ESP

(IPv4 only)

Services provided

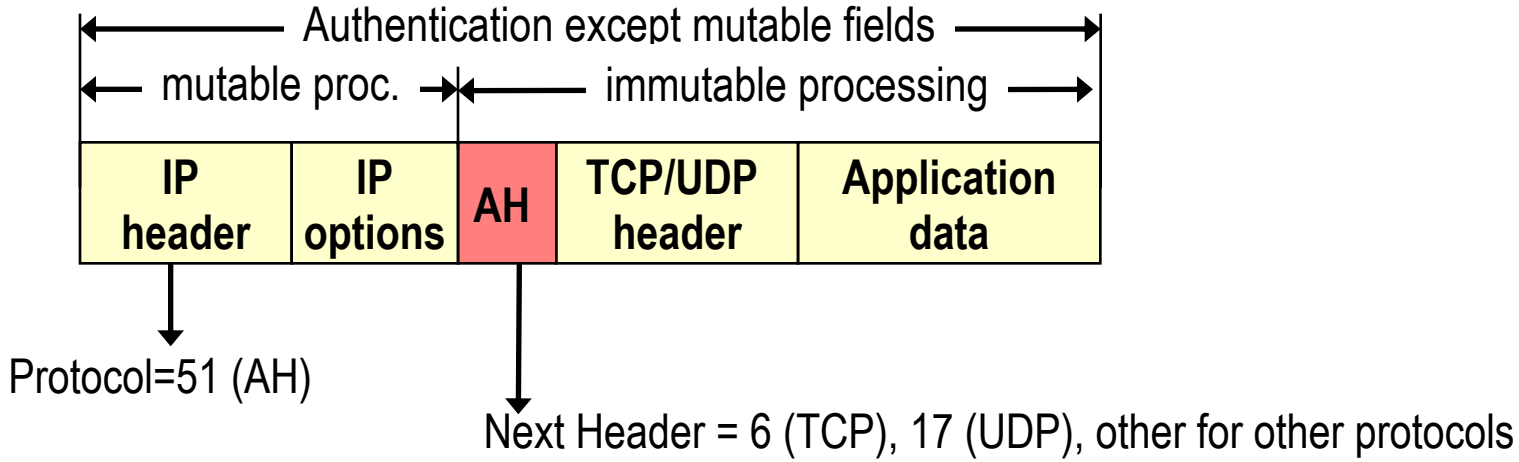
- **Authentication Header**
  - Integrity and data origin authentication
    - Authentication covers both payload and parts of IP header that do not modify in transfer
  - Protection against replays
    - Optional, through extended sequence numbers
- **Encapsulated Security Payload**
  - Same services as AH
    - Though authentication limited to IP payload
  - Confidentiality through encryption
  - Traffic flow confidentiality
    - Improved privacy against eavesdropping
    - Through padding and dummy traffic generation

Authentication Header

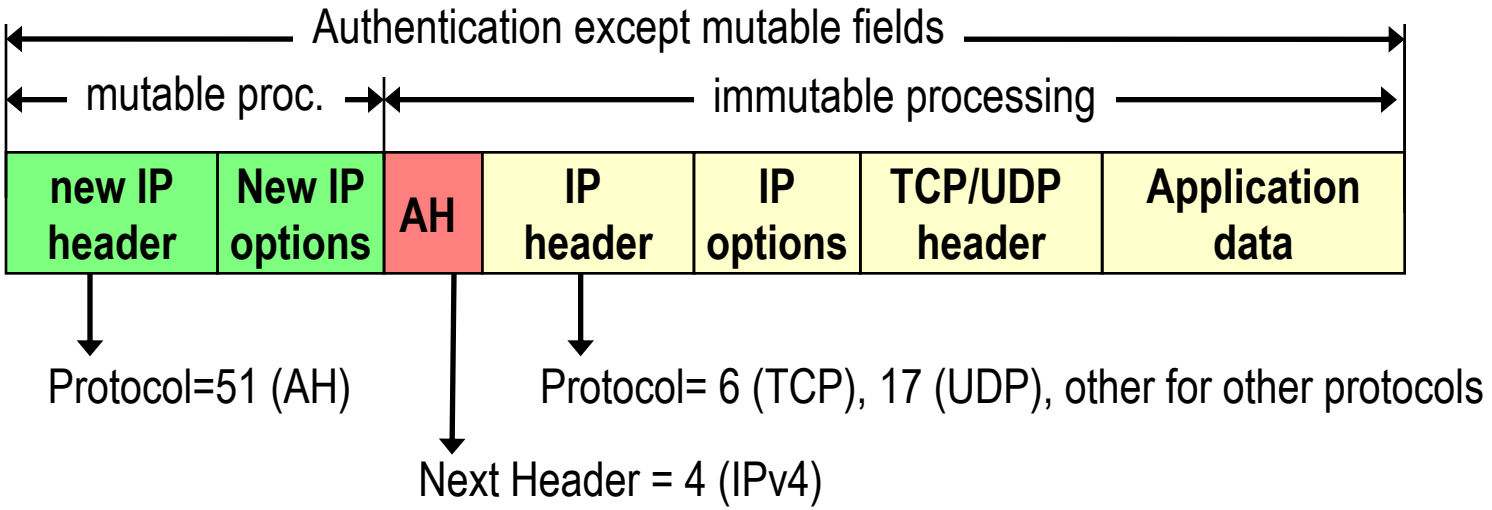


Transport mode, tunnel mode

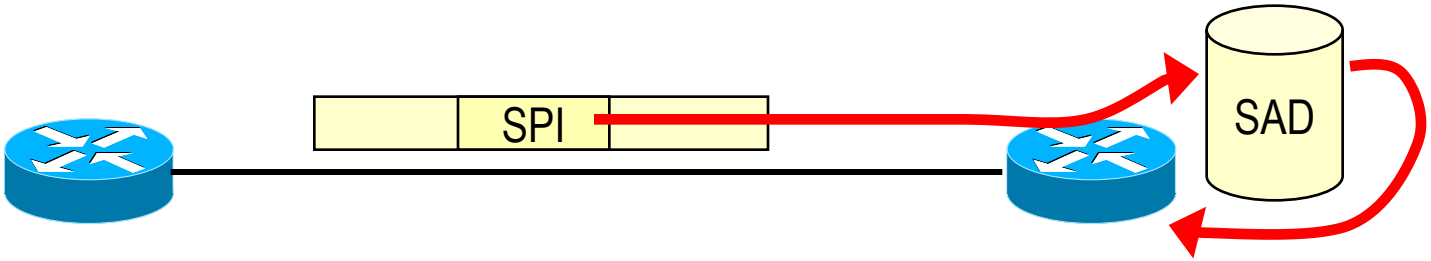
Transport mode:



Tunnel mode:



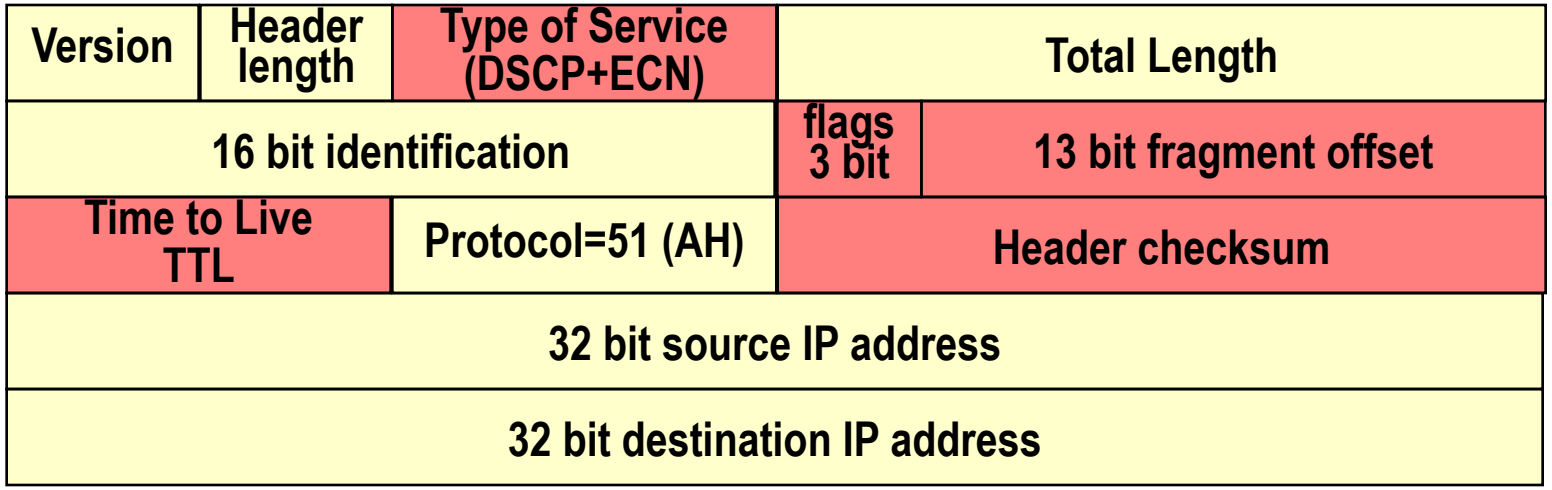
Security Parameters Index



- **32 bit index**
- **Used to lookup the SAD at destination**
  - Lookup also uses
    - destination address
    - source address
    - security protocol (AH/ESP)
- **Retrieves algorithms and parameters that allow to process received packet**

Integrity Check computation

- **Only on immutable fields in the IP header**
  - Or mutable but predictable
    - e.g. destination address with strict/loose source routing option
- **Mutable fields set to 0 during MAC computation**
  - Highlighted in red in next figure
    - Note: AH apply before fragmentation, and checked after reassembly
- **Options classified as either mutable or not**
    - Mutable options: details in appendix A RFC 4302
    - mutable options = all zeroed



auth algorithm and ICV

- **2007+ implementation requirements [RFC 4835]**
- **MUST support:**
  - HMAC-SHA-1-96
    - First 96 bits of standard HMAC-SHA-1 → 20 bytes → 160 bits
- **SHOULD+ support:**
  - AES-XCBC-MAC-96
    - see description in RFC 3566
- **MAY support:**
  - HMAC-MD5-96
    - First 96 bits of standard HMAC-MD5 → 16 bytes → 128 bits
- **Improved algorithms available (optional)**
  - HMAC-SHA-256/384/512
    - First nnn/2 bits
- **ICV must be multiple of 32 bits**
  - Padding necessary if MAC not multiple of 32
    - Arbitrary padding chosed at sender

Why sequence number?

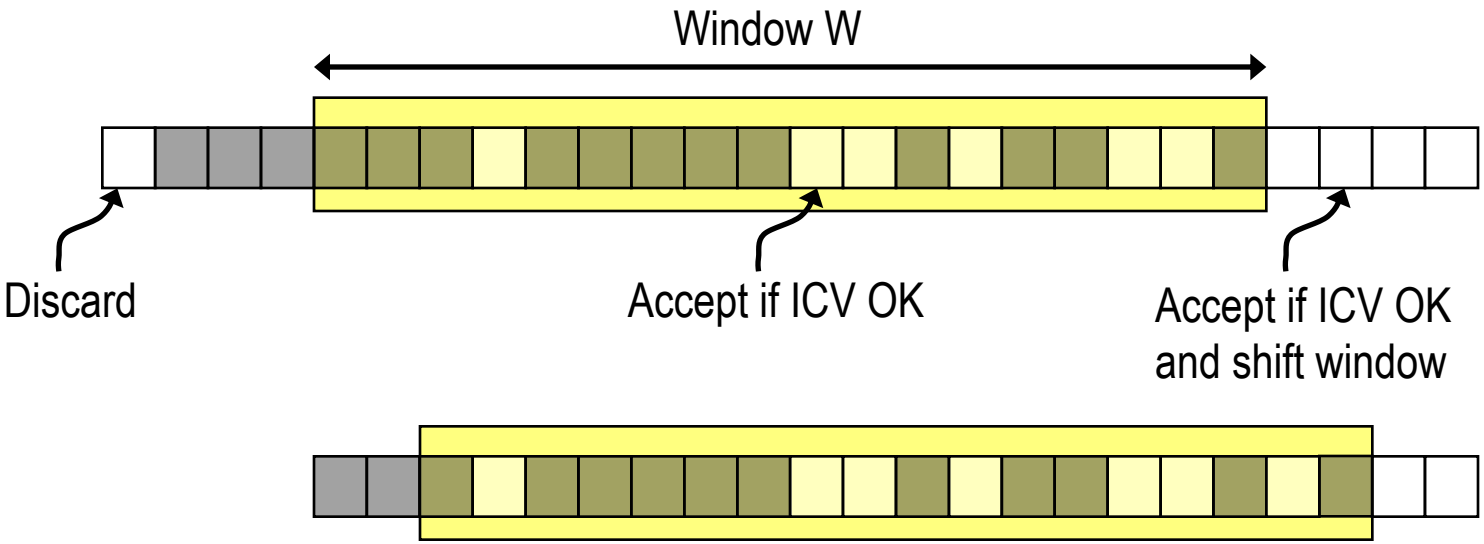
- **IP header DOES NOT contain a sequence number!**
  - Hence replay of an authenticated IP packet is possible
    - And may alter in an umpredictable manner the overlaying service (e.g. ICMP replies can be dangerous ☺)
- **Sequence number: 32 bit counter**
  - Initialized to 0 when the Security Association is established
  - Increments of 1 per each transmitted packet
    - First transmitted packet: SN=1
  - Maximum value 2^{32}-1, afterwards Security Association must be terminated
    - No counter cycling allowed when anti-replay service active
    - Anti-replay: optional (but default = on)
      - » Anti-replay typically OFF when manual (static) keys configured

Extended Sequence Number

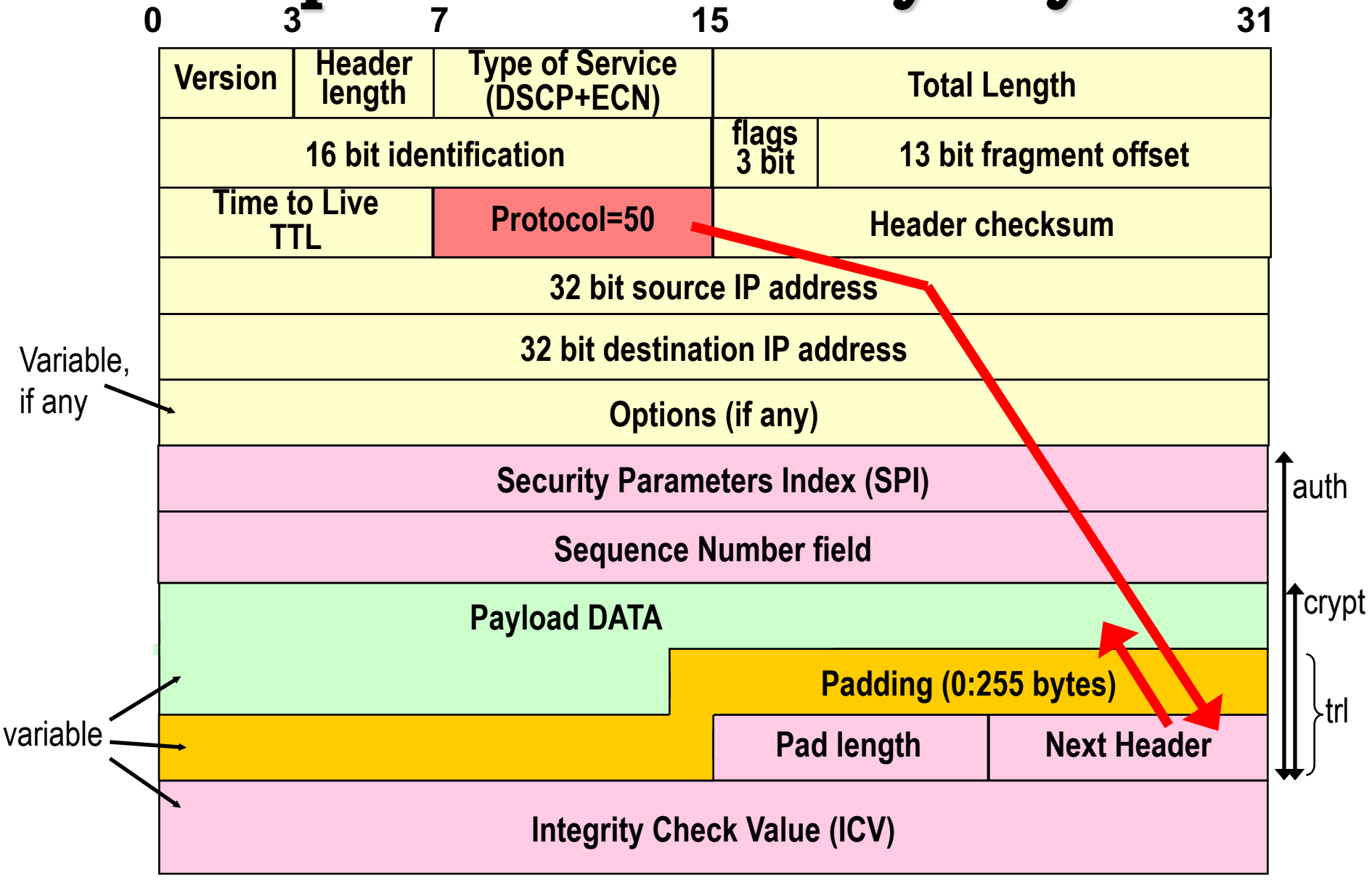
- **2^{32} ~ 4.3 billion**
  - A lot, but not REALLY al lot!
    - Packet size = 1500 (1460 bytes payload)
    - 2^{32} x 1460 bytes = 6270 GB
    - About 14 h transmission of a 1 gbps link
- **Extended Sequence Number:**
  - 64 bits - this should be enough, now ☺
  - Transmit only low order 32 bits
  - But use high order 32 bits in ICV computation!

Anti-replay

- **Sliding Window W**
  - Size locally decided at receiver
    - Minimum = 32; default = 64; higher values recommended for high speed links
    - eventually very large: maximum 2^{31}-1 with SN and 2^{32}-1 with ESN
  - Window right margin = highest NS packet received
- **Duplicates discarded**
- **Packets out of left window edge discarded**
- **Packets greater than right window margin make W shift**

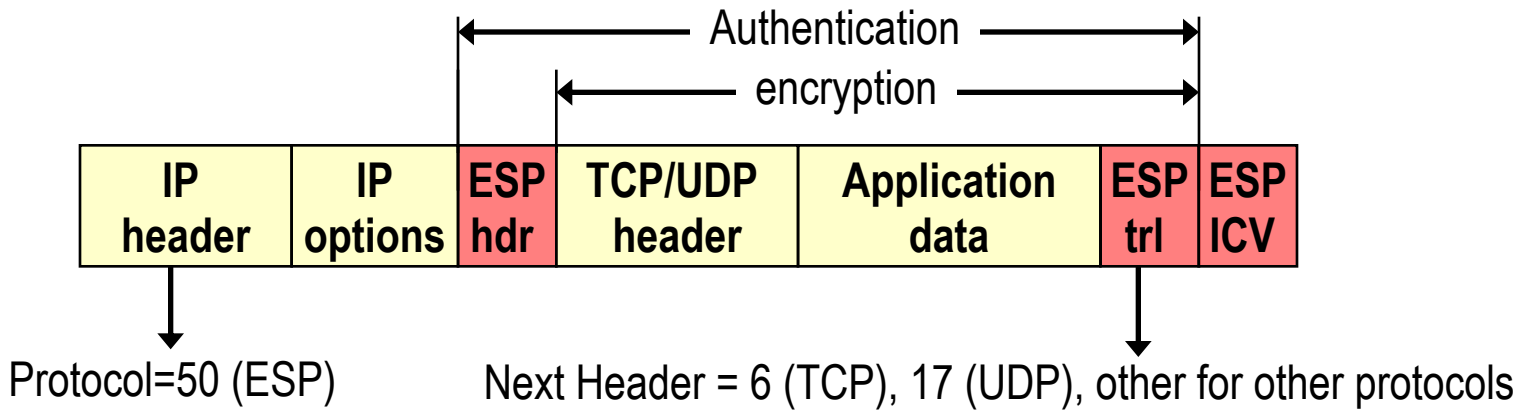


Encapsulated Security Payload

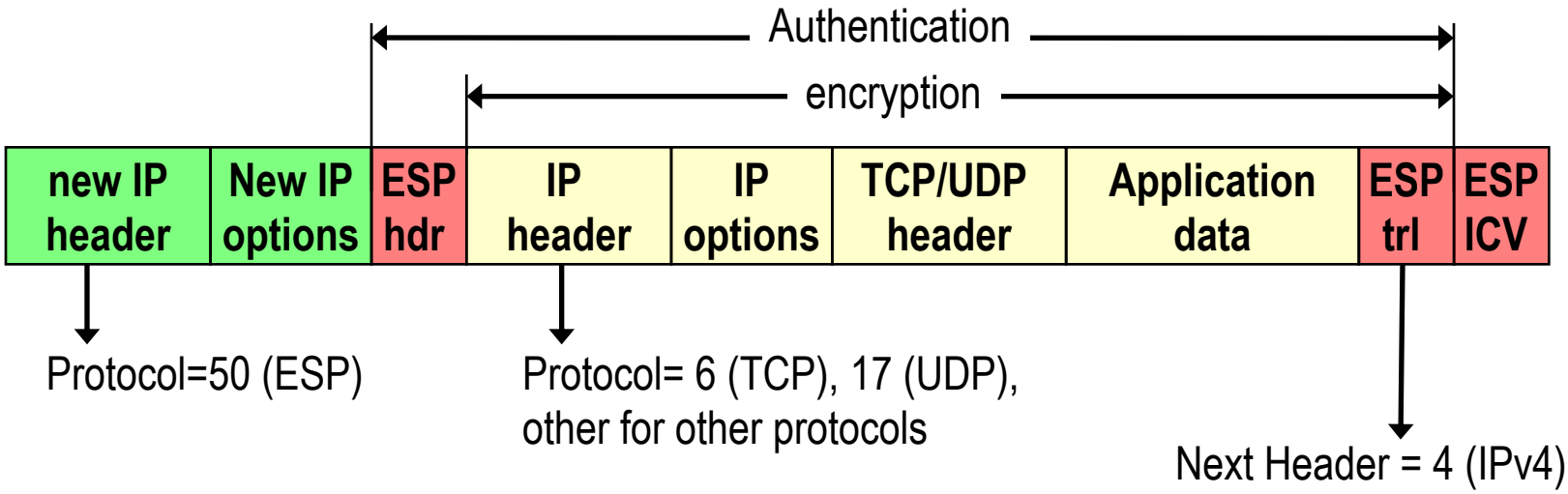


Transport mode, tunnel mode

Transport mode:



Tunnel mode:



ESP Header & Trailer

- **ESP header**
  - 8 bytes, SPI + SN
    - Same as AH (including ESN and optional anti-replay)
    - Plain text
- **ESP Trailer**
  - Padding, variable size + pad length + next header
    - Pad length = 1 byte = 0:255
    - Next Header = 1 byte = type of data payload
      - » Next Header = 4 → Tunnel mode (IP header inside)
      - » Next Header = 59 → Dummy packet!!
  - Padding: for two reasons
    - → Encryption algorithm may require plaintext to be a multiple of some number of bytes
      - » E.g. block size of a block cipher
    - Resulting ciphertext must terminate on a 4-byte boundary

Encryption & authentication algos

- **2007+ implementation requirements [RFC 4835]**
- **Authentication: same as AH**

- **MUST support:**
  - AES-CBC-128 [RFC 3602]
- **MUST- support:**
  - 3DES-CBC [RFC 2405]
- **SHOULD support:**
  - AES-CTR [RFC 3686]
- **SHOULD NOT (!) support**
  - DES-CBC

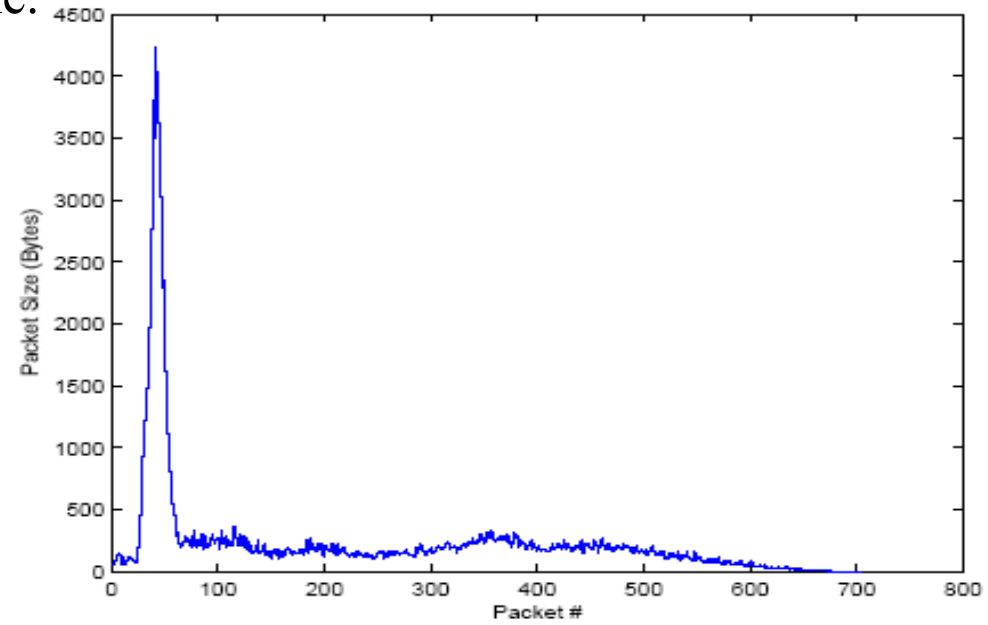
- **Combined mode (authenticated encryption)**
  - Not suggested or recommended, but mentioned as of interest – no ICV = lower overhead
    - AES-CCM [RFC 3686] - used also in 802.11i
    - AES-GCM [RFC 4106]
  - Both could support Authenticated Encryption with Associated Data
    - AEAD – would yield AH and ESP in a same cipher!!

Encryption & IP unreliability

- **A problem: some good encryption algorithms need to maintain synchronization**
  - E.g. Cipher Block Chaining, etc
- **However IP packets may arrive out of order or may be lost!**
- **Consequence: for some encryption algorithms, packets must carry data needed to resync the deciphering**
  - If such data (per-packet Initialization Vector) necessary, IV prepended to DATA payload
    - Not an ADDITIONAL ESP header! i.e. it is "invisible" to ESP

Traffic Flow Confidentiality Service

- **Encryption does not guarantee unlinkability!**
- **Linkability via statistical traffic pattern analysis**
- **Traffic source (e.g. web-site) fingerprinting**
  - E.g. sample size profile for www.amazon.com
    - Details in "Privacy Vulnerabilities in Encrypted HTTP Streams, by Bissias, Liberatore, Levine.
  - Many other traffic/protocol fingerprinting approaches (literature + pre-prints)



ESP Traffic Flow Confidentiality

- **ESP TFC: Two counter-measures against traffic analysis attacks**
  - Ability to alterate packet size
    - Supplementary padding facility added to data payload
      - » Easy to manage in tunnel mode, as inner packet size is known in the IP/TCP header
    - Native (up to) 255 bytes padding insufficient
      - » And why messing up by mixing TCF padding with that necessary for encryption?

  - Ability to generate "dummy" packets
    - Example: one can transform a VBR traffic into CBR
      - » Heavy on the network load, though: reduces statistical multiplexing effectiveness

Lecture 5.3:

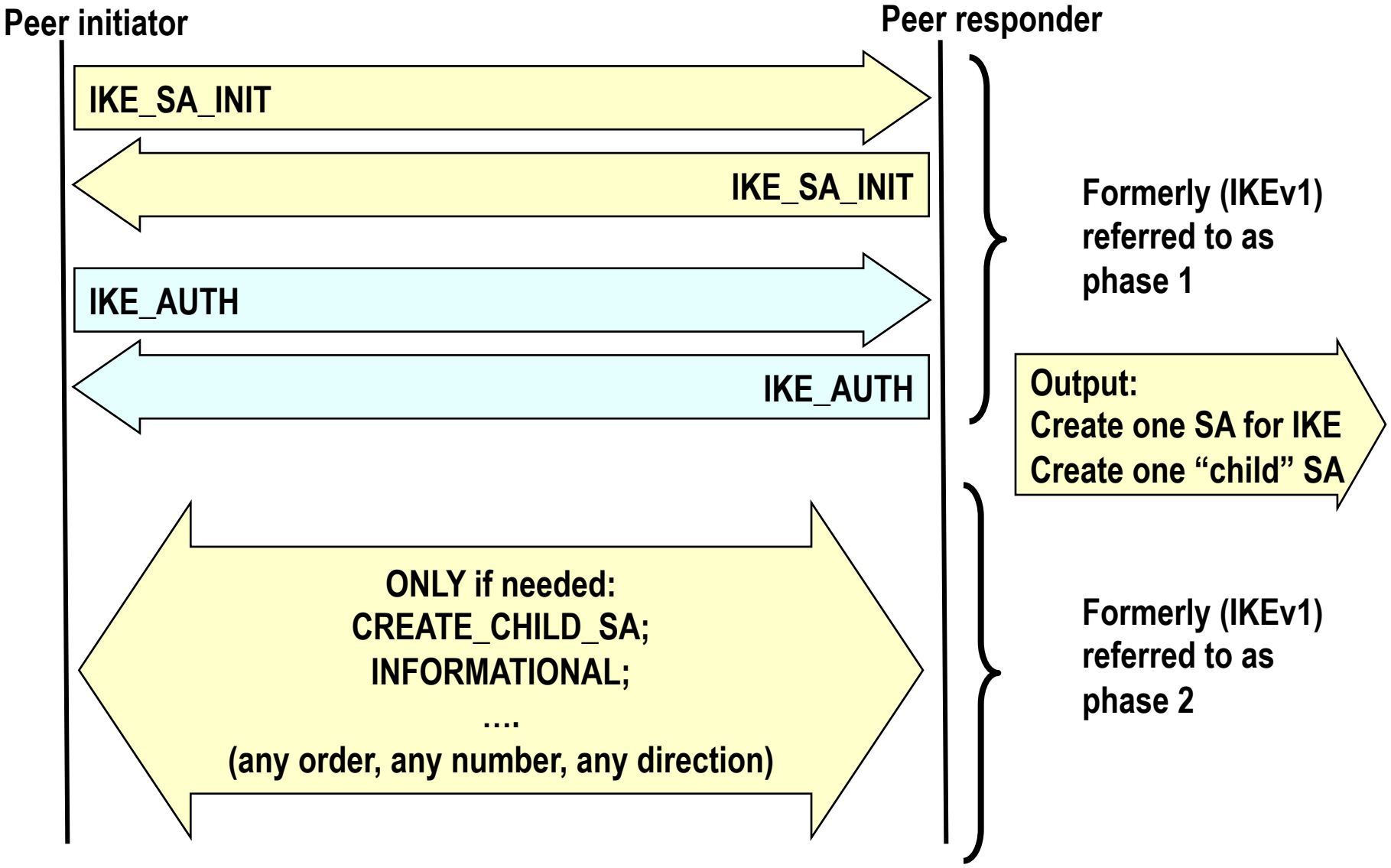
IKEv2

Note: minor updates in RFC 5996, following material based on RFC 4306

Rationale for IKE

- **shared state must be maintained between source and sink**
  - Which security services (AH, ESP)
  - Which Crypto algorithms
  - Which crypto keys
- **Manual maintenance not scalable**
  - Partially OK only for small scale VPNs
  - In any case, weak approach
    - Infinite lifetime SA → no rekeying!
- **IKE = Internet Key Exchange protocol**
  - Goal: dynamically establish and maintain SA
  - IKE now (september 2010, RFC 5996) in version 2
    - Replaces protocols specified in RFCs 2407, 2408, 2409 (IKE, ISAKMP, DOI)
    - extends IKEv2 in RFC 4306
    - IKEv2 quite different (and much cleaner!!) than former specifications

IKE phases at a glance

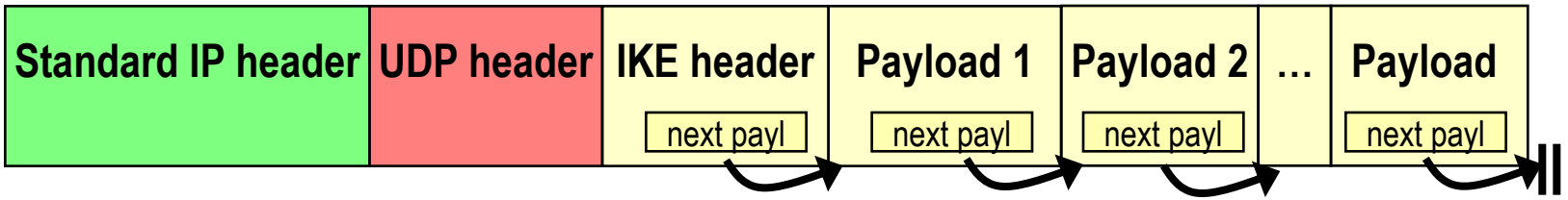


IKE SA and CHILD SA

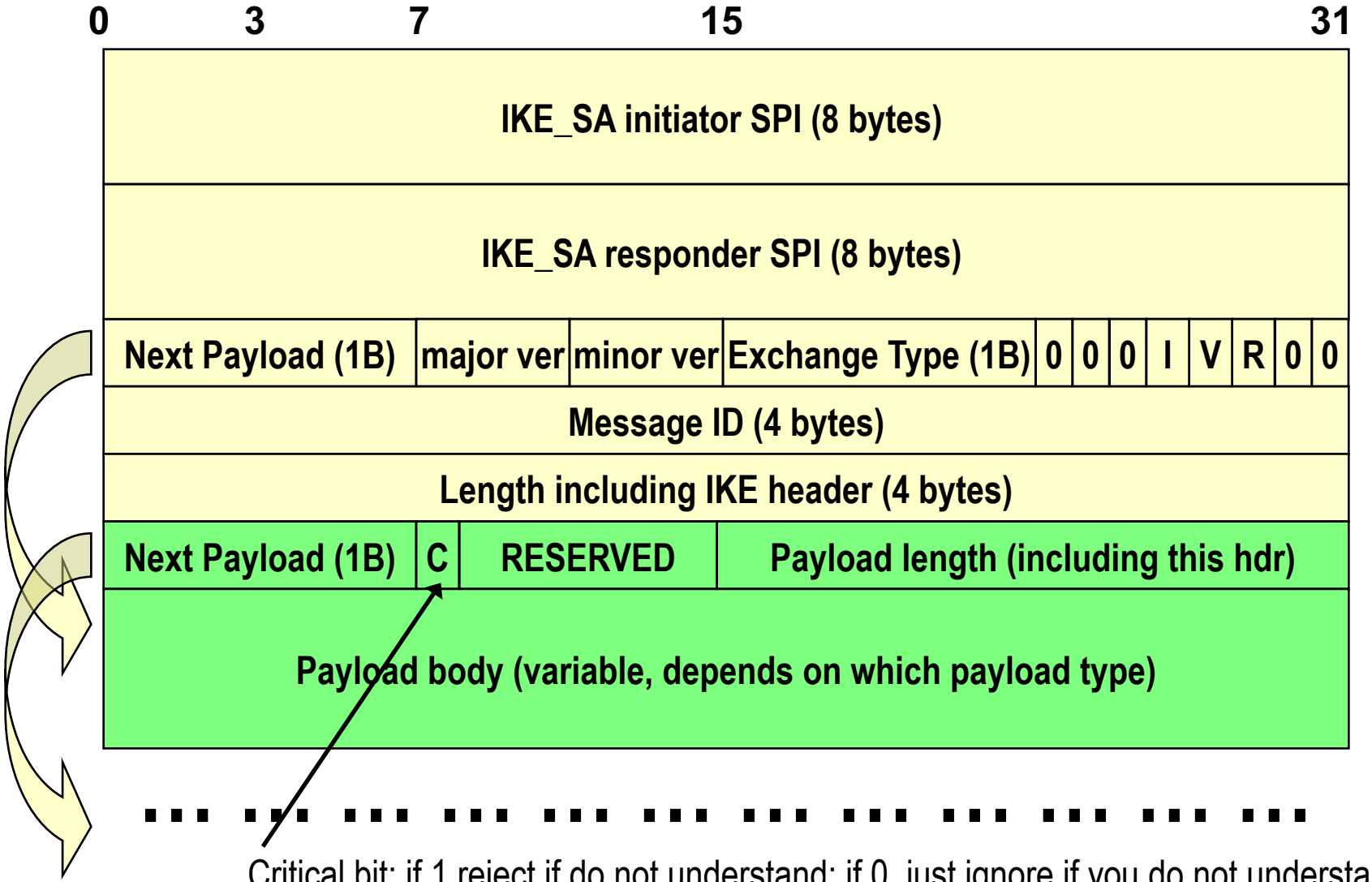
- **IKE SA:**
  - Security association to exchange IKE messages (control messages)
- **CHILD SA**
  - Security association to exchange data messages
    - Making use of AH or ESP
  - Many CHILD SA may be set up between two peers

IKE message format

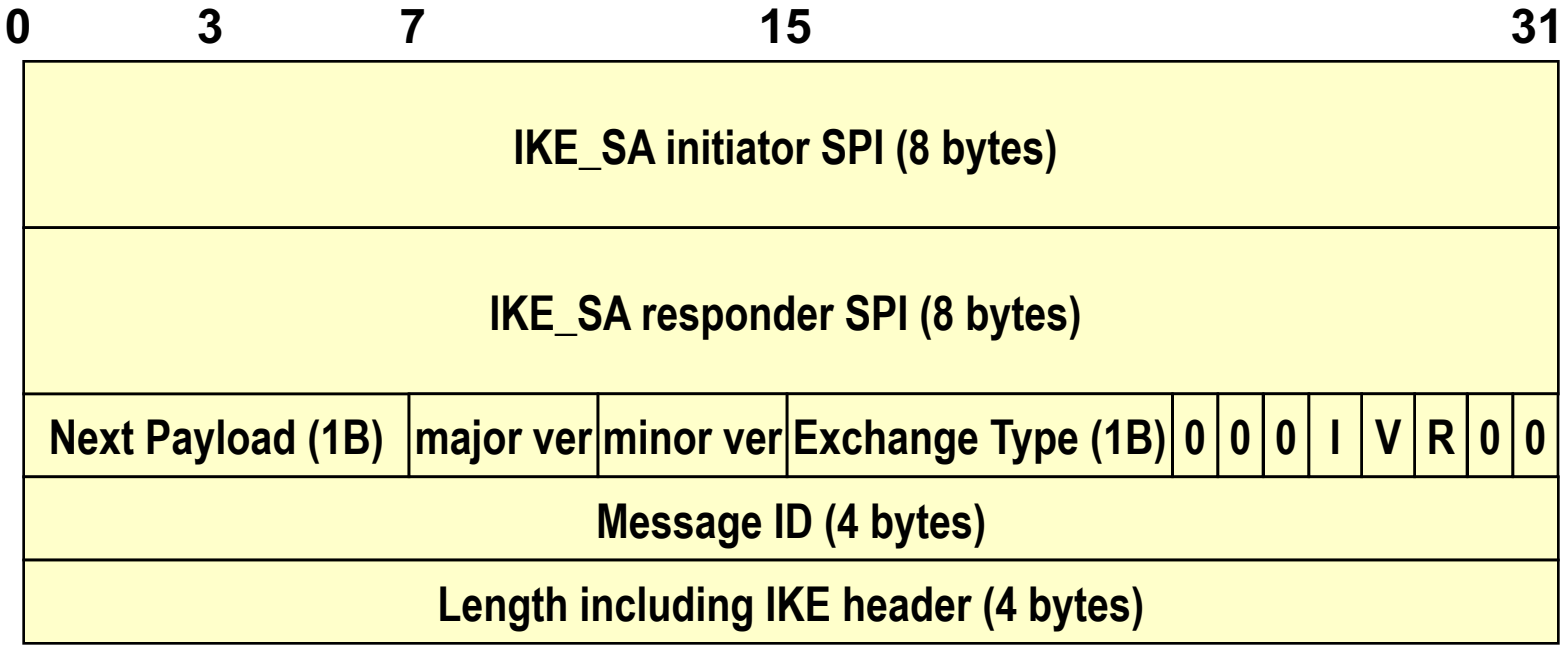
- **UDP encapsulated**
  - Ports 500 and/or 4500
  - Reliable delivery managed by IKE through retransmission
    - A new important feature of IKEv2!
    - No details here… see RFC 4306 for a thorough discussion
- **IKE header first**
- **Followed by one or more IKE payloads**
  - Brilliant idea (for a perhaps stretched analogy think to AVP concept; a more appropriate analogy is with the next header concept of IPv6)
    - flexible approach: new payloads added at later stages



IKE hdr, generic payload hdr



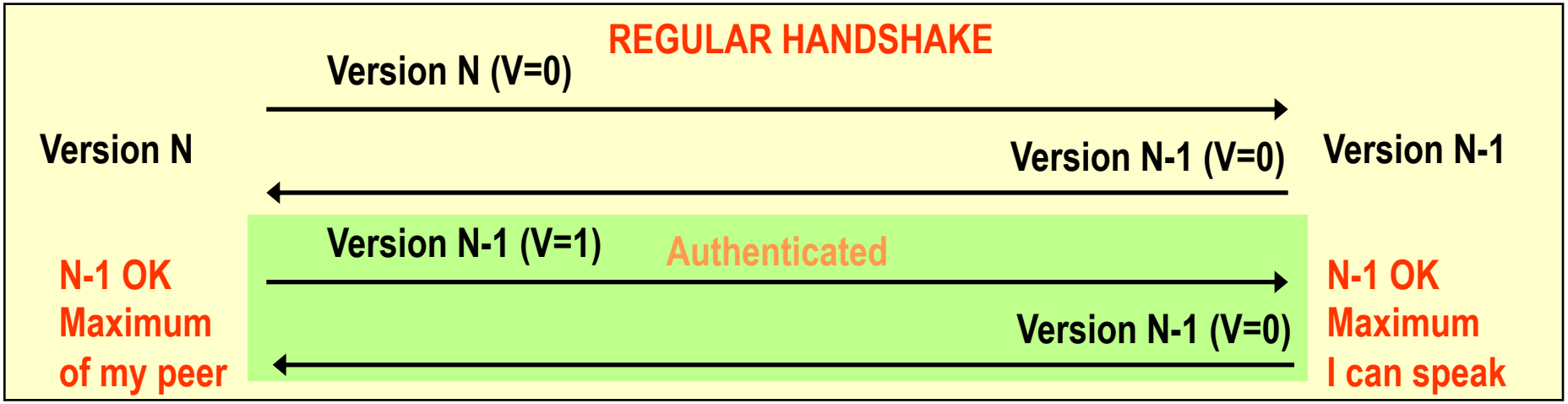
IKE header (explanation)

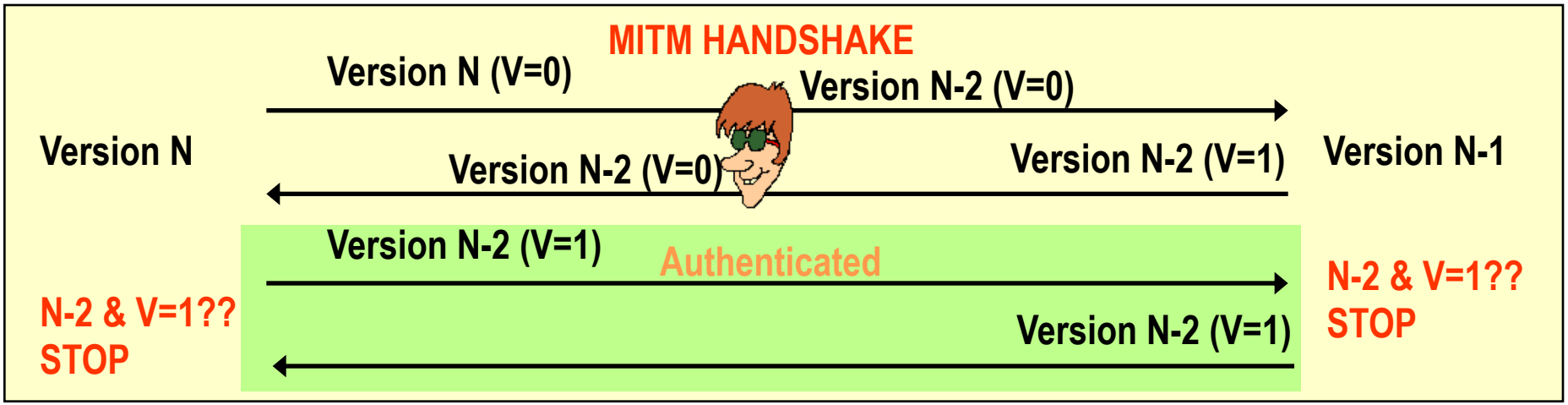


- **Initiator SPI, responder SPI:**
  - SA id for the IKE protocol. Initially, responder SPI set to 0; responder fills it in
    - don't confuse with the SPI for the CHILD AH/ESP SA
- **Versions: major = 2, minor = 0**
  - Bit V=1 says that implementation can "speak" higher version than what written in the hdr
- **Exchange type: one of 4 messages: IKE_SA_INIT, IKE_AUTH, CREATE_CHILD_SA, INFORMATIONAL**
  - Direction: bit R (response=1, request=0); bit I (original initiator=1, original responder=0; needed to properly match SPIs
- **Message ID: Sequence number counter, to match requests with responses, to manage retransmission, to combat replay attacks**

Version bit
Protection against version rollback

Different approach than SSL: based on V (version) flag



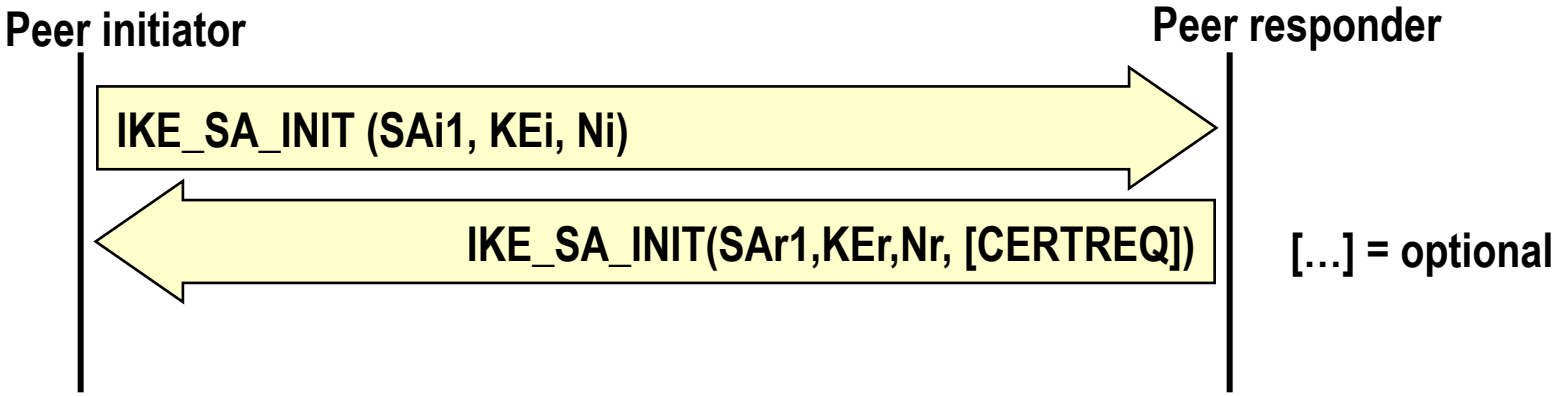


Too bad v1 does NOT include this flag!!! (Hence rollback to IKEv1 is not protected)

IKE_SA_INIT phase

- **Clear Text Request followed by Clear Text response**
  - Negotiates security parameters for the IKE_SA
  - sends nonces
  - sends Diffie-Hellman values
- **Output:**
  - Generate a session key from which all the other encryption and authentication keys will be generated
    - Called SKEYSEED

Exchanged Info



- **SAi1/SAr1 (Security Association) payloads:**
  - Initiator sends list of crypto algorithms supported
  - SAr1 choose one algo per each function (encryption, authentication, pseudo-random function)
- **KEi/KEr (Key Exchange) payloads:**
  - Diffie-Hellman Ephemeral values
- **Ni/Nr payloads:**
  - Random values used in the crypto parameters derivation, to protect replay
  - At least 16 bytes, up to 256 bytes
- **CERTREQ (optional payload)**
  - Request of a certificate

Security Association Payload example

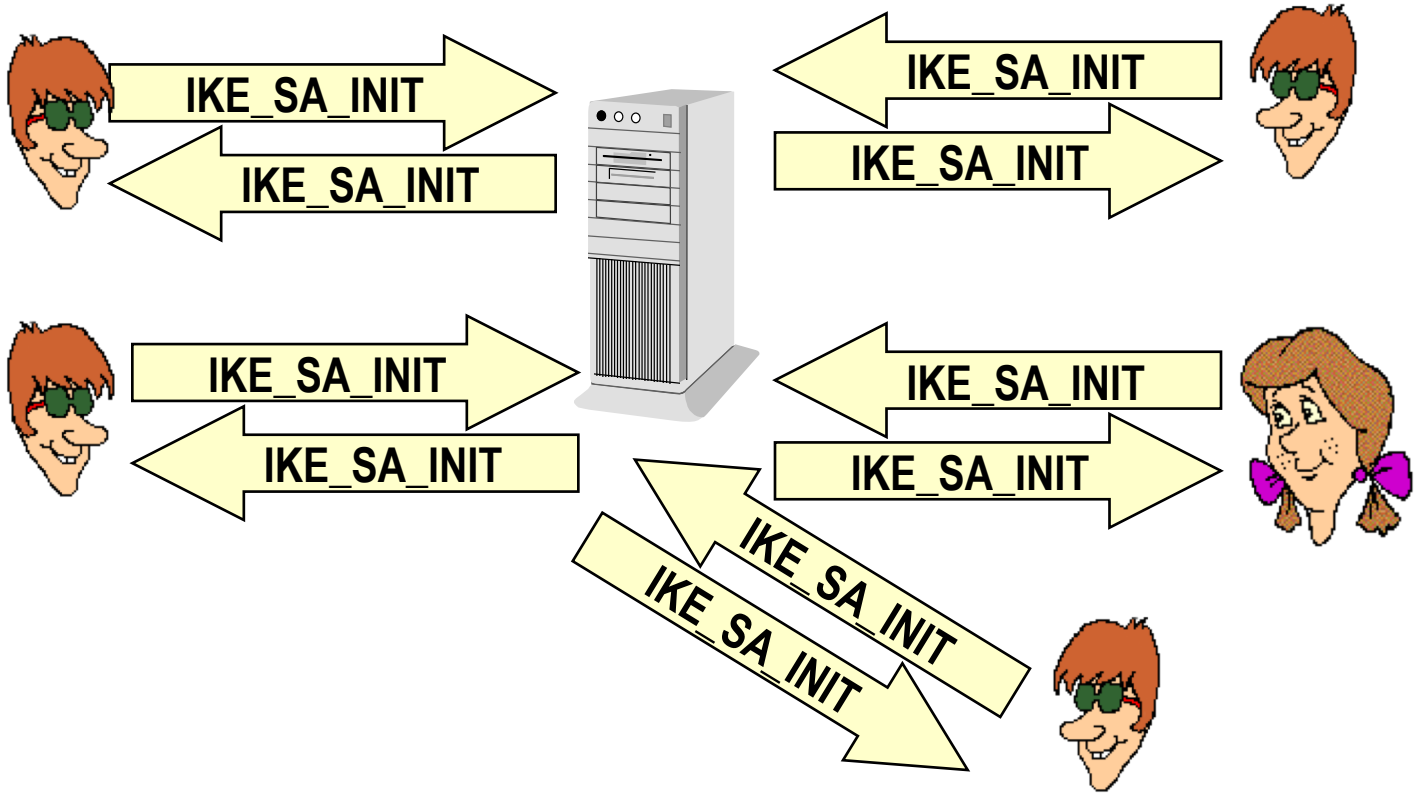
```
SA Payload
   |
   +--- Proposal #1 ( Proto ID = ESP(3), SPI size = 4,
   |      |             7 transforms,      SPI = 0x052357bb )
   |      |
   |      +-- Transform ENCR ( Name = ENCR_AES_CBC )
   |      |     +-- Attribute ( Key Length = 128 )
   |      |
   |      +-- Transform ENCR ( Name = ENCR_AES_CBC )
   |      |     +-- Attribute ( Key Length = 192 )
   |      |
   |      +-- Transform ENCR ( Name = ENCR_AES_CBC )
   |      |     +-- Attribute ( Key Length = 256 )
   |      |
   |      +-- Transform INTEG ( Name = AUTH_HMAC_SHA1_96 )
   |      +-- Transform INTEG ( Name = AUTH_AES_XCBC_96 )
   |      +-- Transform ESN ( Name = ESNs )
   |      +-- Transform ESN ( Name = No ESNs )
   |
   +--- Proposal #2 ( Proto ID = ESP(3), SPI size = 4,
          |             4 transforms,      SPI = 0x35a1d6f2 )
          |
          +-- Transform ENCR ( Name = AES-GCM with a 8 octet ICV )
          |     +-- Attribute ( Key Length = 128 )
          |
          +-- Transform ENCR ( Name = AES-GCM with a 8 octet ICV )
          |     +-- Attribute ( Key Length = 256 )
          |
          +-- Transform ESN ( Name = ESNs )
          +-- Transform ESN ( Name = No ESNs )
```

Key generation

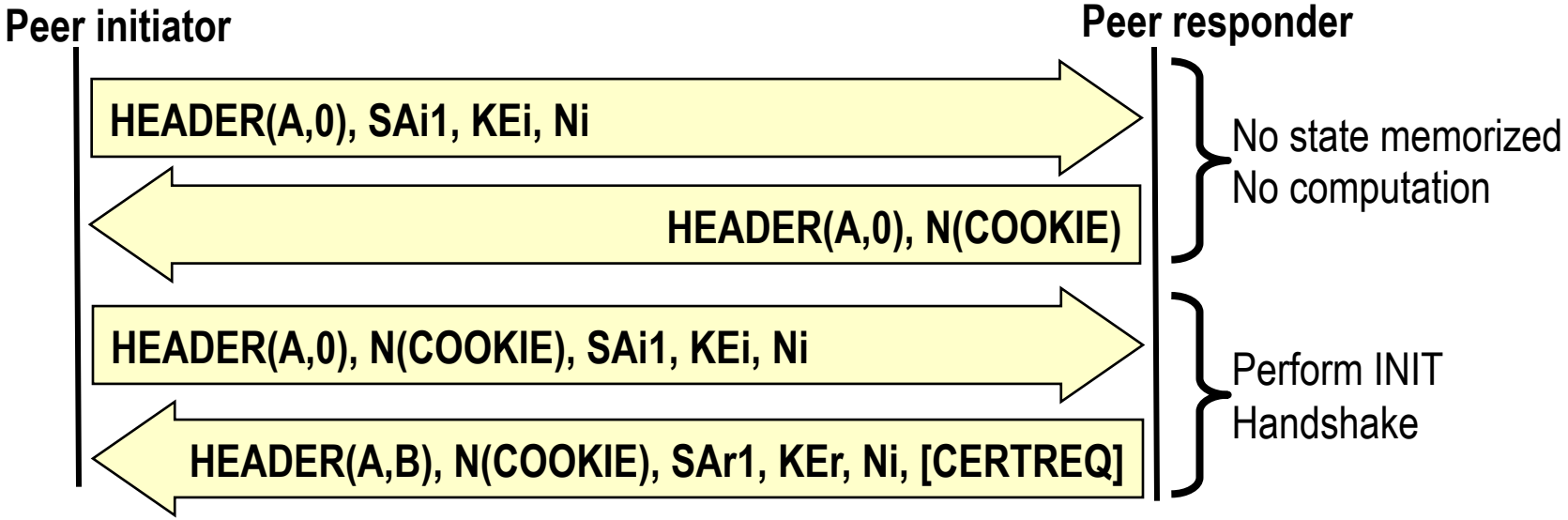
- **SKEYSEED:**
  - Contruction based upon a negotiated prf
    - Note the difference with TLS (where the PRF is explicitly given)
  - SKEYSEED = prf(Ni, Nr, g^ir)
    - g^ir = Diffie-Hellman shared key
- **7 keys generated:**
    - Through prf+ extended construction (similar to TLS)
    - prf+(SKEYSEED, Ni | Nr | SPIi | SPIr)
  - SK_{ai} SK_{ar} for authentication at initiator and responder
  - SK_{ei} SK_{er} for encryption at initiator and responder
  - SK_{pi} SK_{pr} for generating AUTH payload in SA_AUTH phase
  - SK_{d} to derive new keys for the CHILD_SA

Protection against DoS attacks

- **DH computation and key generation is a computational expensive process; state must be memorized**
- **Denial of Service Attack: spoof INIT requests (using forged IP addresses) and overload server (exhaust CPU and memory)**
  - ⇨ When load is sufficiently high, "Normal" requests cannot be processed anymore



Solution: cookie-based 4-way INIT handshake



➔ **Idea:**

- ⇨ Responder first replies with a cookie
- ⇨ Real initiators will reformulate the request including the cookie provided
- ⇨ Only at this point a state (SPI=B) is instantiated

➔ **Fools DoS attacks based on spoofed IPs**

- ⇨ Typical attacks!

Is this a real solution?

- **Q: What if DoS attacker spoofs cookies too**
  - E.g. using random cookies?
- **A: cookies must be "valid", i.e. issued by the responder**

…. but ….

- **Server must recognize valid cookies**
  - Hence it must store a state for the cookies, e.g. a database
  - And hence it must use memory!!

… is this true?

Idea: use stateless cookies

- **Use cookies which do not require any state memorization**
  - i.e. the validity of the cookie may be checked without any lookup at a database of issued cookies, but only looking at the request and at the cookie itself
- **Example:**

  Cookie = <VersionIDofSecret> | Hash(Ni | IPi | SPIi | <secret>)
- **where:**
  - Ni, IPi, SPIi is information that is available in the IKE_SA_INIT request (nonce, IP address, SPI)
    - Ni included to avoid that an attacker who sees only the server response may spoof the INIT+COOKIE message!
  - <secret> is an information available only at the responder
  - <secret> changes over time (hence VersionIDofSecret initial label)
    - Refresh to avoid build up of cookie dictionaries

IKE_AUTH phase

➔ **IKE message Encrypted and authenticated, except IKE header**

- ⇨ Using previously derived SKe as encryption key
- ⇨ Using previously derived SKa as authentication key
- ⇨ Using previously negotiated encryption/authentication algorithms

➔ **Sends AUTH payload**

- ⇨ Authenticates previous message + Identity of initiator and responder (IDi, IDr – could be IP addresses, domains, email addresses or else)
  - →Combats downgrade attacks
- ⇨ When certificates exchanged, authentication uses corresponding private key

CHILD_SA generation

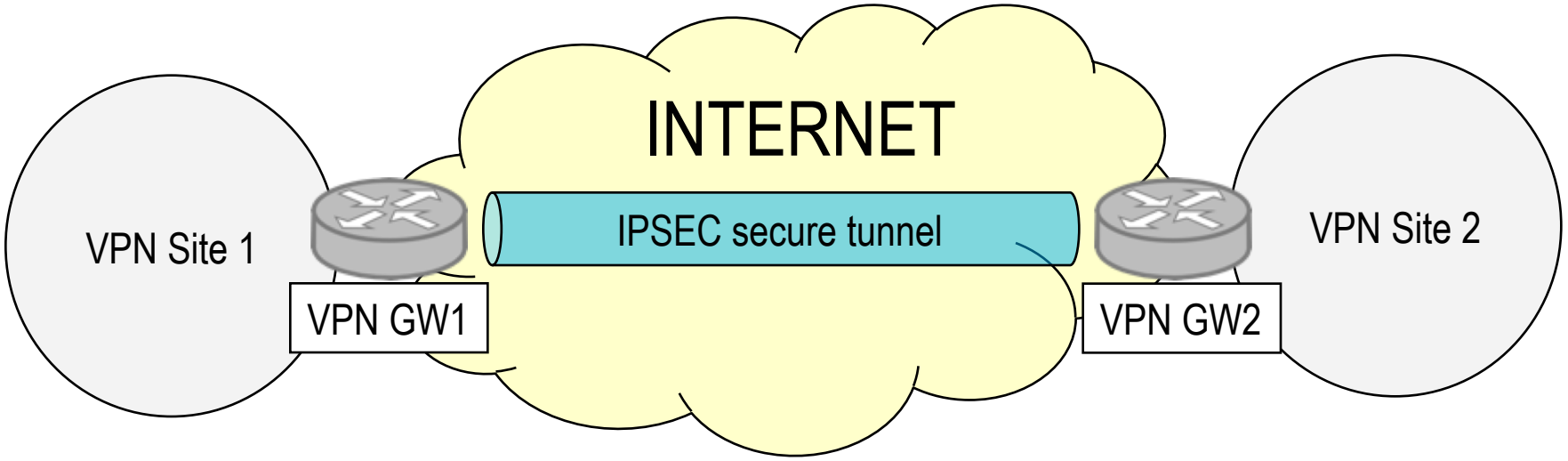
- **First CHILD_SA directly incorporated in AUTH messages**
  - Transmits new Security Association payloads to negotiate security parameters for the CHILD_SA
  - Transmits the Traffic Selector (TS) parameters, i.e. the IP addresses, port numbers & protocols to which the SA must be applied
    - E.g. IP addresses in the range 192.168.1.1-12
    - TS parameters included in the Security Policy Database
- **Other CHILD_SA may follow**
  - Further include new nonces

INFORMATIONAL

- **Notification messages**
- **Configuration messages**
  - E.g. assign internal IP address to remote terminal tunneled into an IPsec SA
- **Report error conditions**
- **Empty INFORMATIONAL payload = check for liveness**

IPSEC on Linux

Reference topology



GOAL: connect site 1 and site 2 through an IPSEC GW2GW VPN

IPSEC and Linux

- **IPSEC is natively supported since Linux Kernel 2.6.1**
- **The KAME project was successfully completed in 2006**
- **IPSEC data protection (ESP/AH) implemented in the kernel**
- **IPSEC user-space tools for**
  - ⇨SAD and SPD management (setkey)
  - ⇨Dynamic configuration with IKE (racoon)

IPSEC GW2GW VPN on Linux

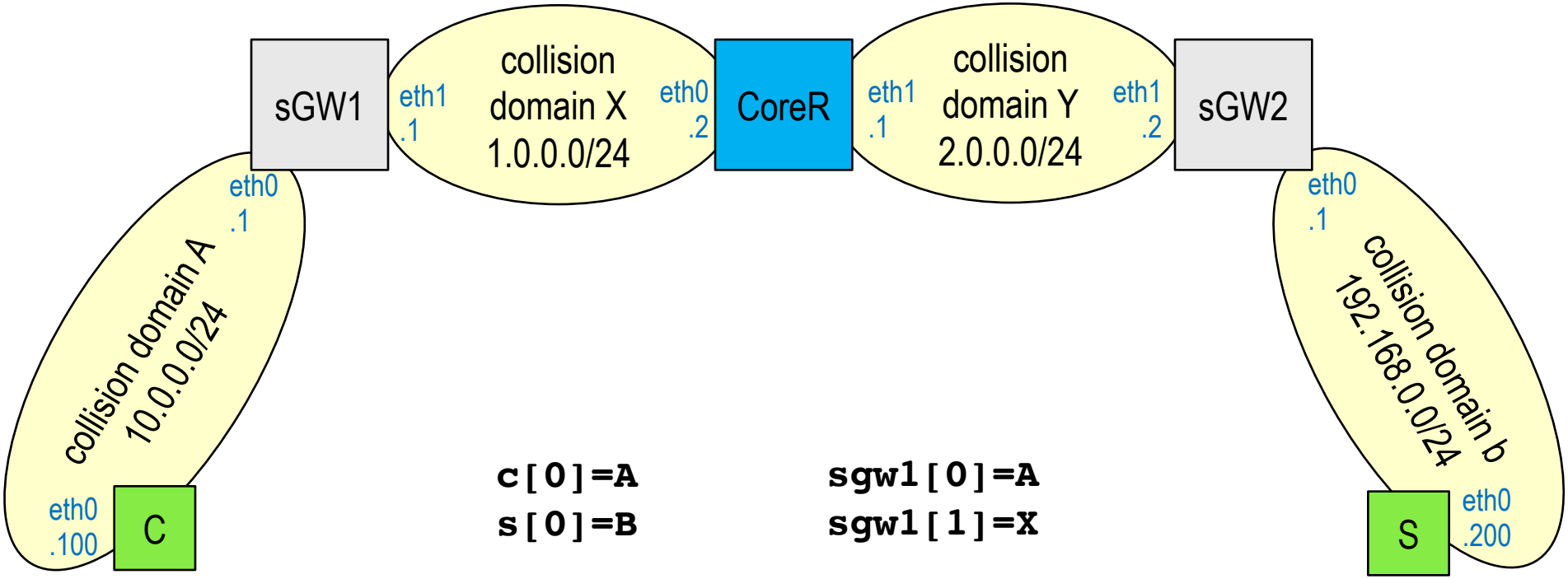
3 laboratories

1. Static SAs
2. Dynamic with pre-shared-key
3. Dynamic with public key authentication

Emulation with netkit (www.netkit.org)

Labs available at:
http//stud.netgroup.uniroma2.it/cgrl/ipsec/

Detailed NETKIT topology




```
c[0]=A          sgw1[0]=A
s[0]=B          sgw1[1]=X

corer[0]=X      sgw2[0]=B
corer[1]=Y      sgw2[1]=Y
```

No IPSEC

- ➔ **VPN site are unreachable from each other unless we set static DNAT rules on sGW1/2**
- ➔ **C and S can reach respectively 2.0.0.2 and 1.0.0.1 as dynamic NAT (MASQUERADE) is configured on sGW1/2**

sgw1.startup

```
ip link set eth0 up
ip link set eth1 up

ip addr add 10.0.0.1/24 dev eth0
ip addr add 1.0.0.1/24 dev eth1

ip r add 2.0.0.0/24 via 1.0.0.2

iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
```

IPSEC - static SAs

- ➔IPSEC SPD and SAD are manipulated with the “setkey” tool
- ➔2 secure policies (1 in, 1 out) are created to match all incoming/outgoing packets
- ➔SAs are bound to the public IP addresses of sGW1/2 (tunnel endpoints)
- ➔We configure 2 ESP SAs in tunnel mode with encryption and authentication
- ➔Obviously, the keys are in clear in the SA parameters

IPSEC - static SAs

sgw1.startup

```
ip link set eth0 up
ip link set eth1 up

ip addr add 10.0.0.1/24 dev eth0
ip addr add 1.0.0.1/24 dev eth1

ip r add 2.0.0.0/24 via 1.0.0.2
ip r add 192.168.0.0/24 via 1.0.0.2

iptables -t nat -A POSTROUTING -o eth1 -d ! 192.168.0.0/24 -j MASQUERADE

echo "configuring ipsec ..."
/bin/sh -c /root/ipsec.conf
echo "done"
```

NOTES:

1. MASQUERADE is disabled for packet addressed to VPN site
2. We need a route to VON SITE (in this case 1.0.0.2 as default GW would have worked)

IPSEC - static SAs

sgw1/root/ipsec.conf

```
# Flush the SPD and SAD
spdflush;
flush;

# Security Policies
spdadd 10.0.0.0/24 192.168.0.0/24 any -P out ipsec
       esp/tunnel/1.0.0.1-2.0.0.2/require;

spdadd 192.168.0.0/24 10.0.0.0/24 any -P in ipsec
       esp/tunnel/2.0.0.2-1.0.0.1/require;

# Security Associations
add 1.0.0.1 2.0.0.2 esp 0x201 -m tunnel
-E 3des-cbc 0x7aeaca3f87d060a12f4a4487d5a5c3355920fae69a96c831
-A hmac-md5 0xc0291ff014dccdd03874d9e8e4cdf3e6;

add 2.0.0.2 1.0.0.1 esp 0x301 -m tunnel
-E 3des-cbc 0xf6ddb555acfd9d77b03ea3843f2653255afe8eb5573965df
-A hmac-md5 0x96358c90783bbfa3d7b196ceabe0536b;
```

Note on key generation and verification

Key generation with dd

```
#128 Bit keys
#$ dd if=/dev/random count=16 bs=1| xxd -ps
#16+0 Records ein
#16+0 Records aus
#cd0456eff95c5529ea9e918043e19cbe

#192 Bit keys
#$ dd if=/dev/random count=24 bs=1| xxd -ps
#24+0 Records ein
#24+0 Records aus
#9d6c4a8275ab12fbfdcaf01f0ba9dcfb5f424c878e97f888
```

To verify the correct configuration just send a ping from C to S

IPSEC - IKE PSK

- ➔Same topology
- ➔In ipsec.conf remove the SA definition
- ➔SAs will be created dynamically by the "racoon" daemon as soon as a packet matching the security policy is found
- ➔PSKs are stored in a separate file with file mode 400 (otherwise we get an error)
- ➔racoon can be run with -F (no background – for debug)
- ➔We report only sGW1 configurations as sGW2 configuration is symmetric

IPSEC - IKE PSK

sgw1.startup

```
ip link set eth0 up
ip link set eth1 up

ip addr add 10.0.0.1/24 dev eth0
ip addr add 1.0.0.1/24 dev eth1

ip r add 2.0.0.0/24 via 1.0.0.2
ip r add 192.168.0.0/24 via 1.0.0.2

iptables -t nat -A POSTROUTING -o eth1
-d ! 192.168.0.0/24 -j MASQUERADE

echo "configuring ipsec and starting racoon”
/bin/sh -c /root/ipsec.conf
chmod 400 /root/psk.txt
racoon -f /root/racoon.conf
echo "done"
```

IPSEC - IKE PSK

sgw1/root/ipsec.conf

```
# Flush the SPD and SAD
spdflush;
flush;

# Security Policies
spdadd 10.0.0.0/24 192.168.0.0/24 any -P out ipsec
       esp/tunnel/1.0.0.1-2.0.0.2/require;

spdadd 192.168.0.0/24 10.0.0.0/24 any -P in ipsec
       esp/tunnel/2.0.0.2-1.0.0.1/require;
```

IPSEC - IKE PSK

sgw1/root/racoon.conf

```
path pre_shared_key "/root/psk.txt";

remote 2.0.0.2 {
        exchange_mode main;
        proposal {
                encryption_algorithm 3des;
                hash_algorithm md5;
                authentication_method pre_shared_key;
                dh_group modp1024;}
}

sainfo address 10.0.0.0/24 any address 192.168.0.0/24 any {
        pfs_group modp768;
        encryption_algorithm 3des;
        authentication_algorithm hmac_md5;
        compression_algorithm deflate;
}
```

sgw1/root/psk.txt

```
2.0.0.2         abcdefghilmnopq1234
```

IPSEC - IKE with X.509 certs

- **IKE (and also racoon) supports public key authentication with X.509 certificates**
- **No pre-shared keys are configured**
- **Authentication is based on public key cryptography, as in TLS**
  - ⇨In particular, in this lab, RSA signature verification
- **We need to create a CA and generate public/private key pairs and certificates**
  - ⇨Certificates (GWs and CA + GS keys) go into the sGW
  - ⇨CA key SHOULD be kept offline (some may say MUST)
- **We report only sGW1 configurations as sGW2 configuration is symmetric**

IPSEC - IKE with X.509 certs

sgw1/root/racoon.conf

```
path certificate "/root";

remote 2.0.0.2 {
        exchange_mode main;
        certificate_type x509 "sgw1.crt" "sgw1.key";
        verify_cert on;
        my_identifier asn1dn;
        ca_type x509 "ca.crt";

        proposal {
                encryption_algorithm 3des;
                hash_algorithm md5;
                authentication_method rsasig;
                dh_group modp1024;}
}

sainfo address 10.0.0.0/24 any address 192.168.0.0/24 any {
        pfs_group modp768;
        encryption_algorithm 3des;
        authentication_algorithm hmac_md5;
        compression_algorithm deflate;
}
```

Certificate generation

Same procedure as in the previous PKI lab

CA key generation

```
$ openssl genrsa -out ca.key 2048
```

CA self signed certificate generation

```
$ openssl req -new -x509 -days 3650 -key ca.key -out ca.crt
```

sGW1 key generation

```
$ openssl genrsa -out sgw1.key 2048
```

sGW1 CSR generation

```
$ openssl req -new -key sgw1.key -out sgw1.csr
```

sGW1 certificate signing

```
$ openssl x509 -req -in sgw1.csr -out sgw1.crt -sha1 -CA ca.crt -
CAkey ca.key -days 3650 -set_serial 1
```

Certificate generation (cont.d)

sGW2 key generation

```
$ openssl genrsa -out sgw2.key 2048
```

sGW1 CSR generation

```
$ openssl req -new -key sgw2.key -out sgw2.csr
```

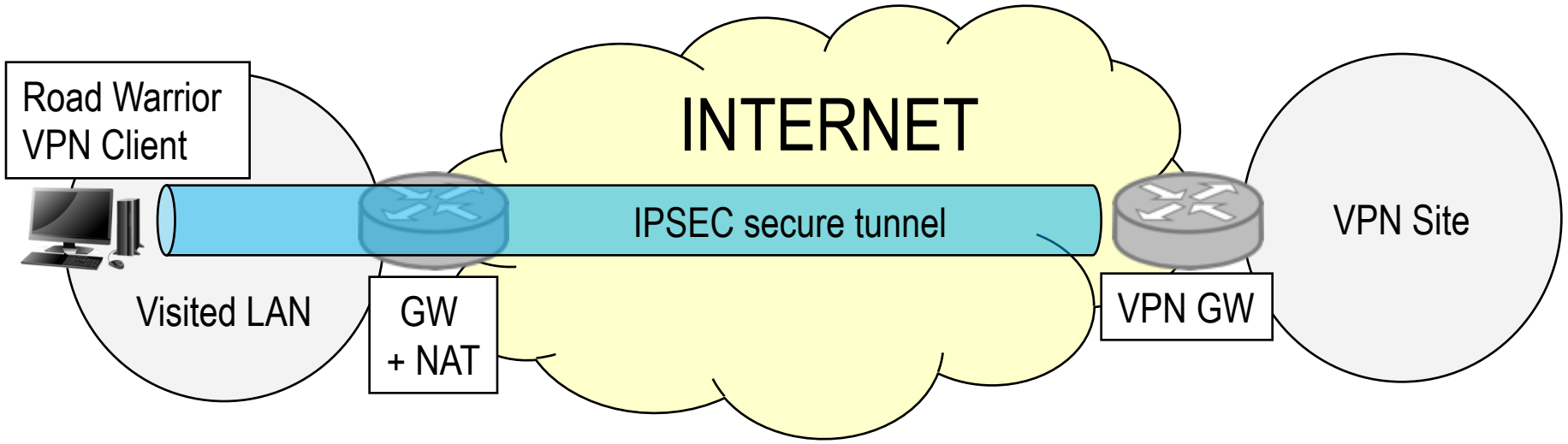
sGW1 certificate signing

```
$ openssl x509 -req -in sgw2.csr -out sgw2.crt -sha1 -CA ca.crt -
CAkey ca.key -days 3650 -set_serial 2
```

NOTE:

1) **we could have use easy CA utilities like easy_rsa or openssl-ca ☺**
2) **sgw1.crt, sgw1.key, ca.crt go into sgw1/root/**
3) **sgw2.crt, sgw2.key, ca.crt go into sgw1/root/**

Road warrior scenario

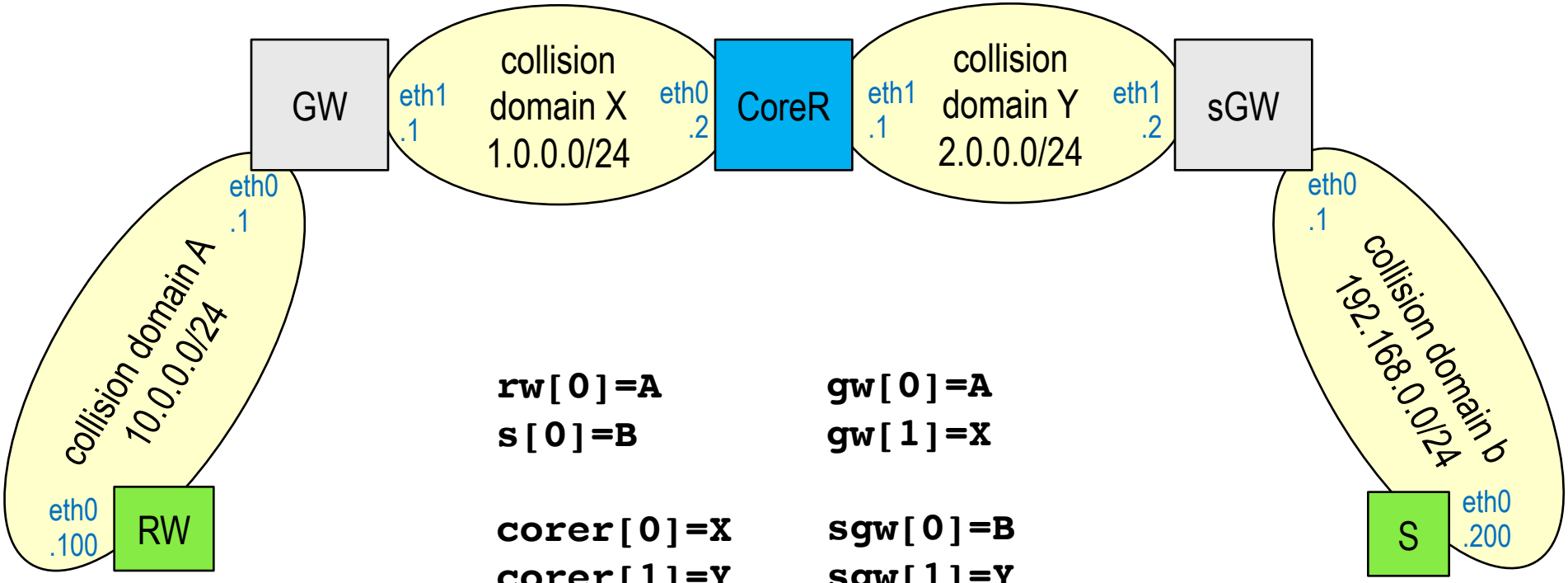


GOAL: connect site VPN client and VPN site through an IPSEC

CHALLENGES:

1. **Dynamic and (usually) private client IP**
2. **NAT MASDQERADE doesn't work with ESP tunnels (more later..)**

Same NETKIT topology
- different VM names -




```
rw[0]=A          gw[0]=A
s[0]=B           gw[1]=X

corer[0]=X       sgw[0]=B
corer[1]=Y       sgw[1]=Y
```

IPSEC - IKE for road warrior clients (X.509 auth)

- **everything works as in IKE GW2GW VPN except the following:**
  - SPs on the sGW must be created “on-demand” as IP addresses are unknown (until they are seen)
  - SPs on the client are bound to the private and dynamic IP address
  - IKE policies on the GW are bound to “unknown” peers
  - Since encapsulation is not ”NAT friendly”, we MUST ENFORCE NAT-T techniques
  - We create a new (key, cert) pair for the client

RW key generation

```
$ openssl genrsa -out rw2.key 2048
```

RW CSR generation

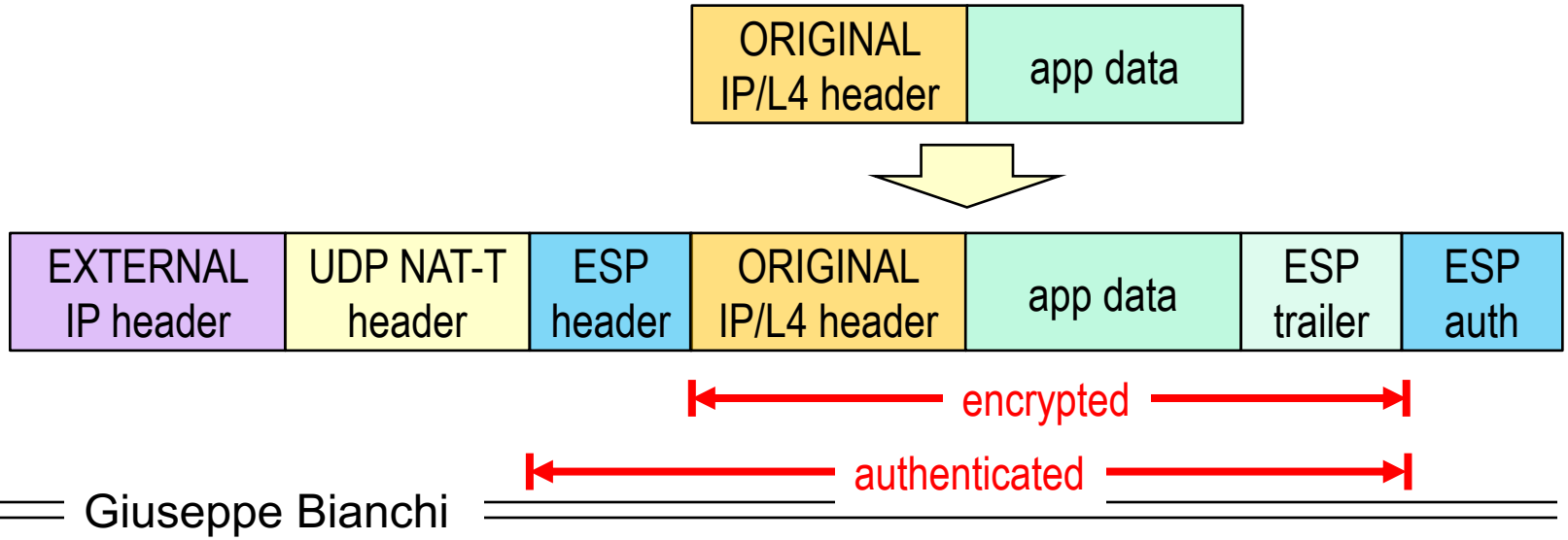
```
$ openssl req -new -key rw2.key -out rw.csr
```

RW certificate signing

```
$ openssl x509 -req -i rw.csr -out rw.crt -sha1 -CA ca.crt -CAkey
ca.key -days 3650 -set_serial 3
```

More on IPSEC NAT traversal

- **IP masquerade (NAPT) works well for TCP and UDP**
- **Linux (as other NAT implementations) does not have an ESP conn_track helper (TCP/UDP ports are encrypted)**
- **Only one masqueraded client can connect to a remote IPsec VPN server at any time**
- **IPSEC NAT traversal (NAT-T, RFC 3947 "Negotiation of NAT-Traversal in the IKE")**
  - It detects if NAT-T is supported by both ends
  - If detect if there is a NAT
  - Enforce ESP in IP/UDP encapsulation (as in RFC394 "UDP Encapsulation of IPsec ESP Packets")



IPSEC - IKE for road warriors

sgw/root/racoon.conf

```
path certificate "/root";
remote anonymous {
        exchange_mode main;
        nat_traversal on;
        generate_policy on;
        passive on;
        certificate_type x509 "sgw2.crt" "sgw2.key";
        verify_cert on;
        my_identifier asn1dn;
        ca_type x509 "ca.crt";
        proposal {
                encryption_algorithm 3des;
                hash_algorithm md5;
                authentication_method rsasig;
                dh_group modp1024;}
}

sainfo anonymous {
        pfs_group modp768;
        encryption_algorithm 3des;
        authentication_algorithm hmac_md5;
        compression_algorithm deflate;
}
```

sGW has no SP configuration with setkey!!!

IPSEC - IKE for road warriors

rw/root/racoon.conf

```
path certificate "/root";
remote 2.0.0.2 {
        nat_traversal on;
        exchange_mode main;
        certificate_type x509 "rw.crt" "rw.key";
        verify_cert on;
        my_identifier asn1dn;
        ca_type x509 "ca.crt";
        proposal {
                encryption_algorithm 3des;
                hash_algorithm md5;
                authentication_method rsasig;
                dh_group modp1024;}
}

sainfo address 10.0.0.0/24 any address 192.168.0.0/24 any {
        pfs_group modp768;
        encryption_algorithm 3des;
        authentication_algorithm hmac_md5;
        compression_algorithm deflate;
}
```

IPSEC - IKE for road warriors

rw/root/ipsec.conf

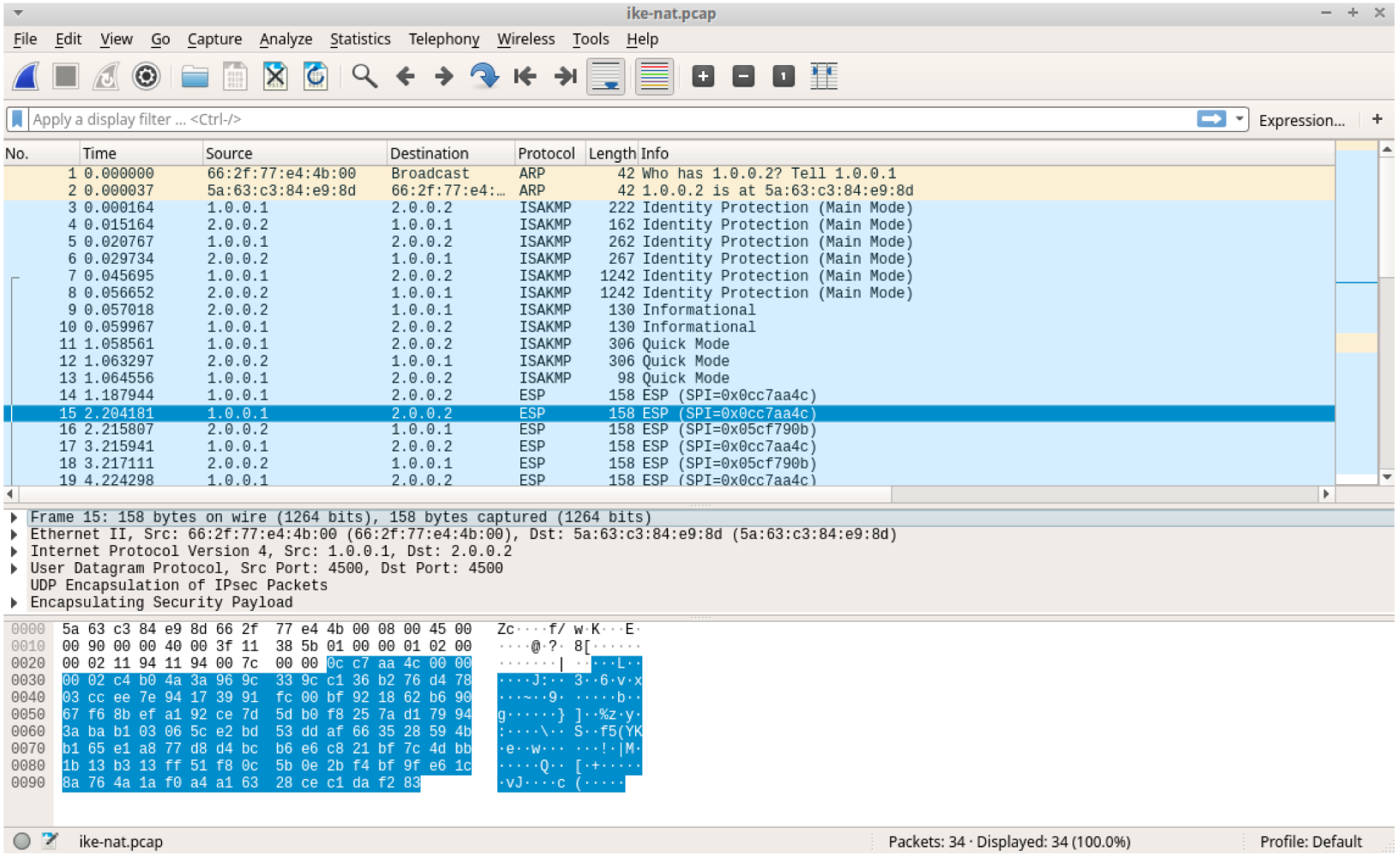
```
#! /usr/sbin/setkey -f

# Flush the SPD and SAD
spdflush;
flush;

# Security Policies
spdadd 10.0.0.0/24 192.168.0.0/24 any
       -P out ipsec
       esp/tunnel/10.0.0.100-2.0.0.2/require;

spdadd 192.168.0.0/24 10.0.0.0/24 any
       -P in ipsec
       esp/tunnel/2.0.0.2-10.0.0.100/require;
```

DEMO:

1) Run "racoon -f /root/racoon.conf" on sgw
2) Run "sh -c /root/ipsec.conf && racoon -f /root/racoon.conf" on rw
3) From rw ping 192.168.0.200 (S)

!!! NOTE: 1 ESP flows works also without NAT-T !!!

Packet trace on CoreR



NOTE the UDP encapsulated ESP packets (clear packets are ICMP echo request/replies)