50.020 Security
Lecture 17 - Security Protocols (ARP, TLS, NS)

This lecture

- This lecture:
  - Security protocols and attacks
    - ARP (Address Resolution Protocol) and ARP Spoofing
    - TLS
    - Needham-Schroeder protocols and attacks

ARP and ARP Spoofing

General Eavesdropping

- We discussed generic eavesdropping earlier
  - Attacker is passive, and wants to learn content of messages
- How does the abstract concept of eavesdropping apply in practise?
- We now look at three scenarios:
  - Internet
  - LAN
  - Wireless networks

Eavesdropping on the Internet

- In general, anyone forwarding your traffic can read it
  - Your WiFi access point
  - Your ISP
  - Other ISPs that forward your traffic (core routers)
  - Infrastructure on the receiver side
- But third parties cannot!
- The problem is: how to know who forwards my traffic
  - The user cannot directly choose, or know
- Big push nowadays to use TLS (e.g. HTTPS) for everything
  - This *should* hide most data from intermediate devices

Eavesdropping on the LAN

- What about local networks (at work, Uni)?
  - Can I just read the traffic of my co-worker?
- Three ways to achieve this
  - Physical Man-In-The-Middle (*taps*)
  - ARP-Spoofing
  - DHCP spoofing

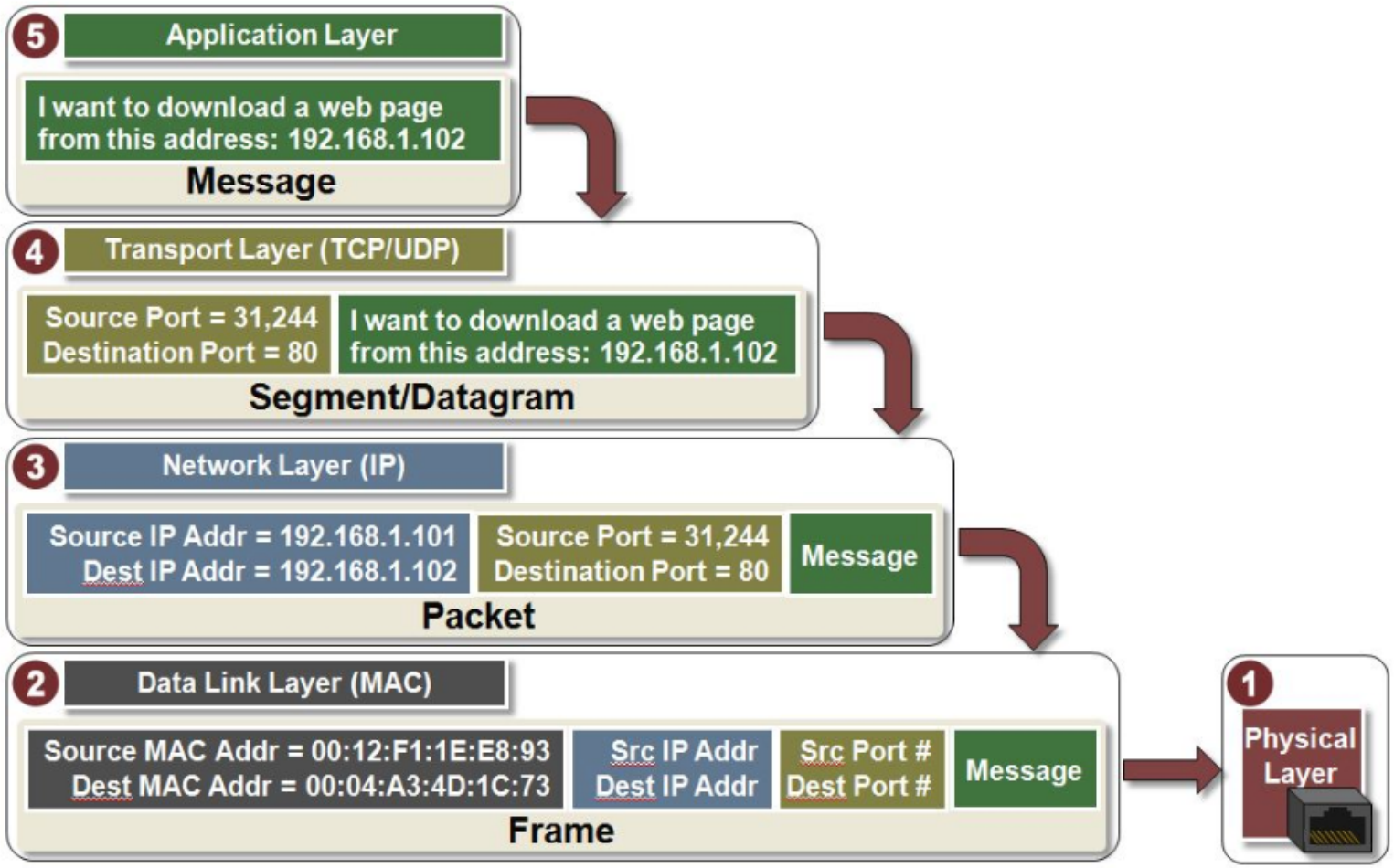
Network taps

- They provide a read-only copy of *all* traffic
- Traffic is including low-layer headers (Ethernet upwards)
- Invisible to network
- Commercial products are available, for example: throwing star LAN Tap `https://www.amazon.com/Throwing-Star-LAN-Tap-communications/dp/B01COWCXF6`
- Can be quite easy to build, for example, `https://www.youtube.com/watch?v=2tsvBnTIjFo`:

Address Resolution Protocol: Recap on TCP/IP Model

Layer #	Layer Name	Protocol	Protocol Data Unit	Addressing
5	Application	HTTP, SMTP, etc...	Messages	n/a
4	Transport	TCP/UDP	Segments/ Datagrams	Port #s
3	Network or Internet	IP	Packets	IP Address
2	Data Link	Ethernet, Wi-Fi	Frames	MAC Address
1	Physical	10 Base T, 802.11	Bits	n/a

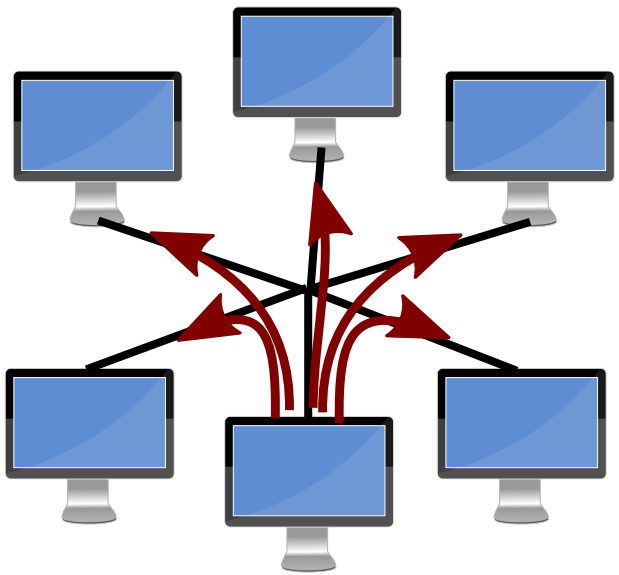
Address Resolution Protocol: Recap on TCP/IP Model



Address Resolution Protocol

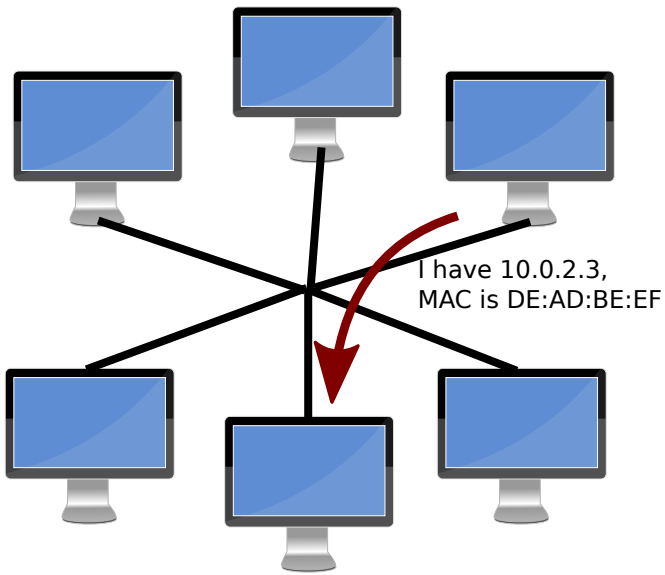
- Traffic to hosts in the same subnetwork is actually sent directly using their Link-layer MAC address
- Same subnetwork:
  - Hosts are connected through switches (not routers)
  - Hosts in same IP network (e.g. 192.168.1.0/24)
- The Address Resolution Protocol (ARP) is used to find correct MAC address for local target IP

ARP example



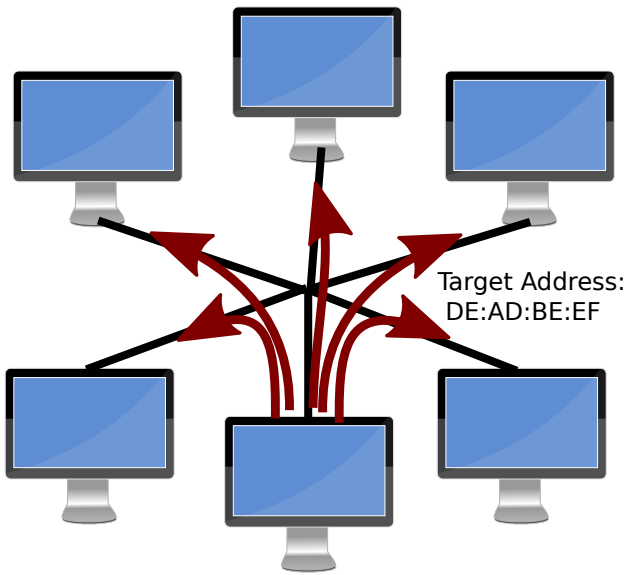
Which NIC has 10.0.2.3?

ARP example



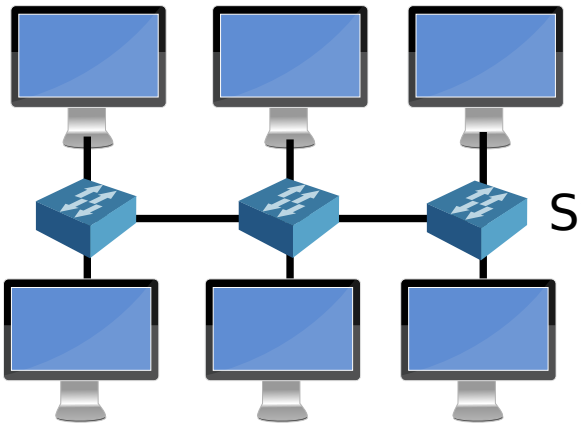
User learns target MAC

ARP example

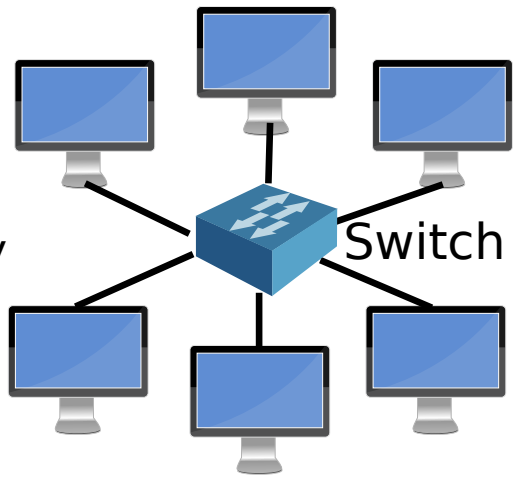


Broadcast to DE:AD:BE:EF

Ethernet Switched Topologies



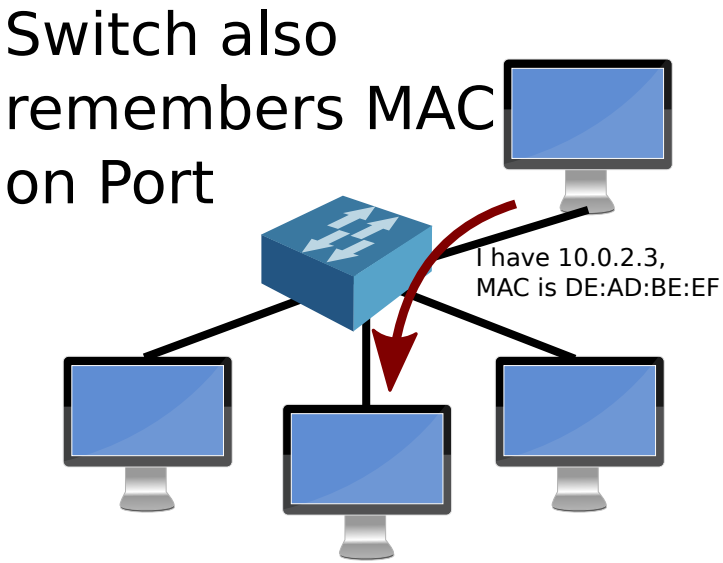
Switched Bus Topology



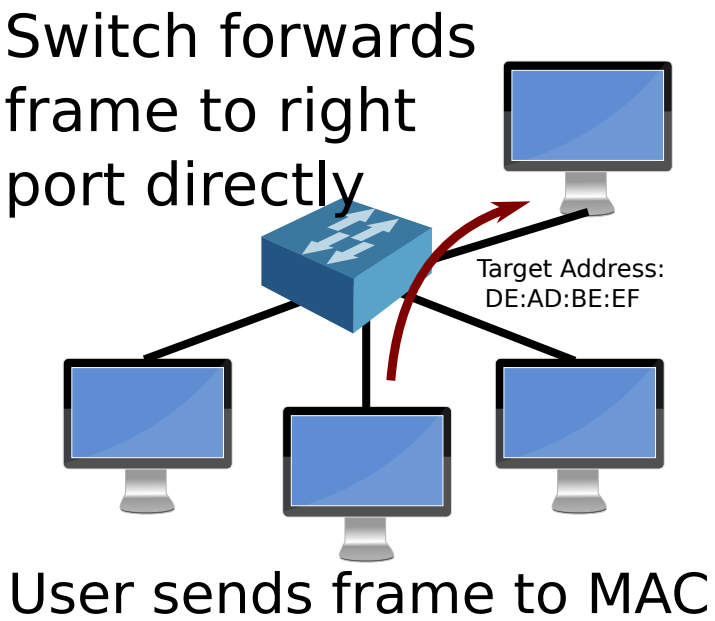
Switched Star Topology

Switch

ARP in Switched Network example



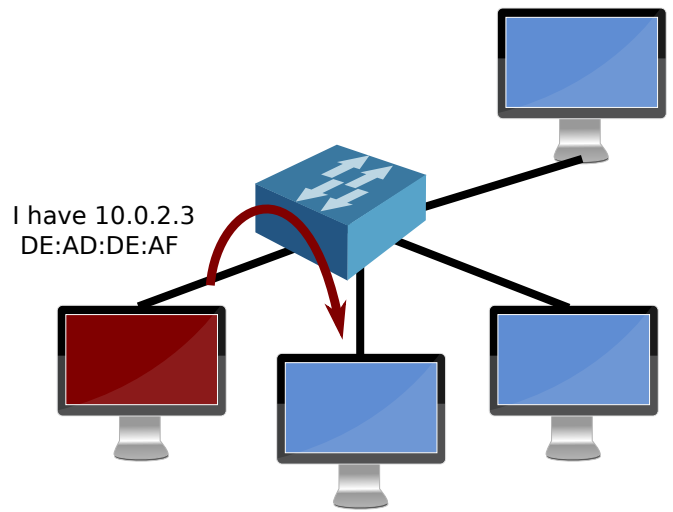
ARP in Switched Network example



ARP Spoofing

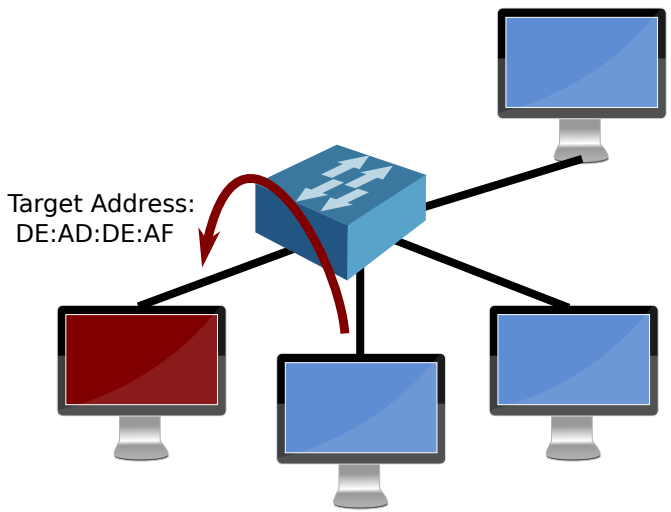
- Attacker wants to MitM traffic between Alice and Bob
- Attacker sends ARP announcements to Bob
  - Using Attacker's MAC address, but Alice's IP
- Bob then updates his ARP cache with attacker MAC for Alice's IP
- Next packet sent from Bob to Alice will go to attacker
- Attacker then forwards traffic to Alice
  - Could just eavesdrop, or manipulate
- Attacker usually attacks both Alice and Bob at same time
  - Redirecting traffic in both directions

ARP Spoofing Example



Attacker spoofs MAC

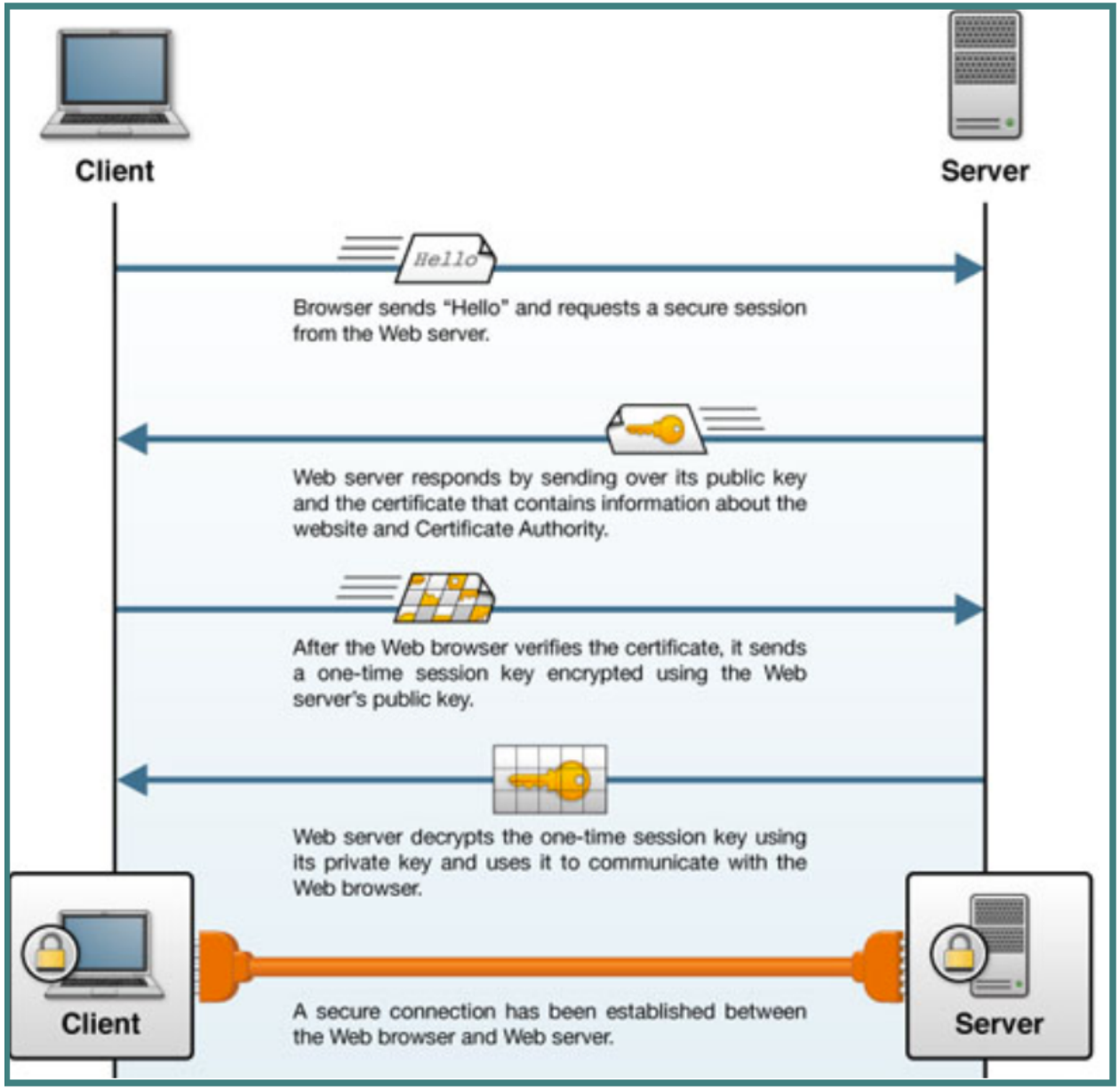
ARP Spoofing Example



User sends to Attacker

Transport Layer Security

Transport Layer Security: How It Works



Transport Layer Security: How It Works

General Steps:

- The client sends a client hello message to server, to request a secure connection.
- The server responds with a server hello message, together with his public key and digital certificate information.
- The client verifies the server's digital certificate. If verification passes, he proceeds with the next step.
- The client selects a session key, and encrypts it with server's public key.
- The server decrypts the session key with his secret key. The server sends (session key) acknowledge information to the client.
- The client and server transmit messages with the session key.

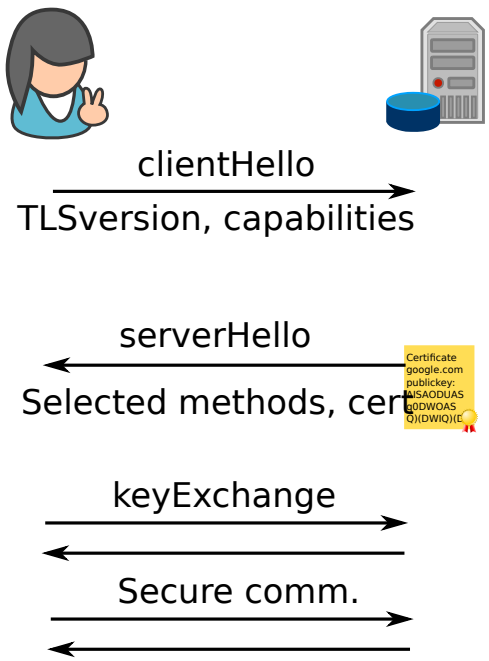
Client's certificate verification is optional.

Transport Layer Security

- TLS is an *application layer* protocol to establish secure channels
- TLS uses public-key crypto, data exchanged over this channel is
  - Authenticated
  - Integrity-protected
  - Confidential
- It's the swiss knife of network security protocols
- TLSv1.2 replaces the older (and insecure) SSL
- Current standard to establish secure channels on the internet
- Can use a number of ciphers (not all clients support all)

TLS security schemes

- TLS is using a range of schemes for authentication, key establishment, encryption, and integrity protection.
- In addition, server and client often both support older protocol version for legacy support
- TLS starts with a *handshake*, in which capabilities are exchanged, and a set of function is chosen
- The picture in this page can be considered as a simplified version of the previous picture (Transport Layer Security: How It Works)



TLS: authentication phase

Operating modes supported in TLSv1.2

- RSA
  - Client creates random key, encrypts with server e,n
- Diffie-Hellman
  - Server has static public DH key
  - Later compromise can give attacker all shared keys
- *Ephemeral* DHKE
  - Server dynamically creates public key, signs it
  - This ephemeral key is then deleted later
- Elliptic-Curve DH and ephemeral ECDH
  - Using EC instead of RSA, encrypt random key
- Anonymous Diffie-Hellman
  - No certificates are used, insecure
- Pre-shared key (PSK)
  - Some key was preshared earlier

TLS Encryption modes

- TLS also offers a wide range of encryption modes
- Block ciphers:
  - AES in CBC mode
  - AES in CBC-MAC counter mode (CCM)
  - 3DES in CBC mode
- Stream ciphers:
  - RC4, but is considered insecure
- *None* (no encryption)

TLS message integrity

- TLS also offers a selection of message integrity schemes
- HMAC
  - using MD5
  - using SHA1/later SHAs
- AES CCM mode (not discussed in this lecture)

Where to use TLS

- As TLS is application level, each application has to set up own TLS session
- It is up to the application what to send through the TLS connection
- HTTP Secure is an example:
  - The browser sends HTTP traffic through a TLS connection
- FTP, SMTP, etc. . . . can be sent via TLS
- TLS is a convenient and relatively fool-proof way to improve security
  - If you have the PKI infrastructure, or pre-shared keys

Example use of TLS

- Using TLS in flask is easy
- You need to create your own CA certificate and server certificate
- Use your CA certificate to sign your server certificate request
- Then, load your certificate and private key into your flask application

Example

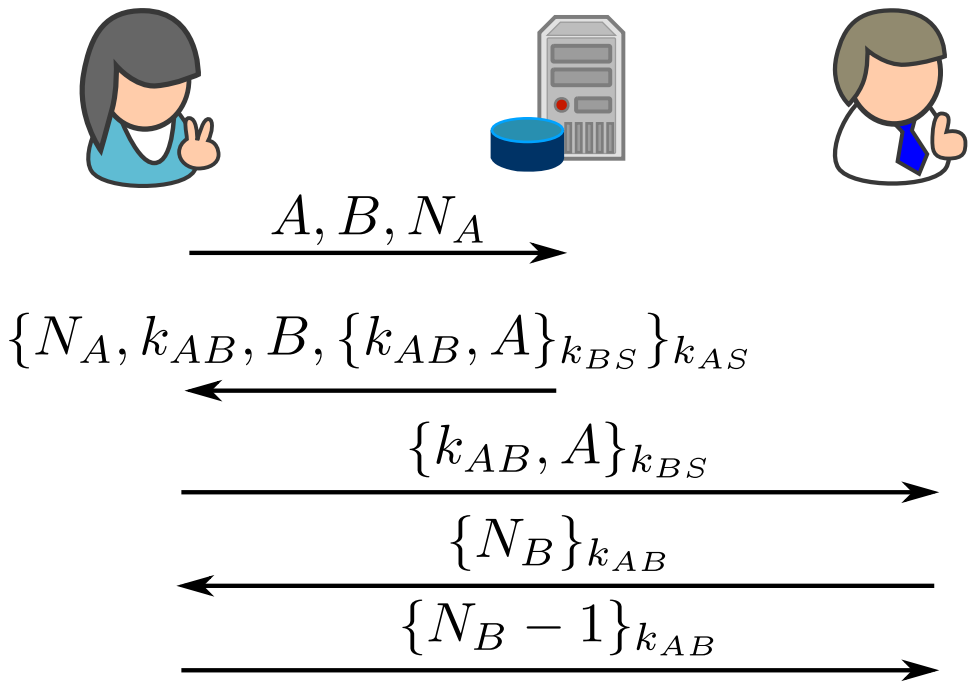
```
#Requires Python 2.7.9, upgrade if necessary
import ssl
app.run(host='10.0.1.50',port=443, debug = False,
        ssl_context=('yourserver.crt', 'yourserver.key'))
```

Needham-Schroeder protocols

Needham-Schroeder protocols

- Why we talk about Needham-Schroeder (NS) protocols?
  - NS protocol is the base for the Kerberos procotol
    - Kerberos is used in many places, e.g. MS Active Directory
- NS protocols have two variants: *symmetric* and *asymmetric*
- Both NS protocols use a server to distribute shared keys
- Difference: how to securely communicate with server
  - One uses public keys, one uses private keys
- Both initial protocols had vulnerabilities, that were fixed
- Notation used in the following: $\{\}_x$ is an encryption/signing operation using key x

Symmetric Needham-Schroeder



The protocol establishes a shared key between Alice and Bob, by using a trusted Server which is also known as Key Distribution Center (KDC) in Kerberos. No Public Key Infrastructure (PKI) is used.

Symmetric Needham-Schroeder (Continue)

In the protocol:

- Alice (A) initiates the communication to Bob (B). S is a server trusted by both parties.
- A and B are identities of Alice and Bob respectively
- K_{AS} is a symmetric key known only to A and S
- K_{BS} is a symmetric key known only to B and S
- N_A and N_B are nonces generated by A and B respectively
- K_{AB} is a symmetric, generated key, which will be the session key of the session between A and B
- In the last step, Alice performs a simple operation on the nonce, i.e., calculating $N_B - 1$, re-encrypts it and sends it back verifying that she is still alive and that she holds the key

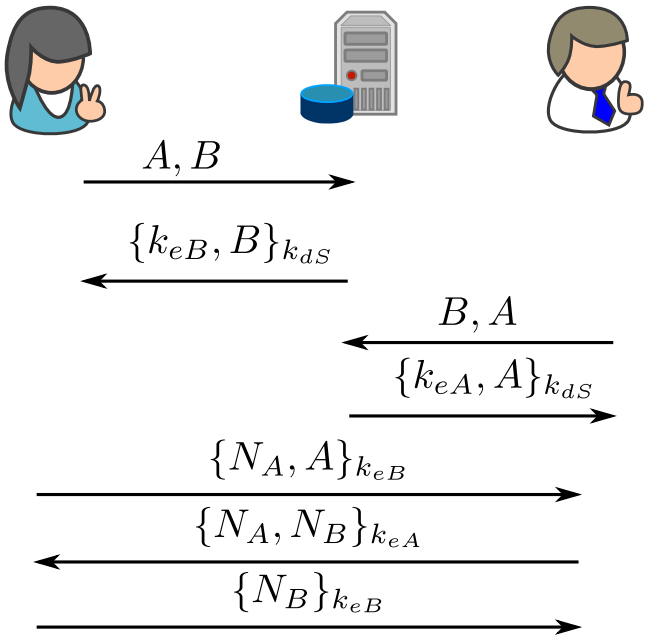
Attacks on Symmetric Needham-Schroeder

Replay attack:

- If an attacker uses an old, compromised value for K_{AB}, he can then replay the message $\{K_{AB}, A\}_{K_{BS}}$ to Bob, who will accept it, being unable to tell that the key is not fresh.

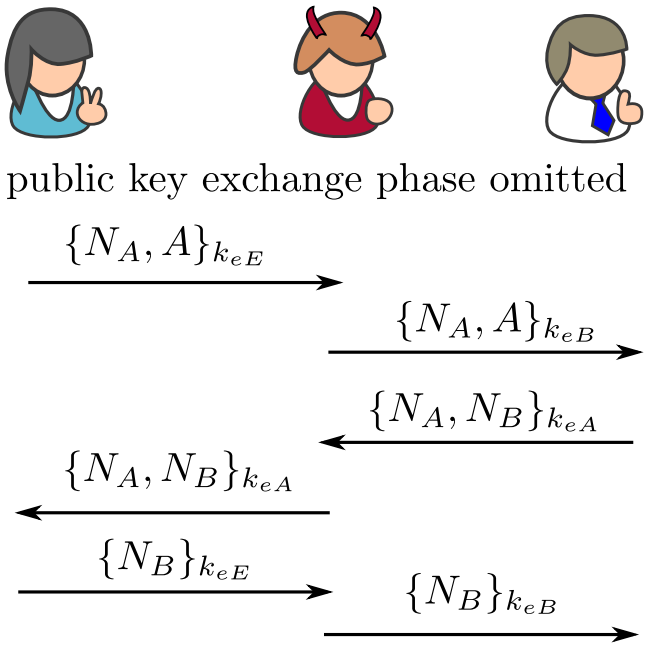
How to Fix it?
Including a timestamp (as in Kerberos protocol)

Asymmetric Needham-Schroeder



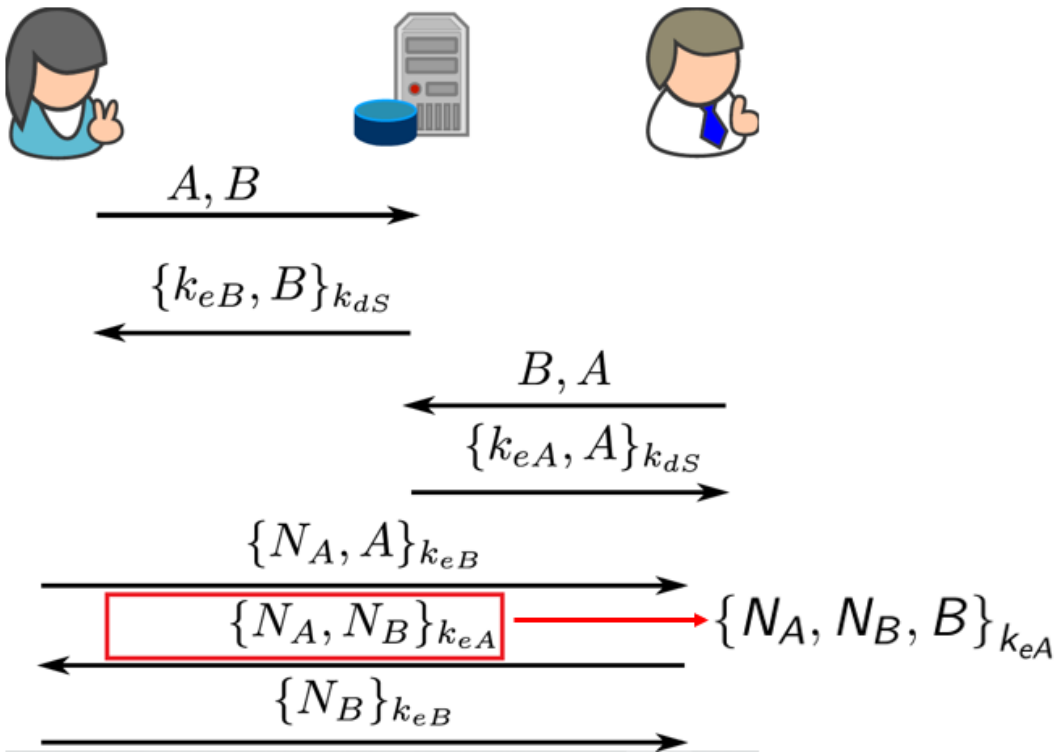
- The protocol exchanges public keys between Alice and Bob, and lets them know each other's identities
- Alice and Bob both have pre-shared public k_{eS} of Server. k_{dS} is the private key of Server.
- k_{eA} is the public key for Alice (A). k_{eB} is the public key for Bob (B).

Attacks on Asymmetric NS (Man-in-the-Middle Attacks)



- If Alice contacts Eve for some reason, Eve can re-use N_A
- Eve replays Alice's message to Bob, and convinces Bob that Bob is commutating with Alice
- As result, Bob will believe that Alice established a connection
- Alice will not realize that Eve re-used the N_A

Fixed Asymmetric NS



How to fix the Asymmetrix NS to prevent against the Man-in-the-Middle Attack?

- Include Bob's identity in Bob's responding message $\{N_A, N_B\}_{k_{eA}}$. That is, modifying this responding message to $\{N_A, N_B, B\}_{k_{eA}}$

Kerberos

- Building on symmetric NS, Kerberos realizes a network with a central authentication server, users on workstations, and service providers
- Instead of authenticating to each service provider, the user gets *tickets* to access the services
- These tickets are essentially time-limited shared keys as in NS
- The service provider will trust the tickets, as they are encrypted by a shared key between service provider and central server

Conclusions

- Protocols can get fairly complicated, and have subtle security problems
- When implementing systems, ideally do not invent your own protocol
- Use an existing library and standard
  - Rely on other experts for security analysis
  - Make your system interoperable with other systems