LUND UNIVERSITY

EITA25 Computer Security (Datasäkerhet)
Internet Security Protocols

PAUL STANKOVSKI WAGNER, EIT, 2020-02-10

Internet Security Protocols

Network Attacks and Threats

- **Passive Attacker –** Can only listen to traffic
- **Active Attacker –** Can modify, delete and insert messages

Examples of attacks

- **Traffic analysis** – Communication patterns can be found. Who is talking to who and how often?
- **Man-in-the-middle attack –** Intercept and forward modified traffic (often using independent connections)
- **Replay attacks** – Unauthorized data can be retransmitted.
- **Spoofing attacks** – Disguise as legitimate sender

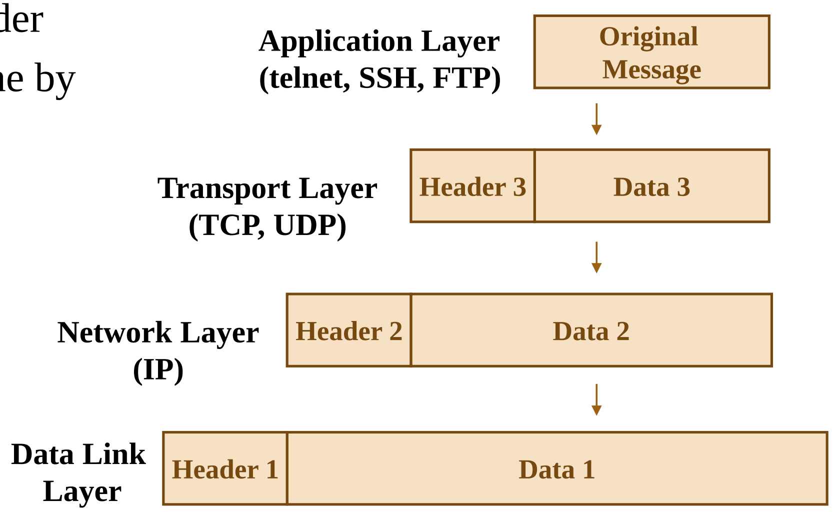
Services Needed

- **Data integrity** – The contents of a packet can otherwise be accidentally or deliberately modified.
- **Data confidentiality** – Sensitive data can otherwise be read by an eavesdropper
- **Data origin authentication** – The origin of an IP packet can otherwise be forged *(identity spoofing)*

LUND UNIVERSITY

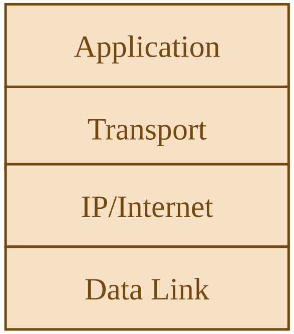
TCP/IP, Layered Model

- TCP/IP model has four layers
- Each layer adds new header
- Headers are peeled off one by one at destination



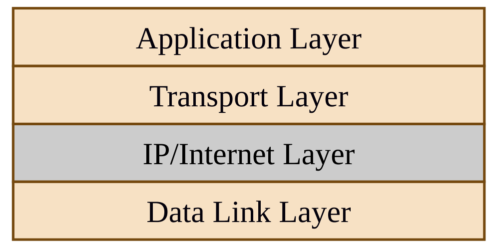
Security at Different Layers

- Application layer (e.g., PGP, Kerberos, SSH, etc.)
  - Security can meet the exact demands of the application
  - Has to be designed for each application
- Transport layer (e.g., SSL/TLS)
  - Application developer can choose if it is to be used or not
  - Existing applications have to be modified
- Internet layer (e.g., IPsec)
  - Seamless security for applications
  - More difficult to exercise on a per user basis in multiuser system, or per application basis
- Data link layer (e.g., hardware encryption)
  - Very fast
  - Need dedicated links



IPsec (Internet Protocol Security)

- IPsec provides security at network (Internet) layer.
  - All IP datagrams covered
  - No re-engineering of applications
  - Transparent to users
- Mandatory for IPv6
  - Extension headers defined in the protocol
- Optional for IPv4
- Two major security mechanisms:
  - AH: Authentication Header
  - ESP: Encapsulating Security Payload
- Two options
  - Transport mode
  - Tunnel mode



Security Associations

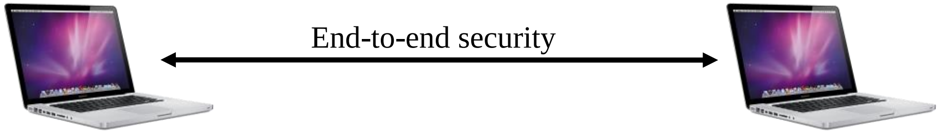
- An SA is identified by a Security Parameters Index (SPI) and includes e.g.
  - Sequence number counter
  - Algorithms, keys and additional parameters for AH or ESP
  - Protocol mode (tunnel or transport)
- Different for each combination of

{ESP, AH} x {Transport, Tunnel} x {Sender, Receiver}

- Possible to combine SAs
  - *Transport Adjacency* – Several SAs used on same IP datagram in transport mode
  - *Iterated Tunneling* – Several nested tunnels

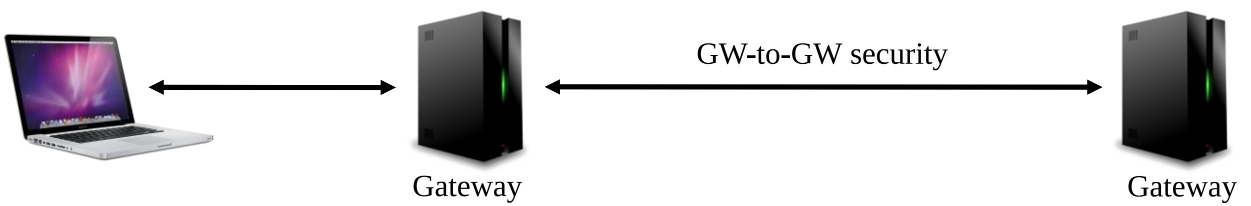
IPsec Transport Mode

- Protection for upper-layer protocols.
- Protection covers IP datagram payload (and selected header fields).
  - Could be TCP packet, UDP, ICMP message,….
- Host-to-host (end-to-end) security:
  - IPsec processing performed at endpoints of secure channel
  - So endpoint hosts must be IPsec-aware

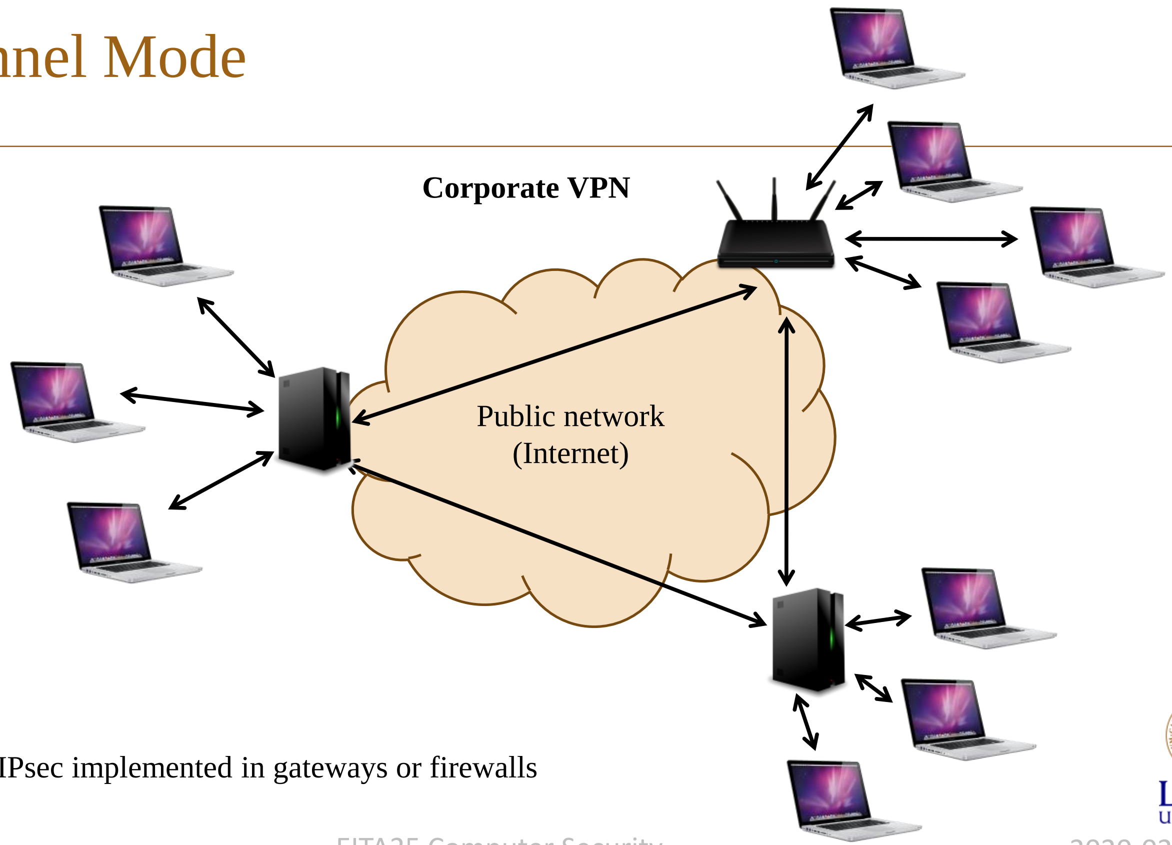


IPsec Tunnel Mode

- Protection for *entire* IP datagram
  - Entire datagram plus security fields treated as new payload of 'outer' IP datagram.
  - Original IP datagram encapsulated within an outer IP datagram.
- IPsec processing performed at security gateways on behalf of endpoint hosts
  - Gateway could be perimeter firewall or router.
  - Gateway-to-gateway rather than end-to-end security.
  - Hosts need not be IPsec-aware.
- Intermediate routers have no visibility of inner IP datagram when encrypted
  - Even original source and destination addresses encapsulated and hence 'hidden'.



IPsec Tunnel Mode



IPsec implemented in gateways or firewalls

AH Protocol

- AH = Authentication Header (RFC 4302)
- Provides ***data origin authentication*** and ***data integrity*** using a MAC
- **Purpose:** AH authenticates whole payload and most of header
- Prevents IP address spoofing since source IP is authenticated
- Prevents replay attack
  - AH sequence number is authenticated
  - New SA with new key when sequence number reaches max (2^{32}-1)
  - Replay protection must be implemented by receiver

The Authentication Header

- Header is added to original IP packet
- Fields in header include:
  - Next header
    - the type of payload data (TCP, UDP, ICMP, IGMP,…)
  - Payload length
    - Number of 32-bit words minus 2 (length of the authentication header - 2)
  - SPI (Security Parameters Index)
    - Identifies algorithms and keys
  - 32-bit sequence number
  - Integrity Check Value (MAC value)
    - Calculate over all fields except mutable IP header fields and ICV
    - Default 96 bits

The Authentication Header

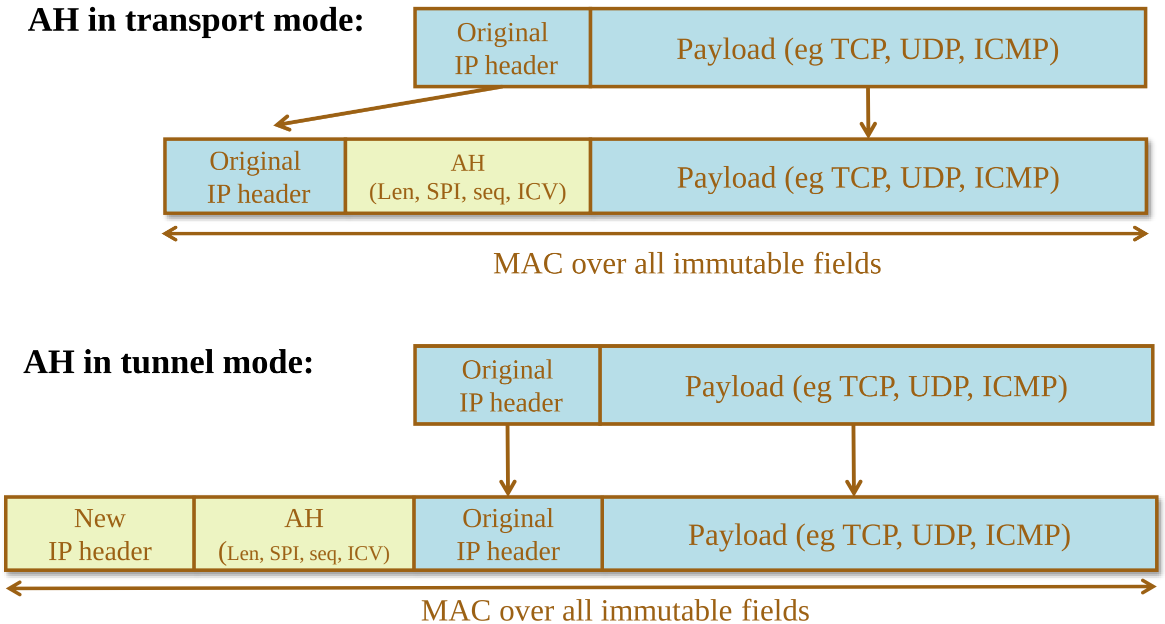
0	8	16	31
Next header	Payload length	RESERVED	
Security Parameter Index (SPI)			
Sequence Number			
Integrity Check Value (ICV)			

- Integrity Check Value (Authentication data) is of variable length (multiple of 32 bits)
  - Put all mutable fields in headers to zero before calculating checksum
    - E.g., TTL, flags and the MAC itself,

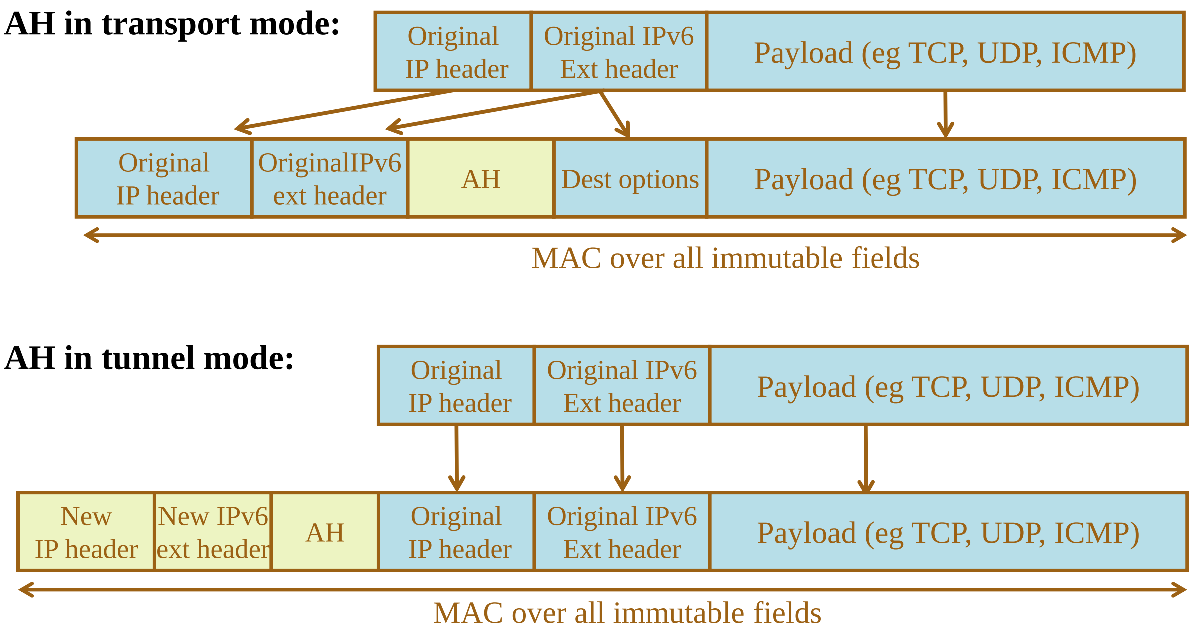
AH Protocol – Transport and Tunnel (IPv4)

AH in transport mode:

AH in tunnel mode:



AH Protocol – Transport and Tunnel (IPv6)



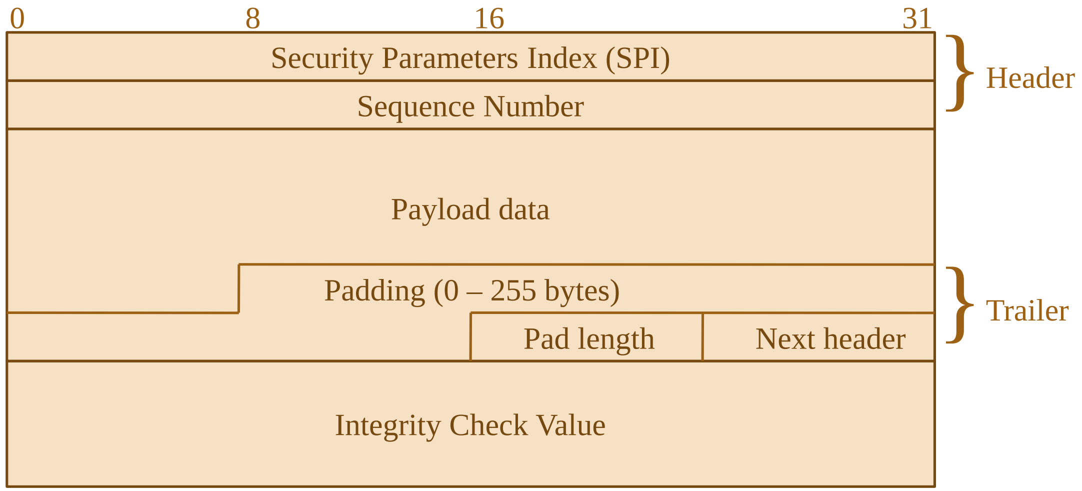
ESP Protocol

- ESP = Encapsulating Security Payload (RFC 4303 obsoletes RFC 2406).
- Provides one or both of:
  - confidentiality
  - authentication
- Uses symmetric encryption and MACs based on secret keys shared between endpoints
  - Key stored in SA

ESP Header and Trailer

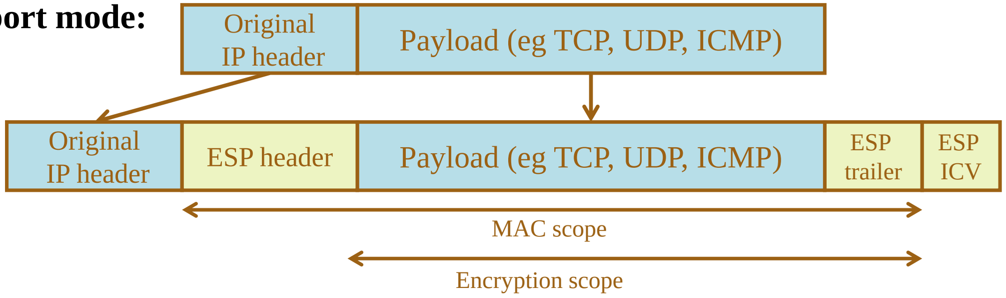
- ESP specifies a header and trailing fields to be added to IP datagrams.
- Fields in header include:
  - SPI = Security Parameters Index
    - Identifies algorithms and keys
  - 32-bit sequence number
- Fields in trailer include:
  - Any padding needed for encryption algorithm (may also help disguise payload length)
  - Padding length
  - Next header
- Integrity check value (Authentication data) if authentication is used
  - the MAC value

ESP Packet

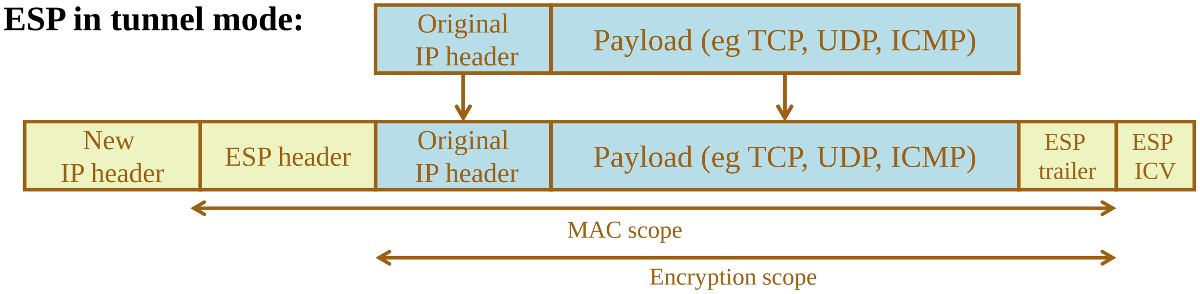

0
8
16
31
Security Parameters Index (SPI)
Sequence Number
Header
Payload data
Padding (0 – 255 bytes)
Pad length
Next header
Trailer
Integrity Check Value

ESP Protocol – Transport and Tunnel (IPv4)

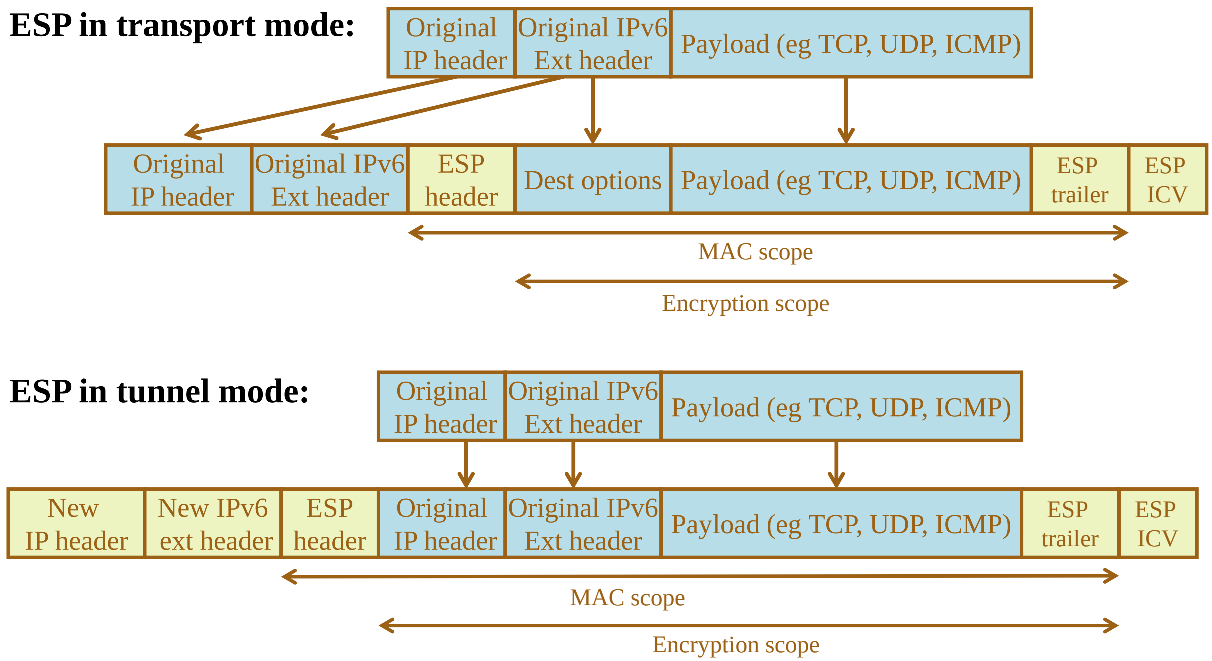
ESP in transport mode:



ESP in tunnel mode:



ESP Protocol – Transport and Tunnel (IPv6)



Algorithms in IPsec

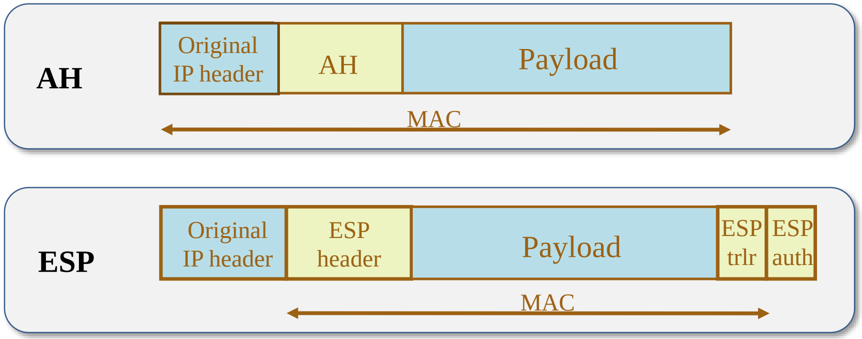
- Algorithms are not fixed
  - If an algorithm turns out to be weak we can pick another
- Still, there are mandatory algorithms
  - We must guarantee that different implementations can be used together

ESP Encryption		**ESP Authentication**		**AH**	
Req	*Algorithm*	*Req*	*Algorithm*	*Req*	*Algorithm*
MUST	NULL	MUST	HMAC-SHA1-96	MUST	HMAC-SHA1-96
MUST-	TripleDES-CBC	MUST	NULL	SHOULD+	AES-XCBC-MAC-96
SHOULD+	AES-CBC	SHOULD+	AES-XCBC-MAC-96	MAY	HMAC-MD5-96
SHOULD	AES-CTR	MAY	HMAC-MD5-96		
SHOULD NOT	DES-CBC				

LUND UNIVERSITY

Combining Security Associations

- Recall MAC in transport mode (IPv4)



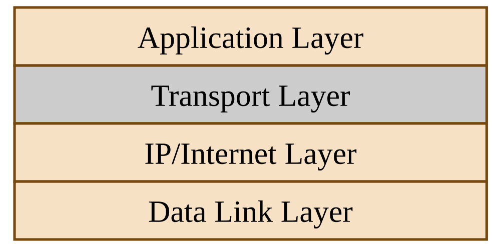
- Authentication in ESP does not cover original IP header
  - If this is needed AH can be added after ESP
  - Called *transport adjacency*
- Drawback: Two SAs are needed instead of one

Key Management

- Key negotiation can be
  - Manual – An administrator configures all communicating systems. Useful in small and static environments
  - Automatic – Automated system enabling on-demand creation of keys and SAs.
- Default automated key management protocol is ISAKMP/IKE
  - Internet Security Association and Key Management Protocol (ISAKMP) defines packet formats to establish, negotiate, modify and delete SAs, e.g., how to transfer certificates, how to exchange key material etc.
  - Internet Key Exchange protocol (IKE) defines how keys can be exchanged. It supports Digital signatures, public key encryption and pre-shared keys.
  - IKEv2 proposed in dec 2005.
- VPNs can use IPsec but sometimes the key exchange protocol is proprietary

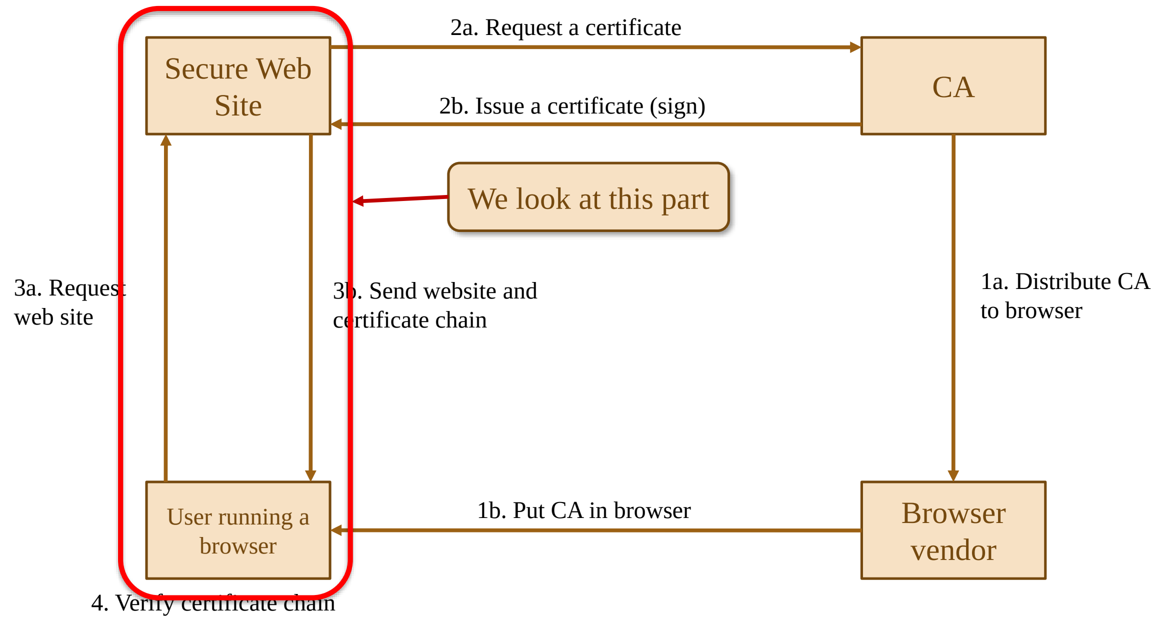
Transport Layer Security (TLS)

- TLS was previously called SSL (Secure Sockets Layer)
  - TLS 1.0: 1999 (RFC 2246)
  - TLS 1.1: 2002 (RFC 4346)
  - TLS 1.2: 2006 (RFC 5246)
  - SSL 2.0 prohibited 2011 (RFC 6176)
  - SSL 3.0 prohibited 2015 (RFC 7568)
  - TLS 1.3: Aug 2018



- TCP protocol: reliable byte stream between two nodes
  - Stateful connection-oriented protocol
  - Detects: lost packets, out of order packets, duplicates, etc.
- TCP lacks strong cryptographic entity authentication, data integrity or confidentiality
- Needs met by the TLS protocol
  - Invented by Netscape (as SSL)
  - Confidentiality
  - Message integrity

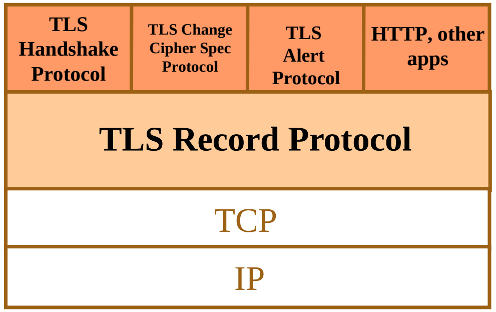
Recall Certificates in TLS



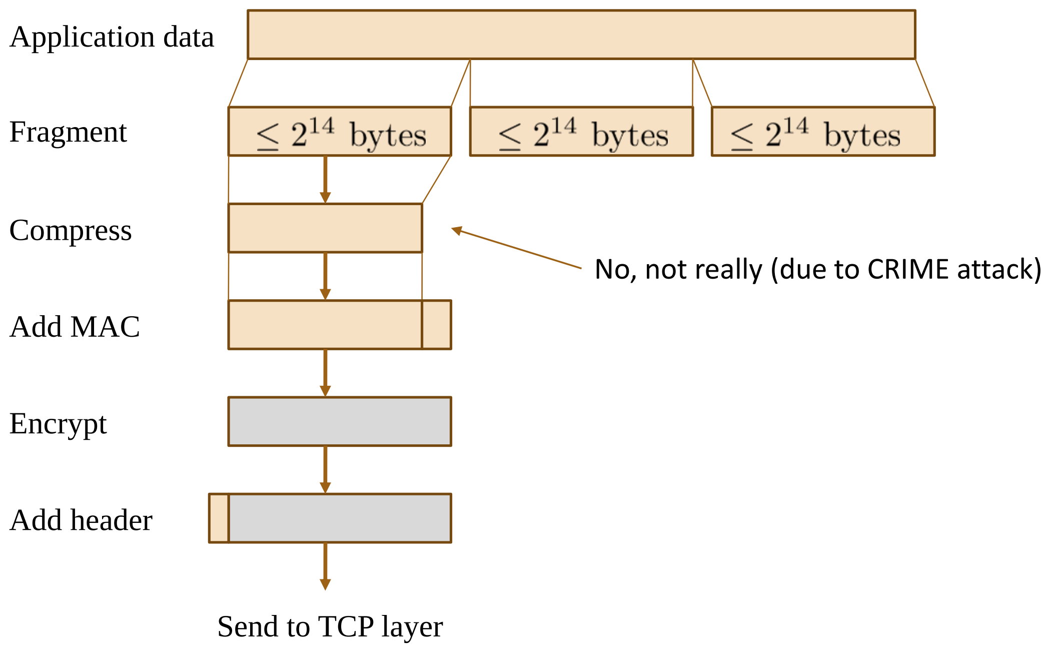
If verification in step 4 is valid, the server and client can set up a secure connection

TLS Protocol Stack

- SSL/TLS has two layers of protocols
  - **TLS Record Protocol**
    - » Provides confidentiality and message integrity
  - **TLS Handshake Protocol**
    - » Authenticate and negotiate keys
  - **TLS Change Cipher Spec Protocol**
    - » One byte message that updates the cipher suite
  - **TLS Alert Protocol**
    - » Used to send warning and error messages e.g., bad_record_mac and bad_certificate
  - Other applications that use the record protocol

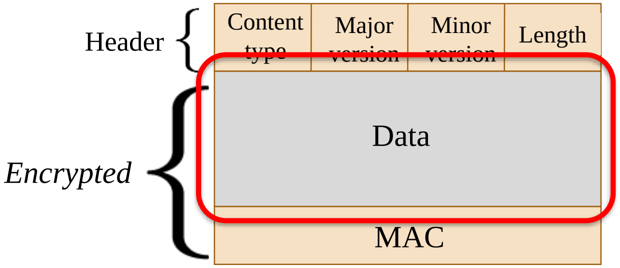


TLS Operation



TLS Record Protocol

- Adds compression (optional)
- Computes a MAC for the packet using HMAC
- Encrypts the packet using the negotiated cipher, e.g., AES, IDEA, DES, 3DES, RC4, Authenticated encryption modes
  - RC4 was prohibited 2015
- Content type defines upper protocol (8 bits)
  - *Change Cipher Spec: 20*
  - *Alert: 21*
  - *Handshake: 22*
  - *Application data: 23*
- Version defined as (8+8 bits)
  - *SSL: 3 and 0*
  - *TLS: 3 and {1,2,3}*
- Length of Data field (16 bits)



Upper Layer Protocols

- Information seen as data in the record protocol

1

1 byte

Change Cipher Spec Protocol

Level	Alert

1 + 1 byte

Level can be ”warning” or ”fatal”

Alert Protocol

Type	Length	Content

$1 + 3 + \geq 0$ bytes

Handshake Protocol

Type defines what type of message it is

Application specific data

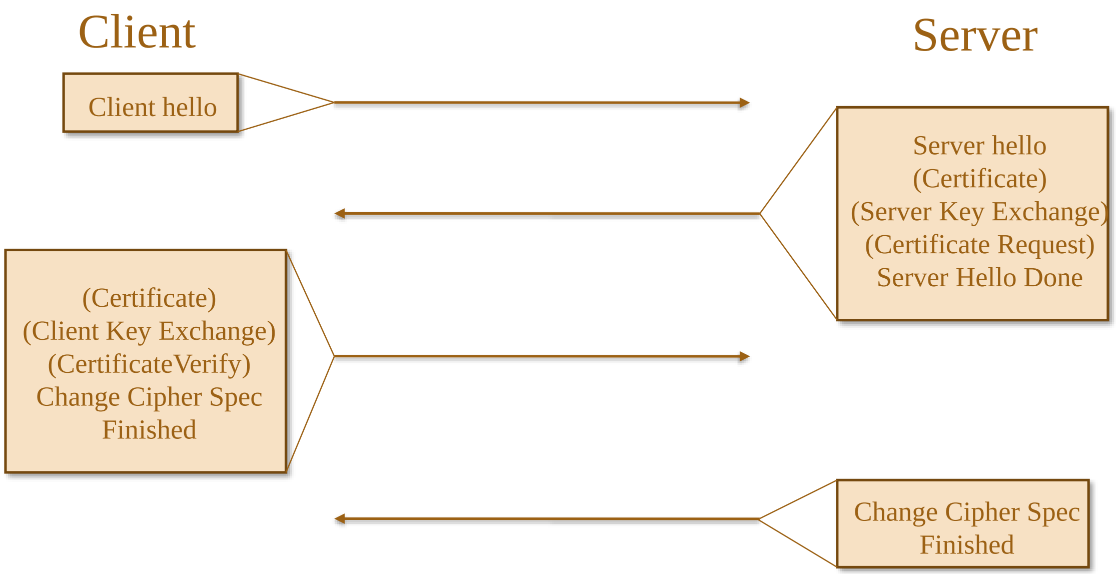
≥ 1 byte

Other protocols, e.g., HTTP

TLS Handshake Protocol (TLS $\leq$ 1.2)

- Purpose of handshake
  1. Authenticate server to client
  2. Establish which algorithms to use
  3. Negotiate keys for encryption and MAC
  4. Authenticate client to server (optional)
- 10 different message types
- Which types are used and what they look like will depend on mainly two things
  - Key exchange method
  - If server authenticates client

TLS Handshake Overview



In total 9 message types! Number 10 is ”hello request” sent from server to inform client that it should start a new negotiation.

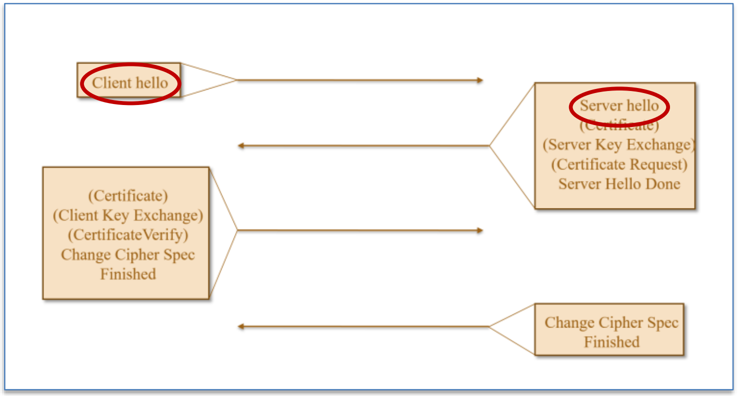
Key Exchange Methods

- **Basic problem:** Server and client must agree on a secret value
  - We call this a ”premaster secret”

- RSA – Client generates ”premaster secret” and uses RSA to encrypt it with public key of server
  - Certificate needed
  - (Removed in TLS 1.3)
- Ephemeral Diffie-Hellman – The premaster secret is negotiated with Diffie-Hellman and values are signed with private key
  - Certificate needed
- Fixed Diffie-Hellman – Diffie-Hellman values (public parameters) are stored in a certificate
  - Certificate needed
- Anonymous Diffie-Hellman – unauthenticated Diffie-Hellman key exchange
  - No certificate needed
  - Vulnerable to Man-In-The-Middle attacks

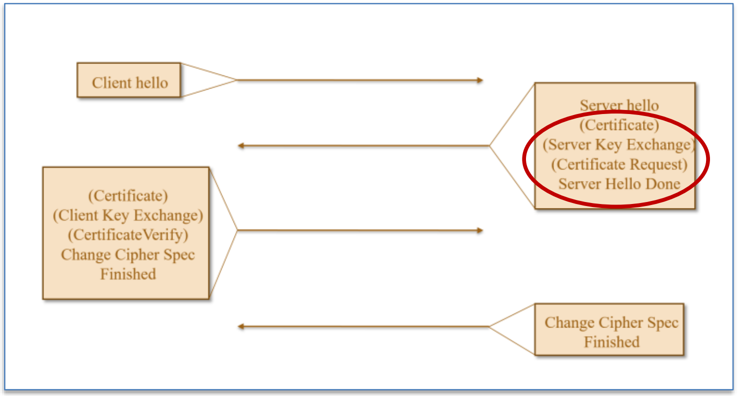
A Closer Look at Messages

- We first look at messages when RSA is used
- Client Hello
  - ClientRandom – 28 bytes used when calculating master secret
  - Suggested cipher suites – Suites implemented on client side e.g., TLS_RSA_WITH_AES_256_CBC_SHA
  - Suggested compression algorithms – compression algorithms implemented by client
- Server Hello
  - ServerRandom – 28 bytes used when calculating master secret
  - Decided cipher suite to use – Server picks a suite that is implemented on both client and server
  - Decided compression to use



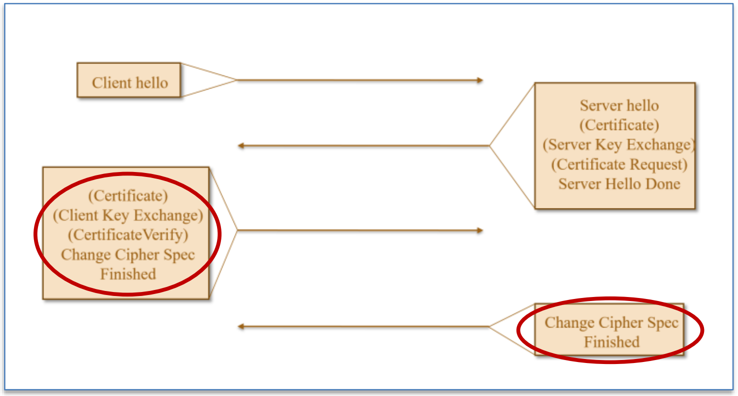
A Closer Look at Messages

- Certificate (server) – Server sends his certificate (chain) to client
- Server Key Exchange – Not used for RSA
- Certificate Request – Sent if server wants the client to authenticate itself
- Server Hello Done – Indicates that the server hello is done



A Closer Look at Messages

- Certificate (client) – sent if server has requested a certificate
- Client Key Exchange – Client generates a *pre-master secret* and encrypts this with the public key of the server. Used later to compute master secret.
- Certificate verify – A signed hash based on the preceeding messages. Used to verify that the client has the private key. Misuse of certificates impossible.
- Change Cipher Spec – After this message the client starts using the new algorithms and keys
- Finished – Contains the encrypted hash of previous messages
- Change Cipher Spec – After this message the server starts using the new algorithms and keys.
- Finished – Contains the encrypted hash of previous messages



Diffie-Hellman Key Exchange

- If Diffie-Hellman is used some messages will look different
- Certificate (Server) – If *Anonymous Diffie-Hellman* is used no certificate is sent
- Server Key Exchange – If *Anonymous* or *Ephemeral Diffie-Hellman* is used, the parameters are sent here (p, g and $g^x \bmod p$)
  - For *Ephemeral Diffie-Hellman* the values are signed
  - For *Anonymous Diffie-Hellman* the values are not signed
- Certificate (Client) – For *Fixed Diffie-Hellman*, parameters in certificate
- Client Key Exchange – If *Anonymous* or *Ephemeral Diffie-Hellman* is used the client parameters are sent here
  - For *Ephemeral Diffie-Hellman* parameters can be signed if server demands it

Pre-master Secret, Master Secret and Keys

- Pre-master secret
  - For RSA, random 48-byte string generated by client. Sent to server by encrypting it with server's public key
  - For Diffie-Hellman, this is the negotiated value in the key exchange
- Master secret and keyblock is calculated (in TLS) by both client and server as

master_secret = PRF(pre_master_secret, "master secret", ClientRandom || ServerRandom)

keyblock = PRF(master_secret, "key expansion", ClientRandom || ServerRandom)

Keys used are extracted from keyblock

PRF given by:
PRF(S1 || S2, label, seed) = P_MD5(S1, label || seed) ⊕ P_SHA-1(S2, label || seed)

- P_hash is an iterated HMAC producing a variable length output.

Usage of Random Numbers

- Provide a known seed to the PRF
  - similar to a salt in password hashing
- Allow both client and server to contribute to the key generation (key agreement)
- Avoid replay attacks
  - A sniffed session cannot be replayed by a fake client or fake server
  - New random number → new MAC in finish message

Some Differences Between SSLv3 and TLS

- Different version numbers
- Different functions to compute master secret and keyblock (still MD5 and SHA)
- Padding in SSL is minimum necessary, while in TLS it is can be any size
  - Arbitrary padding size helps preventing traffic analysis in which length of messages is analyzed
- Finished message calculated differently. TLS uses PRF
- Fields included in certificate verify hash are different
- HMAC in record layer computed slightly different

SSLv3: HMAC = H[(K || opad) || H[(K || ipad) || M]]

TLS: HMAC = H[(K ⊕ opad) || H[(K ⊕ ipad) || M]]

TLS 1.3

- RSA key exchange removed
  - Only Diffie-Hellman allowed (elliptic curves)
- CBC-mode is not used for block ciphers
  - BEAST attack (2011)
  - Lucky 13-attack (2013)
  - POODLE attack (2014)
- Round trip time (RTT) in handshake has been decreased
  - TLS <1.3: 1-RTT for resumptions, 2-RTT for initiation
  - TLS 1.3: 0-RTT (Pre-shared key, no PFS for first client data) or 1-RTT for resumptions, 1-RTT for initiation

TLS Man-in-the-middle Attack

- Any CA that you trust can create a certificate that you will trust
- **Typical connection (no attack)**

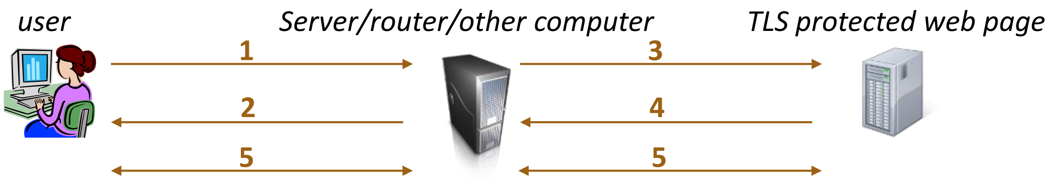
1. Alice sends Client Hello
2. Intermediate server forwards Client Hello
3. Web page answers with Server Hello, Certificates and Server Hello Done
4. Intermediate server forwards Server Hello, Certificates and Server Hello Done
5. User encrypts pre-master secret with web page public key

Result: Only user and web page knows secret keys

User can check address bar to see that certificate actually belongs to web page

TLS Man-in-the-middle Attack

- This could happen instead

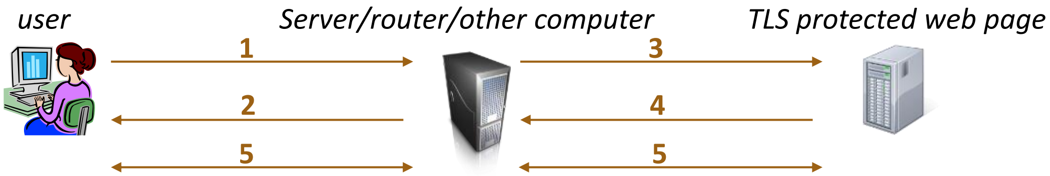


1. Alice sends Client Hello
2. Intermediate server looks at destination and creates a certificate for destination (knowing the private key), and returns this certificate together with Server Hello and Server Hello Done
3. Intermediate server sets up a new TLS connection to the web page that the user requested.
4. Web page accepts this connection (of course)
5. When user sends encrypted information to web page, intermediate server can decrypt and possibly make changes and then re-encrypt traffic

User checks address bar to see that certificate actually belongs to web page (but it does not)
This will work as long as there is a trusted CA certificate from the intermediate server in the browser, e.g., corporate networks can use this

The Nokia Man-in-the-Middle

- In January 2013, it was discovered that Nokia did a variant of a MITM attack



1. User wanted to connect to web page using TLS, but connection was made to a Nokia server (forced by browser)
2. Nokia server returned valid certificate for itself
3. Nokia server made TLS connection to web page
4. Web page accepted connection from Nokia's server
5. All communication was decrypted and re-encrypted by Nokia's server

- Nokia server was just a proxy (debatable if it counts as MITM)
- **Upside:** Data could be compressed and rewritten in order to provide more efficient browsing
- **Downside:** Nokia could read your passwords, bank info, medical journals, etc.

Implementation bugs

- February 2014
- Small implementation mistakes can have huge security impact
- Code used in iOS 6, iOS 7, OS X (some versions)

Do this if error

Always do this

Never do this

Always do this

```
SSLVerifySignedServerKeyExchange(...)
  ...
  if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;

  ...
  err = sslRawVerify(...)

  fail:
    ...
    return err;
```

Small, simple mistake – terrible consequences

- **Result**: Man-in-the-middle attack possible when Ephemeral Diffie-Hellman was used
  - Signature on Diffie-Hellman parameters was not checked at all