Stack of secure protocols

Luca Ferretti

Protocolli e Archietture di Rete

Laurea Triennale Informatica

Università degli Studi di Modena e Reggio Emilia

Understanding secure communications [1]

Securing communications regards any scenario involving **data in motion**: the characteristics of the communication might largely <u>differ depending on the involved scenario and protocols</u>

As an example:

- **Half-** or **full-duplex** communications
- Communications might be oriented to **streams** or to **messages**
- Communications with a **single** or **multiple receivers** (or “groups”)
- Support to **synchronous** or **asynchronous** communications
- Support to **application-specific requirements**
  - e.g., server-side recording in video conferences

Secure channels

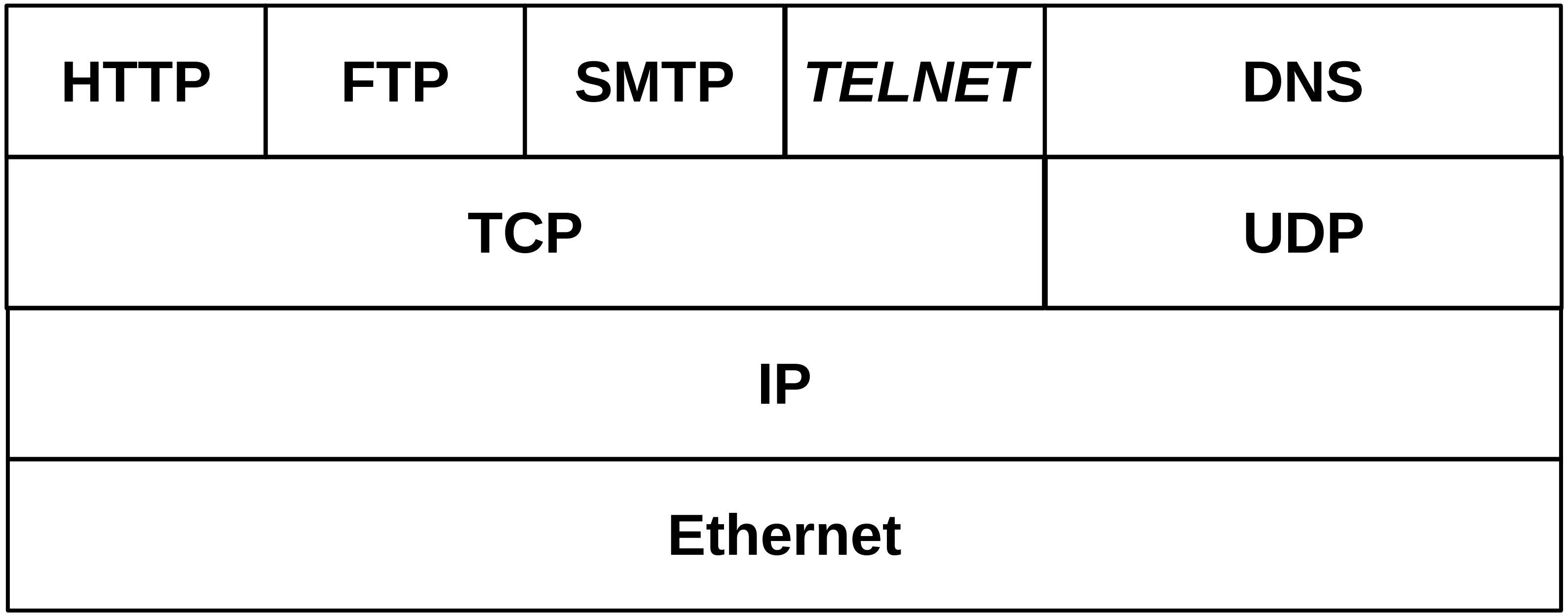
- We know that end-to-end communications in Internet protocols involve establishing multiple “hop-by-hop” communications
  - H2N (Links), Network (Nodes), Application,…
- Establishing a **secure channel** *typically* refers to establish a secure communication between two participants at a certain layer of the networking protocol stack

“Secure channels” VS “End-to-end security”

- Securing communications of an application might be more complex
  - It might make use or combine multiple (hop-by-hop) secure communications, and consider application-specific requirements
  - Might require some additional protocols (e.g., also use secure storage)
- **Beware! Terms often misused and, in general, terms are often ambiguous, thus be careful in understanding what the protocol is protecting!**
  - Sometimes (rarely) communications security at the application layer is formally referred to as “Object security” (RFC3552)

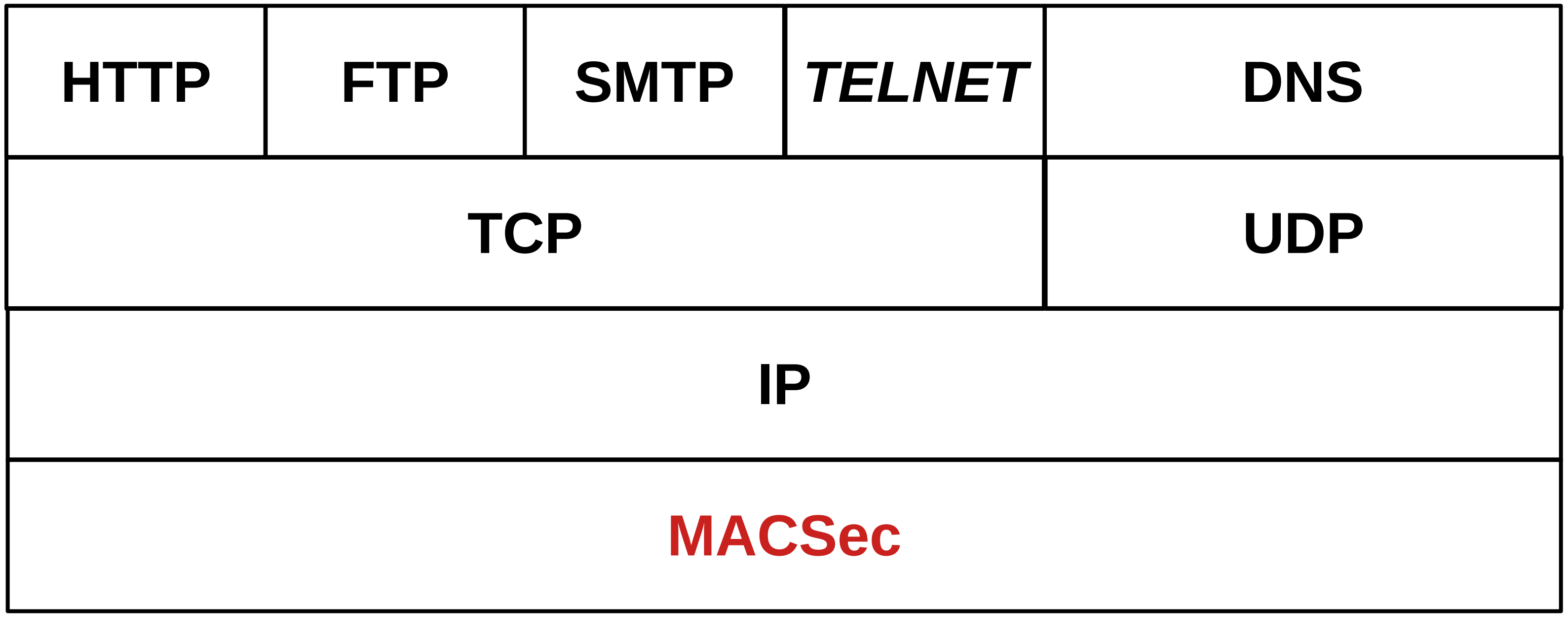
Secure protocols for the TCP/IP stack [1]

- We can implement secure communications at different layers of the TCP/IP stack
- Internet has been designed aiming at <u>performance</u> and <u>reliability</u>, <u>not security</u>
- A typical plaintext (unprotected) network TCP/IP network stack could be as following (some protocol could already be *deprecated*)



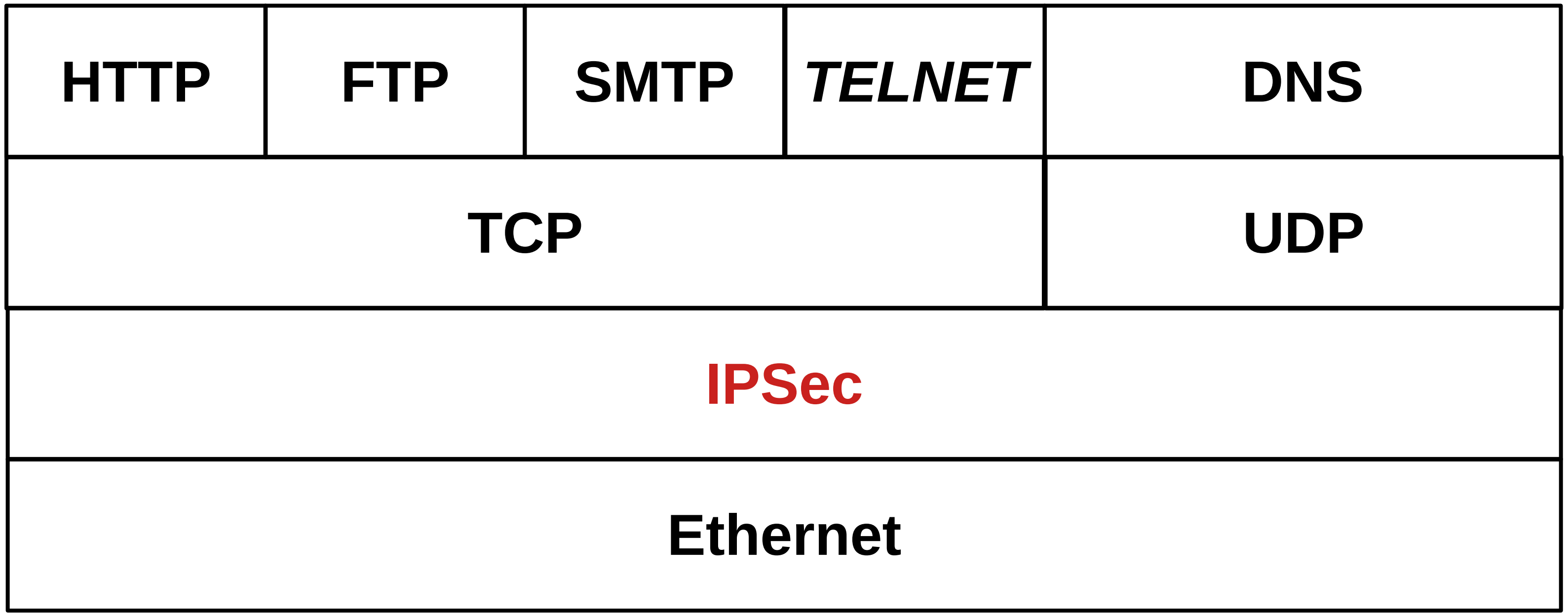
Security of the H2N layer for Ethernet

- MACSec (IEEE 802.1AE) is a quite recent proposal to extend the Ethernet Frame
  - The main aim is to provide a strong mechanism to defend against attacks from the same LAN
    - Esamples: access control, spoofing
  - Note: *it protects end-to-end security on a "physical cable"*



Secure communications at the Network layer

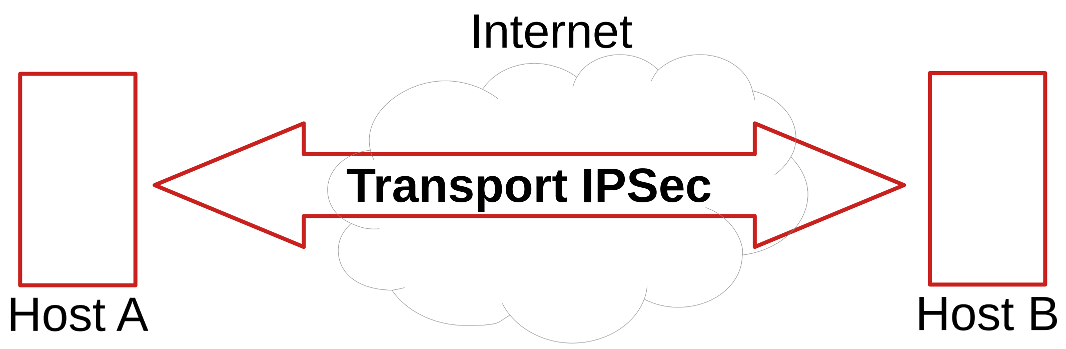
- IPSec adds security services at the IP layer
- Communications can be encrypted and authenticated among **nodes**
  - Sets the value of the protocol type in the IP header to 50
  - OpenSource implementation for Linux → StrongSwan (https://www.strongswan.org/)



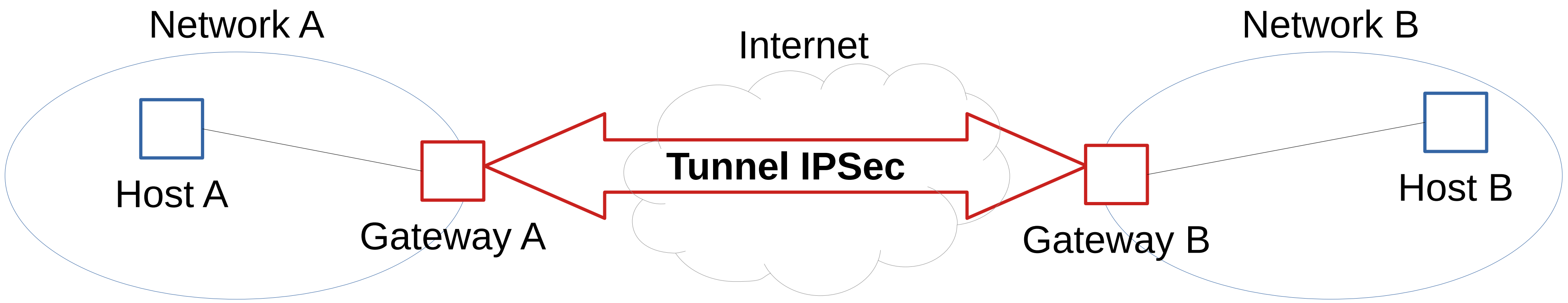
IPSec

Comes in two flavors (two modes of operations):

- **Transport mode**: secure end-to-end communication between two hosts

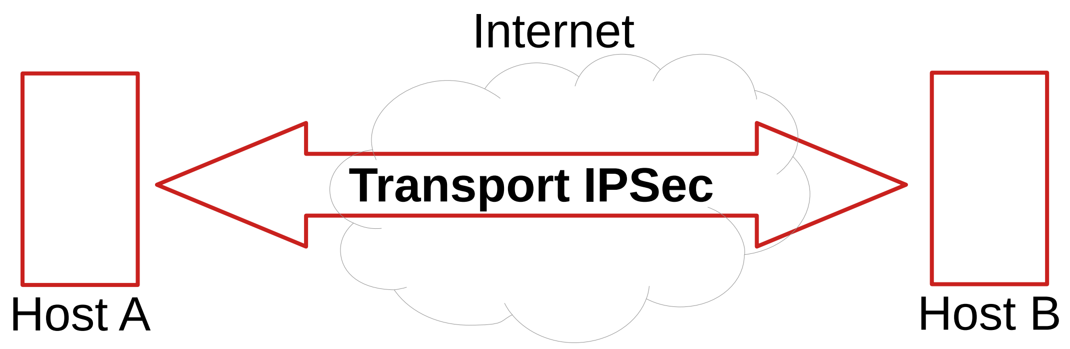


- **Tunnel mode**: secure communication between two networks



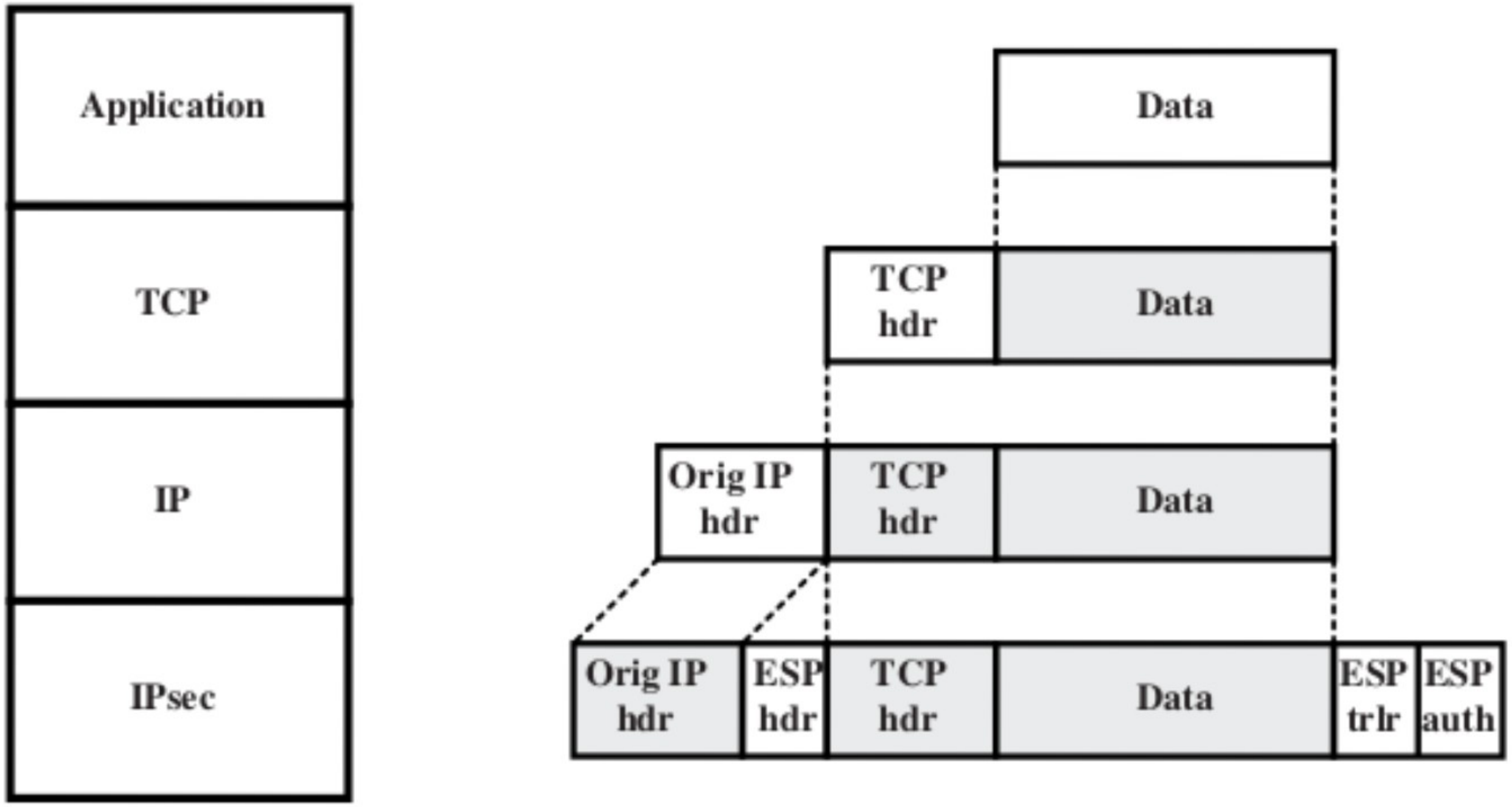
IPSec – Transport [1]

- Can be used to establish end-to-end secure communications between two nodes



- Only the payload of IP packets is encrypted
- Both the payload and the header of IP packets are encrypted
  - In case of hosts in **private networks**, need a NAT-router that is able to re-authenticate the IP header

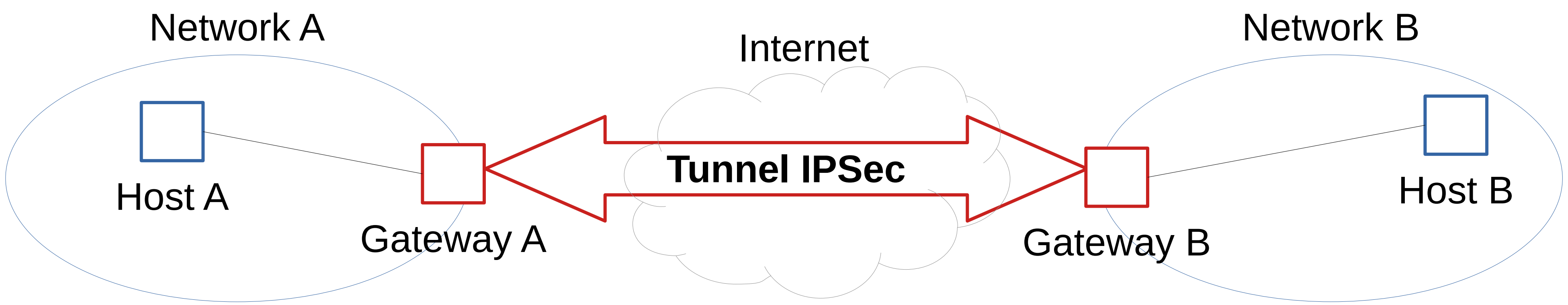
IPSec – Transport [2]



- IP payload is encrypted/authenticated
  - ESP (Encapsulating Security Payload) identifies the encryption scheme used in IPSec
- Communication is not concealed (Original IP header is not encrypted)

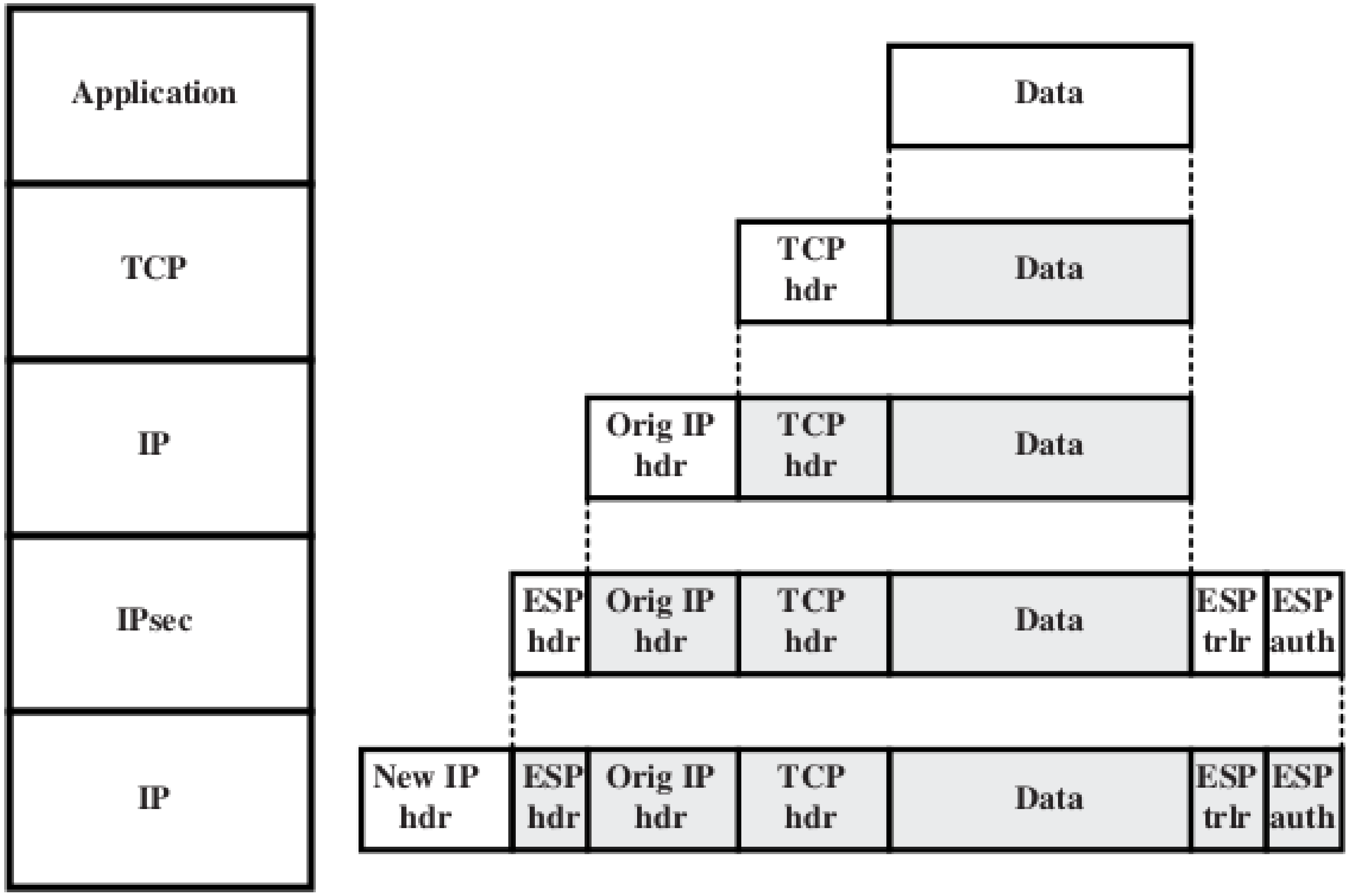
IPSec – Tunnel mode [1]

- Historically used to deploy **Virtual Private Networks (VPN)**
  - The gateways transparently encrypt all data between the two networks



- In tunnel mode, the gateways **encrypt and encapsulate IP packets** received by the host within IP packets sent to the network

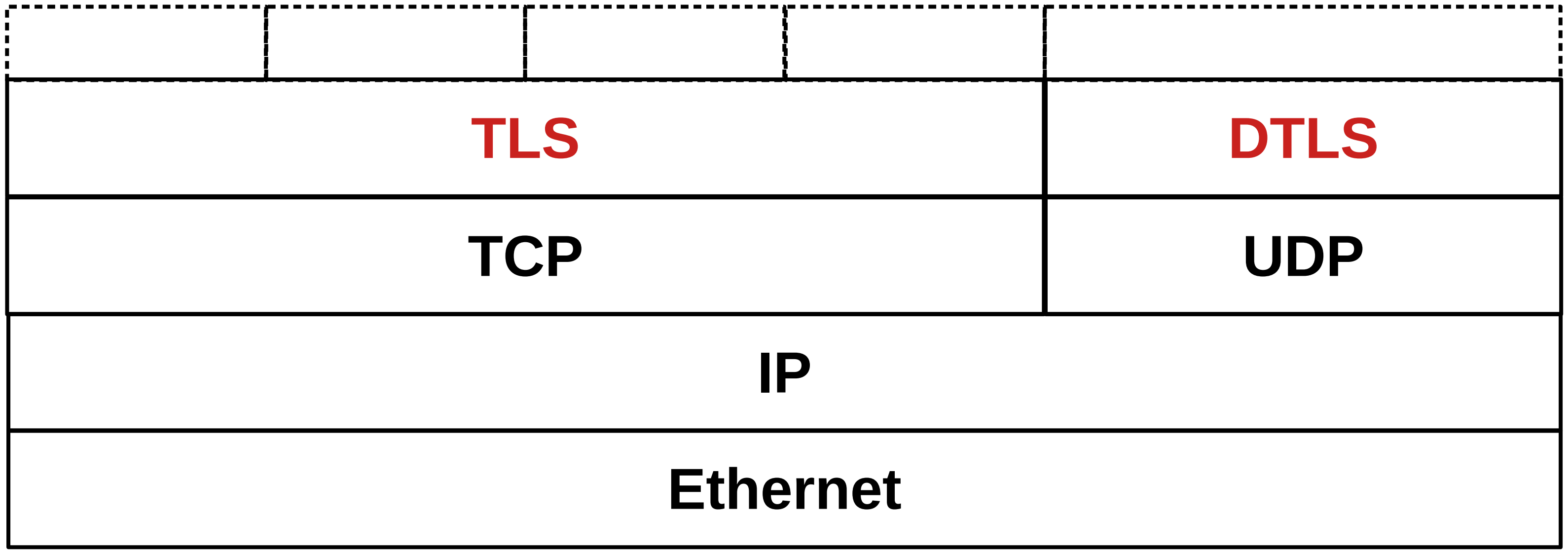
IPSec – Tunnel Mode [2]



- The entire IP datagram is encrypted/authenticated and encapsulated in a new IP header

Secure communications at the Transport layer

- TLS (Transport Layer Security) and DTLS (Datagram TLS) are protocols to provide security guarantees to TCP and UDP, respectively
  - TLS is by far the widely most used protocol for secure communications
    - Last version 1.3, deprecated prior 1.2 (TLS 1.0 and TLS 1.1)
    - Versions prior to 1.0 were known as SSL protocols (sometimes the SSL name is still informally used)



DTLS

- DTLS (Datagram Transport Level Security) is typically **based on TLS**
  - DTLS, as UDP, is datagram-oriented…
    - No stream → no protection against re-ordered packets

    …however, it still requires executing an **handshake**
  - *RFC9147*
  - ***optionally supports replay detection***
    - *Sec. 3.4: "…The replay detection feature is optional, since packet duplication is not always malicious but can also occur due to routing errors. Applications may conceivably detect duplicate packets and accordingly modify their data transmission strategy"*

TLS handshake (version <=1.2)

- Both protocols require to first execute an **handshake** that includes four phases

  1) **Establish security capabilities**
     - Protocol version, **Session ID**, **Cipher Suites**
       - DTLS also includes initial sequence numbers for replay protection of the handshake, and optional replay protection of exchanged data

  2) **Server Authentication and Key Exchange**
     - Server sends certificate (and possibly the certificate chain) and the signed key exchange contribution

  3) **Client Authentication (optional) and Key Exchange**
     - The client sends its certificate only if he wants to authenticate, and its key exchange contribution (signed, if authenticated)

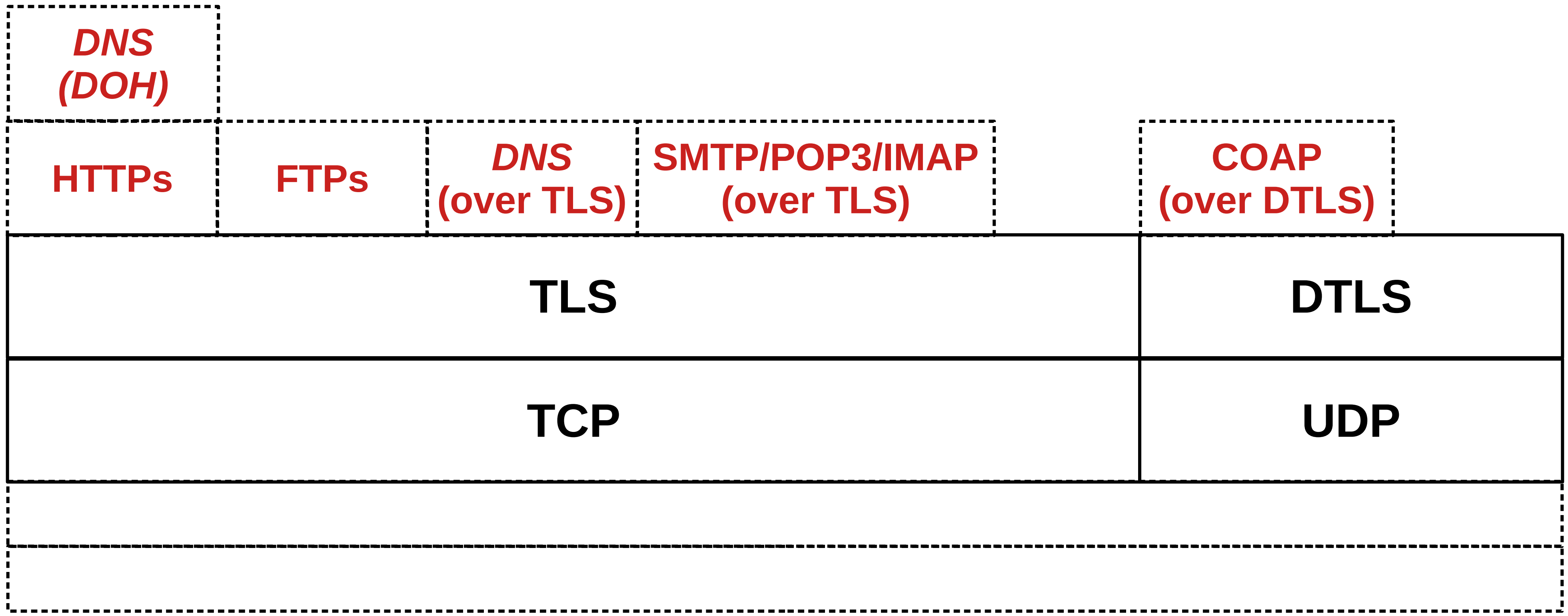
  4) **Finish**
     - Confirm the end of the handshake

TLS 1.3 vs TLS 1.2

- The main aims of TLS1.3 is to improve **performance** of TLS1.2 and to improve its security by removing dangerous **deprecated configurations**
- Performance
  - It deprecates many vulnerable legacy cipher suites (e.g., DES, RC4, SHA1) and provides support for novel cipher suites (e.g., ChaCha20, EdDSA)
- Reduces the number of required Round Trip Times (RTT) by 1
- Two major modifications that should be known by system administrators:
  - **No support for RSA-based key exchange** (no kex based on asymmetric encryption)
  - Support for **0-RTT requests** on cached sessions with <u>no replay attacks protection</u>
    - See e.g., https://blog.cloudflare.com/introducing-0-rtt/

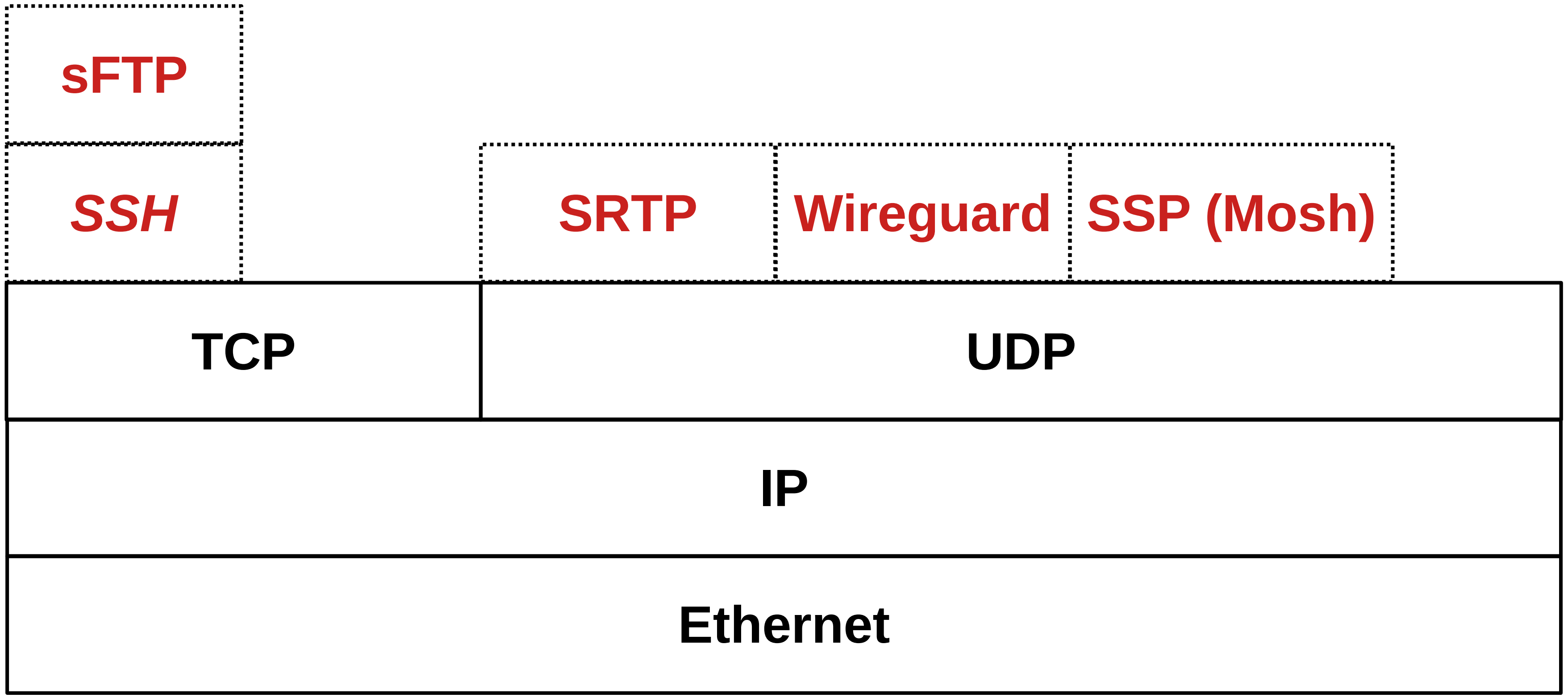
Secure communications at the Transport layer

- Many applications make (almost) transparent use of secure transport protocols to establish secure communications among applications



Designing novel secure application protocols [1]

- However, due to their characteristics, some protocols have been completely (re)designed without using “transparent” secure layers underneath
  - e.g., SSH (Secure SHell) is a protocol for remote terminals which is based on TCP that completely redesigns Telnet



Designing novel secure application protocols [2]

- SSH → Secure SHell
  - SFTP: sub-system of SSH for secure file transfer
- SRTP → Secure Real-Time Protocol
- Wireguard → Recent popular protocol for transport-layer VPNs
- SSP → Secure Synchronization Protocol
  - Mosh → Mobile SHell, a program similar to SSH, but designed to allow *roaming* clients and *high latency* connections

A closer look at HTTPs

- Although HTTPs is obtained by transparently using HTTP over TLS, we need to include

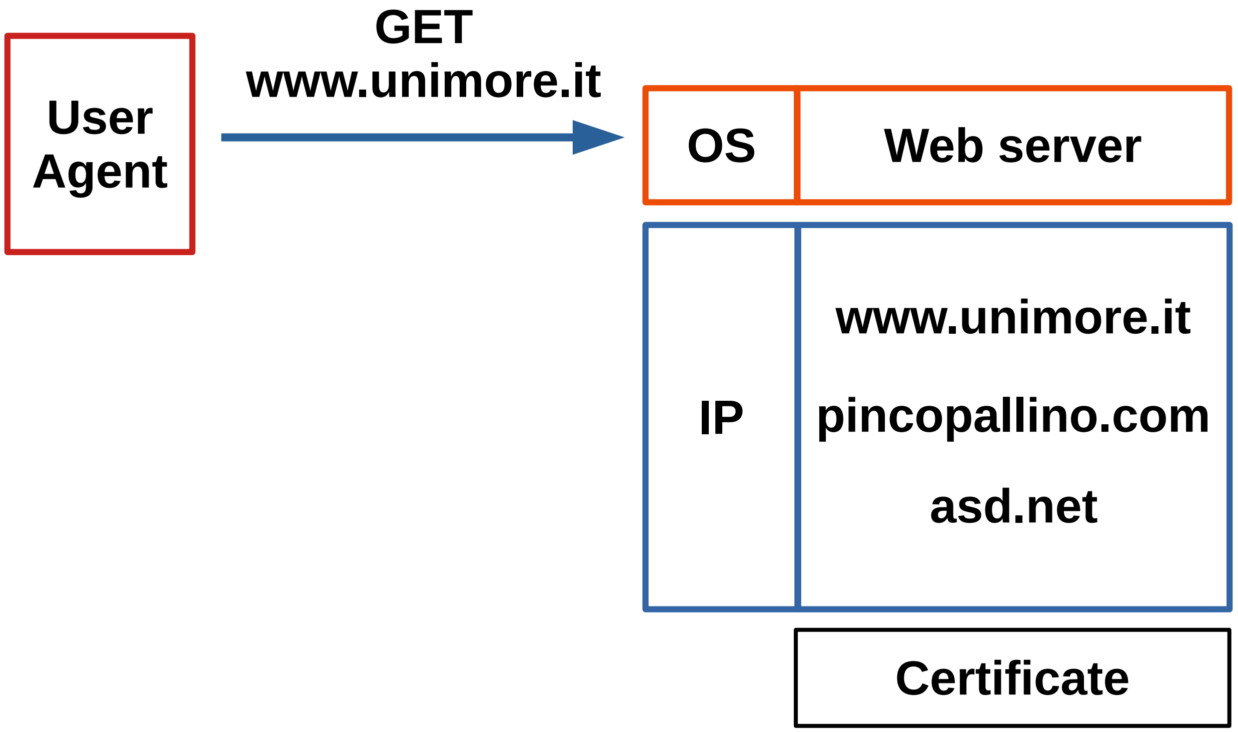
  → **TLS extensions for Virtual Hosting**

- Although we might say that we delegate all security issues to TLS, system administrators should still take care of some important configuration details

  → **Protecting against downgrade attacks (HTTPs → HTTP)**

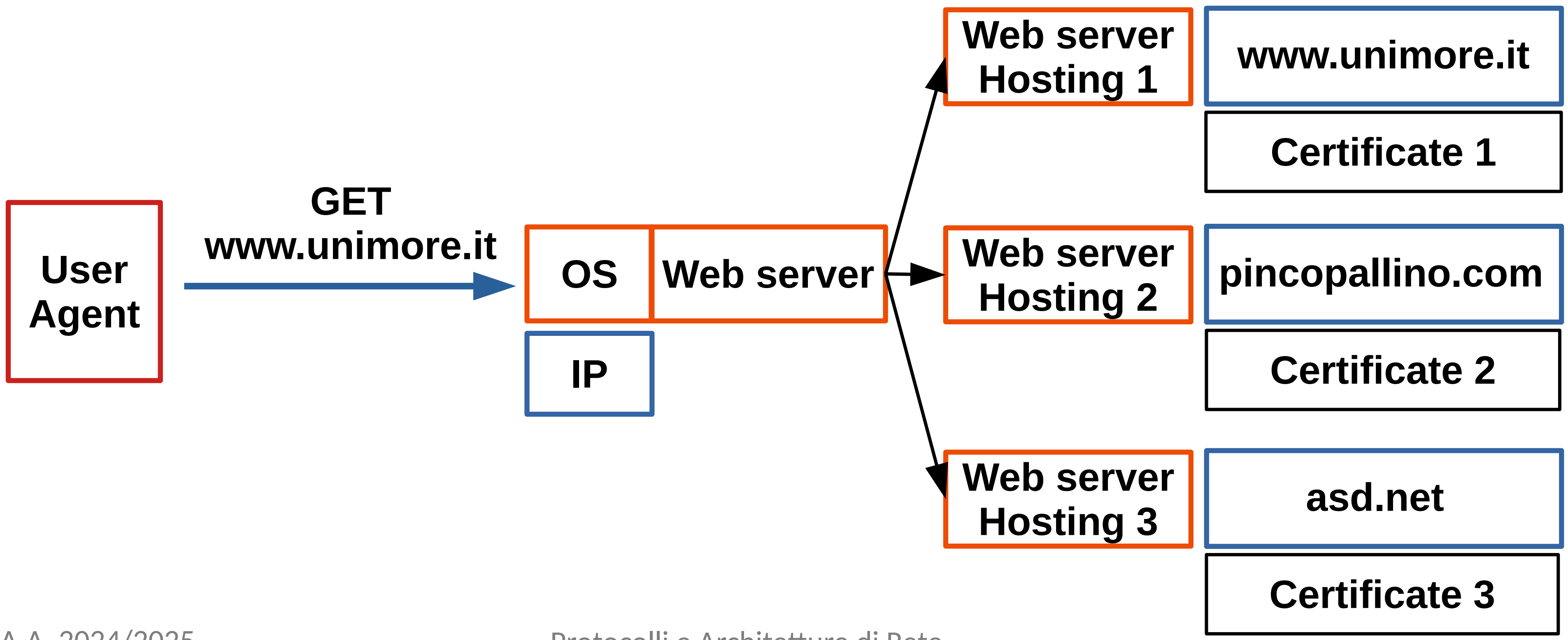
HTTPs and Virtual Hosting [1]

- Virtual Hosting: multiplexing multiple Web sites on the same Web Server



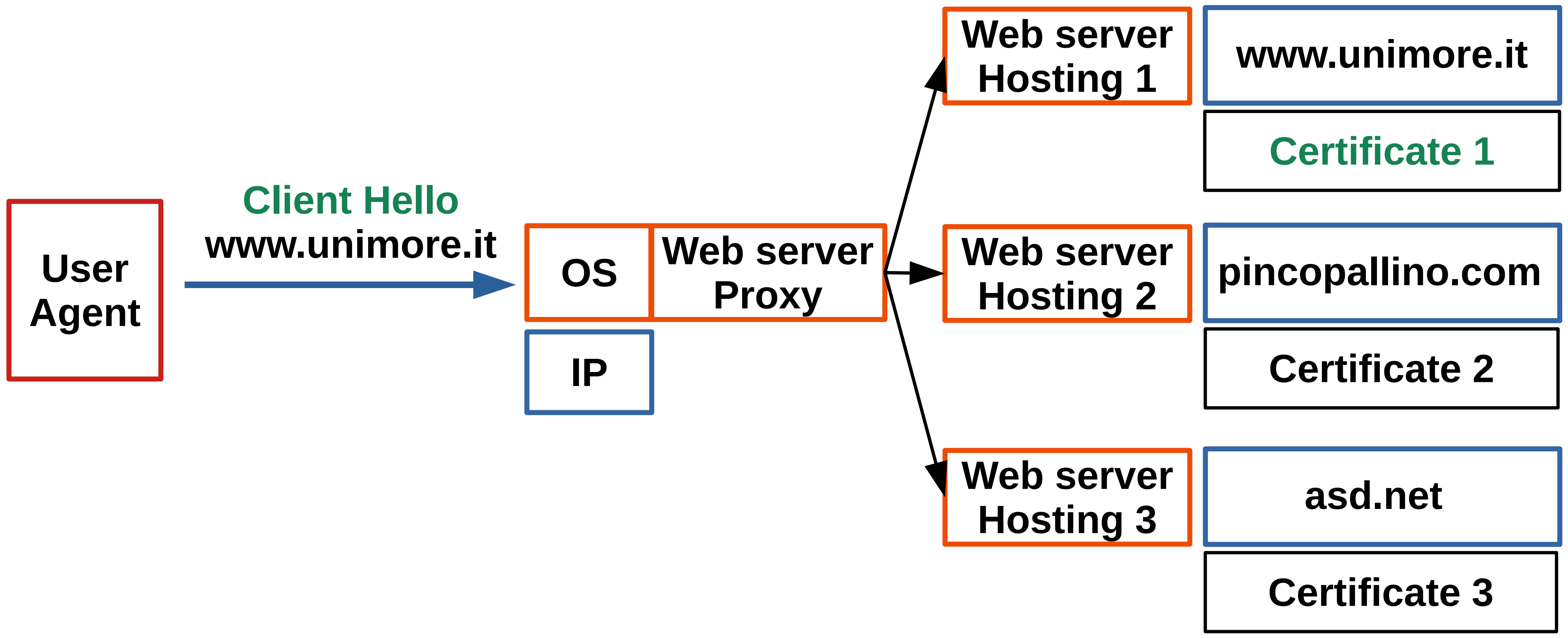
HTTPs and Virtual Hosting [2]

- Virtual Hosting and **distributed architectures**
  - Systems might deploy different websites at the same IP address on multiple machines
  - **Each machine might use a different certificate**
    - The name of the server will be sent **encrypted in the payload**
    - **How to correctly dispatch the request?**



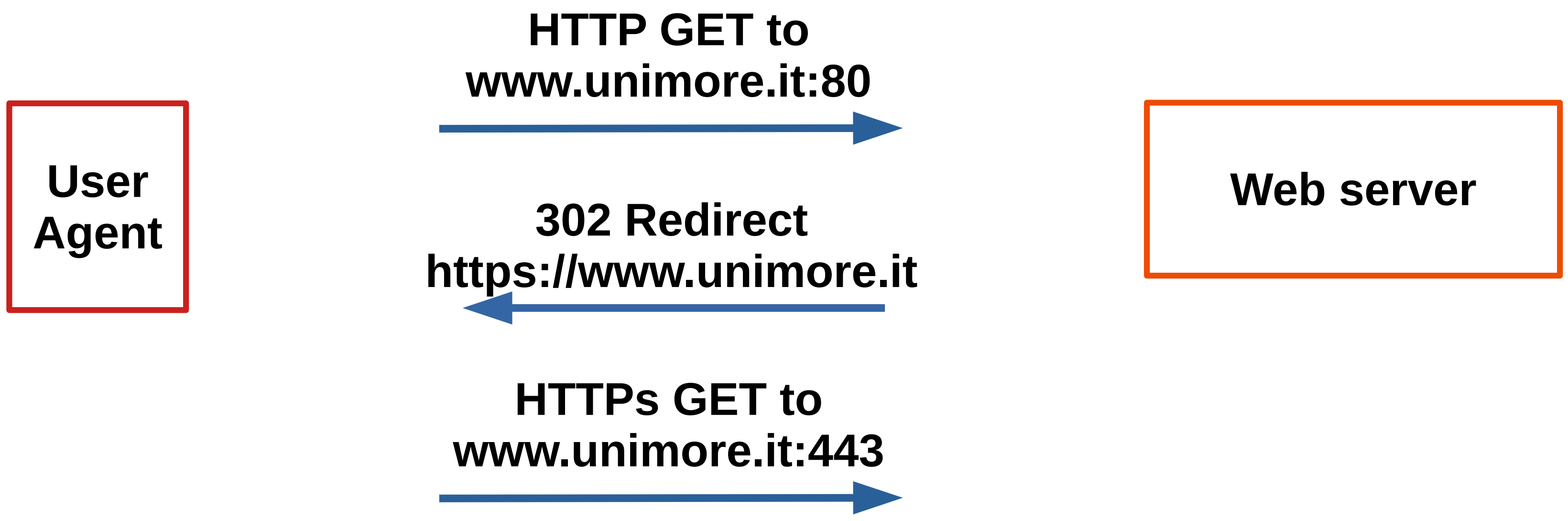
HTTPs and Virtual Hosting [3]

- Virtual Hosting information is transferred **in plaintext in the TLS handshake!**
  - The **client hello** includes a special-purpose **extension** (Server Name Indication - SNI) that can be used by the proxy Web server to select the correct server



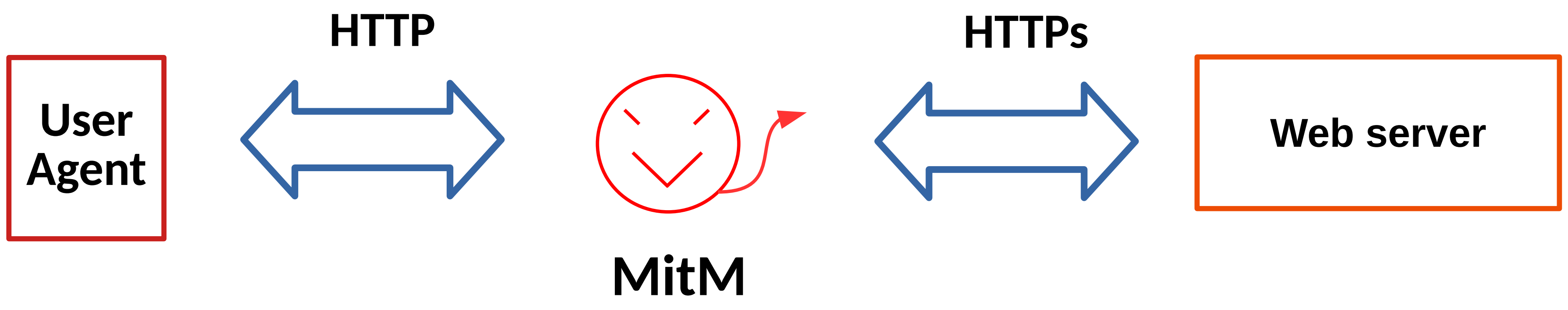
SSL Stripping to HTTPs: downgrade HTTPs to HTTP [1]

- The best practice for HTTPs Web servers that **receive requests to HTTP is to redirect them to the HTTPs endpoint**


User Agent
HTTP GET to www.unimore.it:80
302 Redirect https://www.unimore.it
HTTPs GET to www.unimore.it:443
Web server

SSL stripping to HTTPs [2]

- **SSLStripping** is the commonly used name to refer to a technique that **rewrites HTTP response headers** and HTML documents **in order to remove possible redirections to HTTPs**

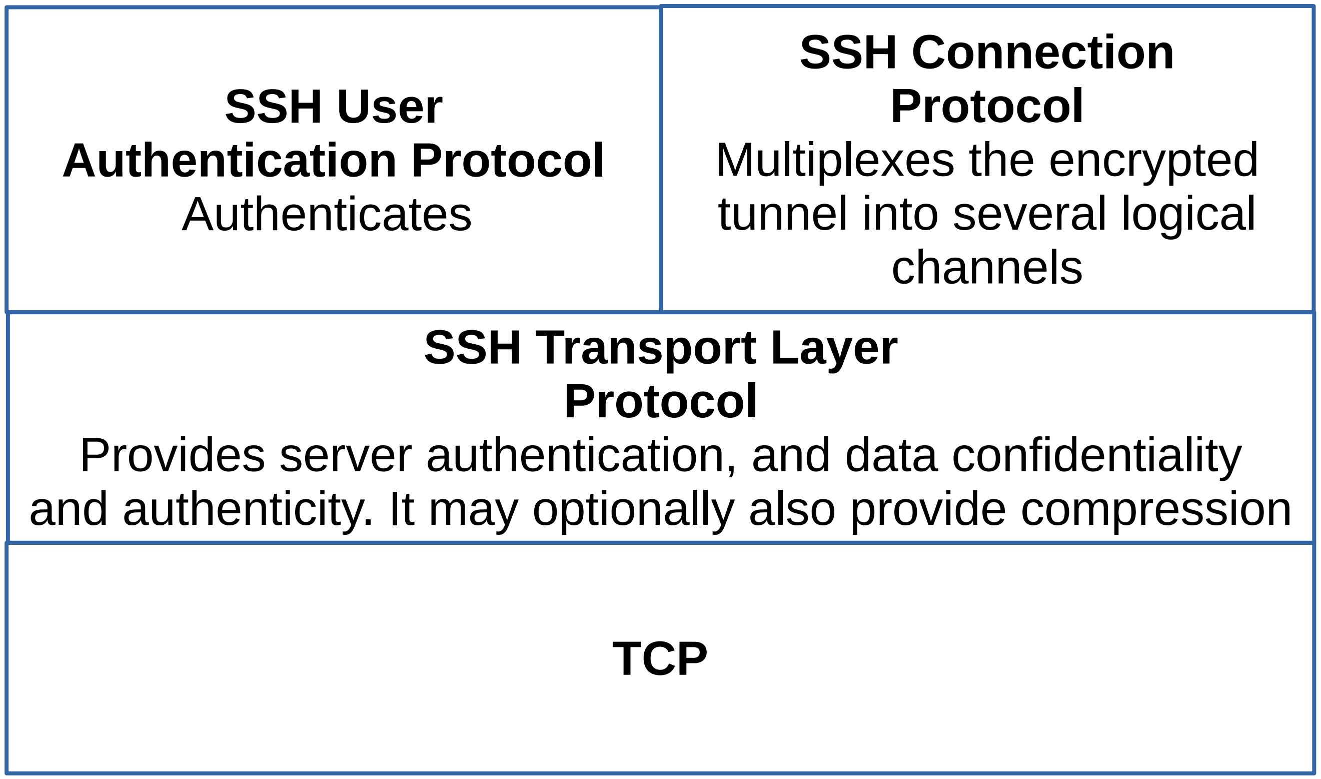


- A more secure approach is to configure clients to always (and only) connect with HTTPs → https://www.eff.org/https-everywhere

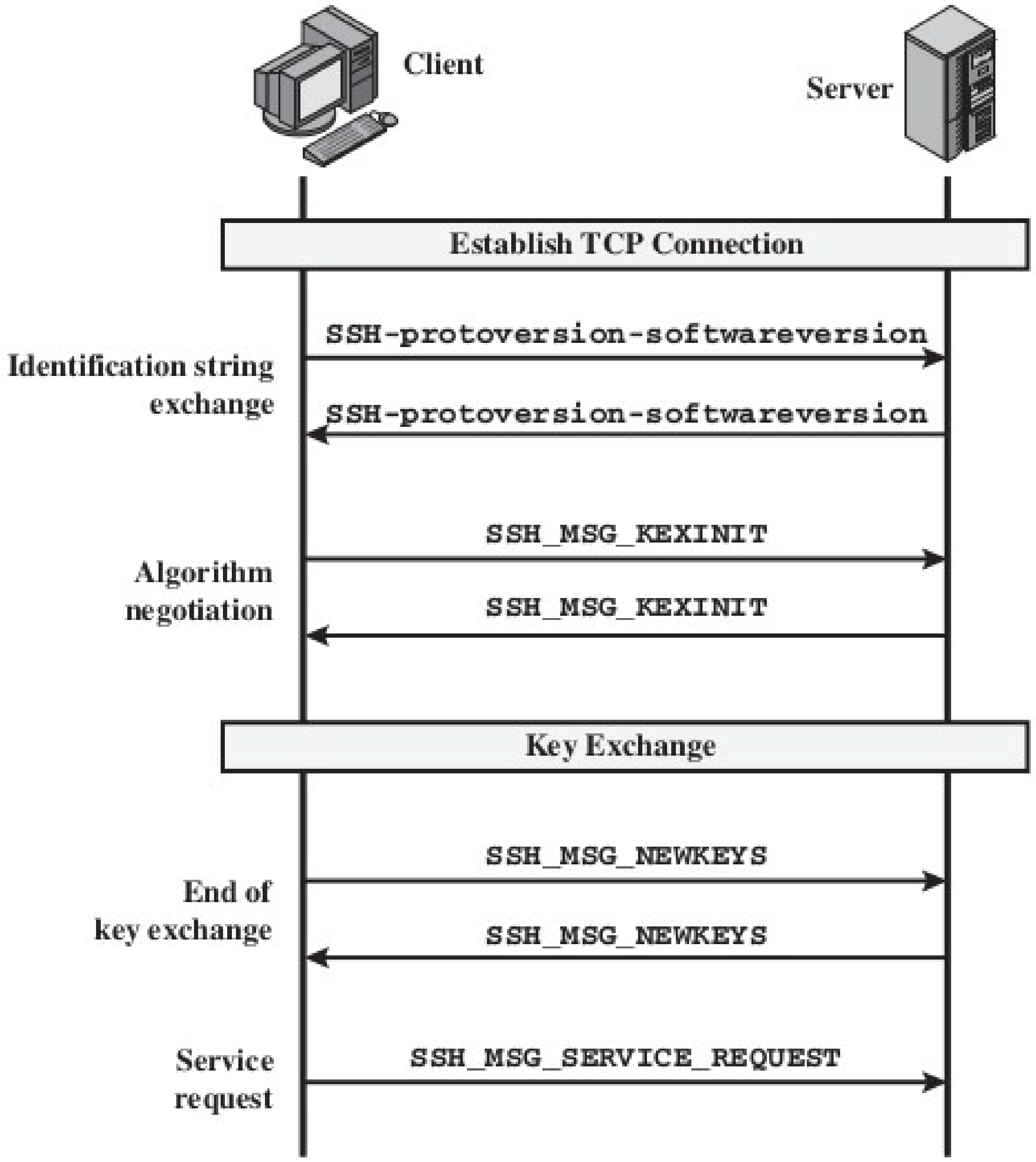
- HTTP String Transport Security **(HSTS)** → the server sets the **Strict-Transport-Security** in the first HTTP request to tell the client to always use HTTPs in the next requests

Secure Shell (SSH)

- SSH is a protocol to manage remote hosts in a secure manner
  - Replaced *telnet* and *rlogin*
  - *Does not use TLS*, implements its own transport layer security protocol


SSH User Authentication Protocol
Authenticates
SSH Connection Protocol
Multiplexes the encrypted tunnel into several logical channels
SSH Transport Layer Protocol
Provides server authentication, and data confidentiality and authenticity. It may optionally also provide compression
TCP

SSH operations flow


Client
Server
Establish TCP Connection
Identification string exchange
SSH-protoversion-softwareversion
SSH-protoversion-softwareversion
Algorithm negotiation
SSH_MSG_KEXINIT
SSH_MSG_KEXINIT
Key Exchange
End of key exchange
SSH_MSG_NEWKEYS
SSH_MSG_NEWKEYS
Service request
SSH_MSG_SERVICE_REQUEST