TÉCNICO LISBOA

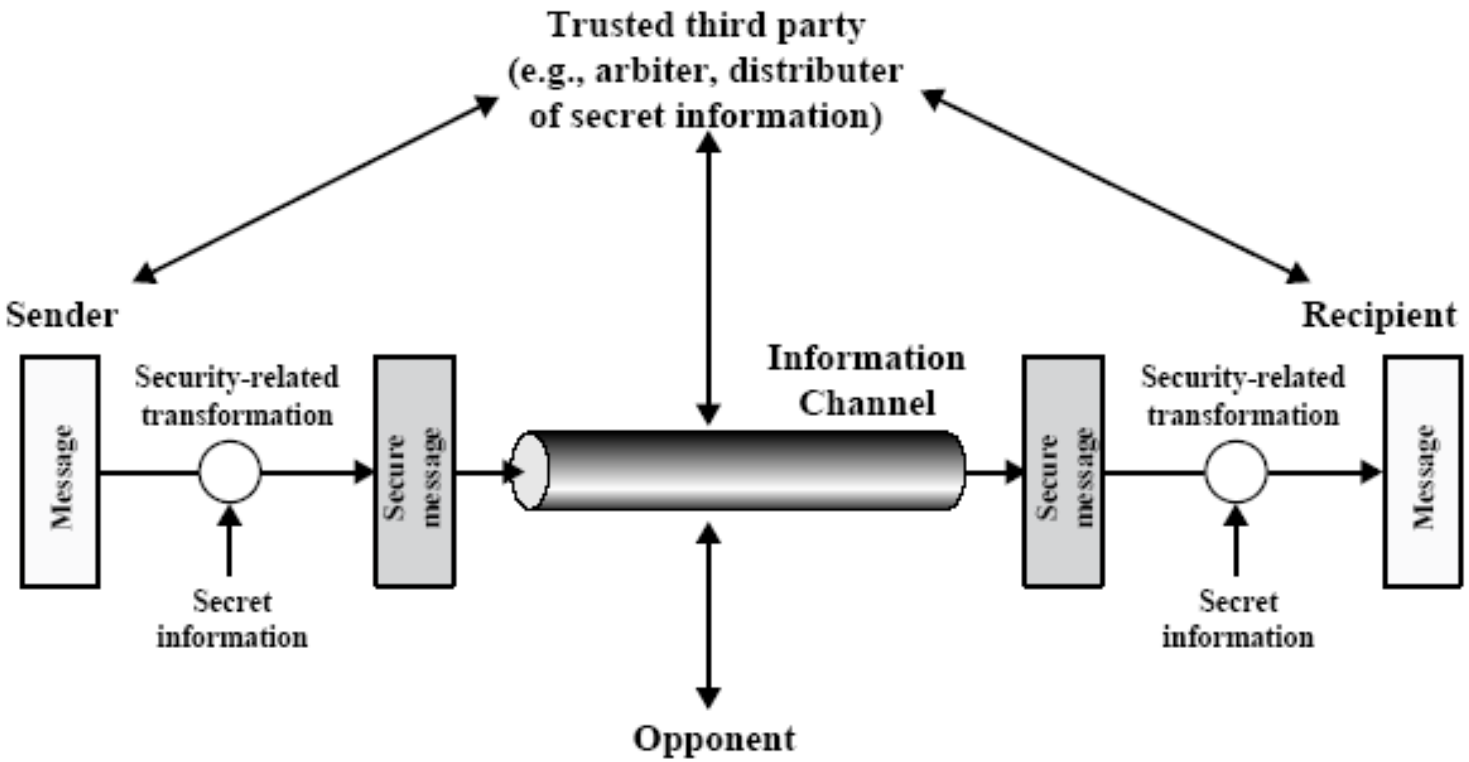
Secure channel: SSH and TLS

Segurança Informática em Redes e Sistemas
2024/25

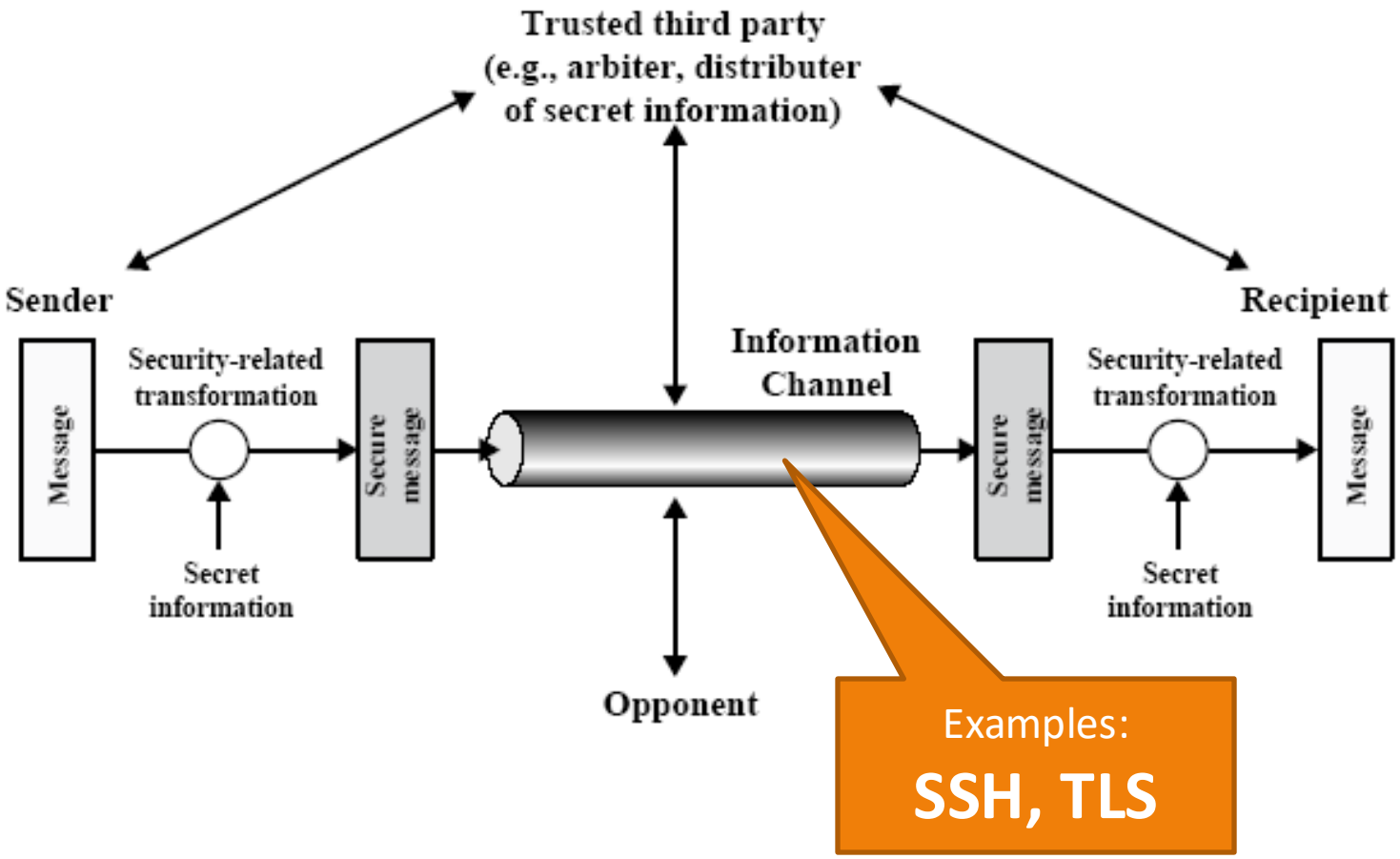
David R. Matos, Ricardo Chaves

Ack: Carlos Ribeiro, André Zúquete,
Miguel P. Correia, Miguel Pardal

Secure communication channel



Secure communication channel



Secure communication: *Transport* layer and above

Layers		Responsibility	Approach	Solutions
	Transaction	Local data manipulation applications		PGP, PEM, S/MIME
OSI Layers	Application	Applications for remote data exchange	End-to-end security	HTTPS, IMAPS **SSH**
	Presentation			
	Session			
	Transport	Operating		**TLS**
	Network	Systems		IPsec
	Link	Devices	Link security	IEEE 802.11*
	Physical			

Attacker model

- Attacker who has full control over the network
  - but cannot break cryptographic primitives
- Attacker can intercept, send, and modify messages
  - but not decrypt them without the key

- This model is called the **Dolev-Yao** model
  - This is the default attacker model presumed in distributed systems
  - Dolev, D., & Yao, A. C. (1983). On the security of public key protocols. IEEE Transactions on Information Theory, 29(2), 198–208. https://doi.org/10.1109/TIT.1983.1056650

Approach: Information Isolation through Cryptography

- Cryptography can render information unintelligible to unauthorized parties
  - Encrypted data is effectively isolated
  - It appears as noise to anyone without the key to decrypt it
- Encrypted information can be transmitted across communication networks or stored in computer systems and stay isolated

Roadmap

- **SSH – Secure Shell**
- TLS – Transport Layer Security

SSH

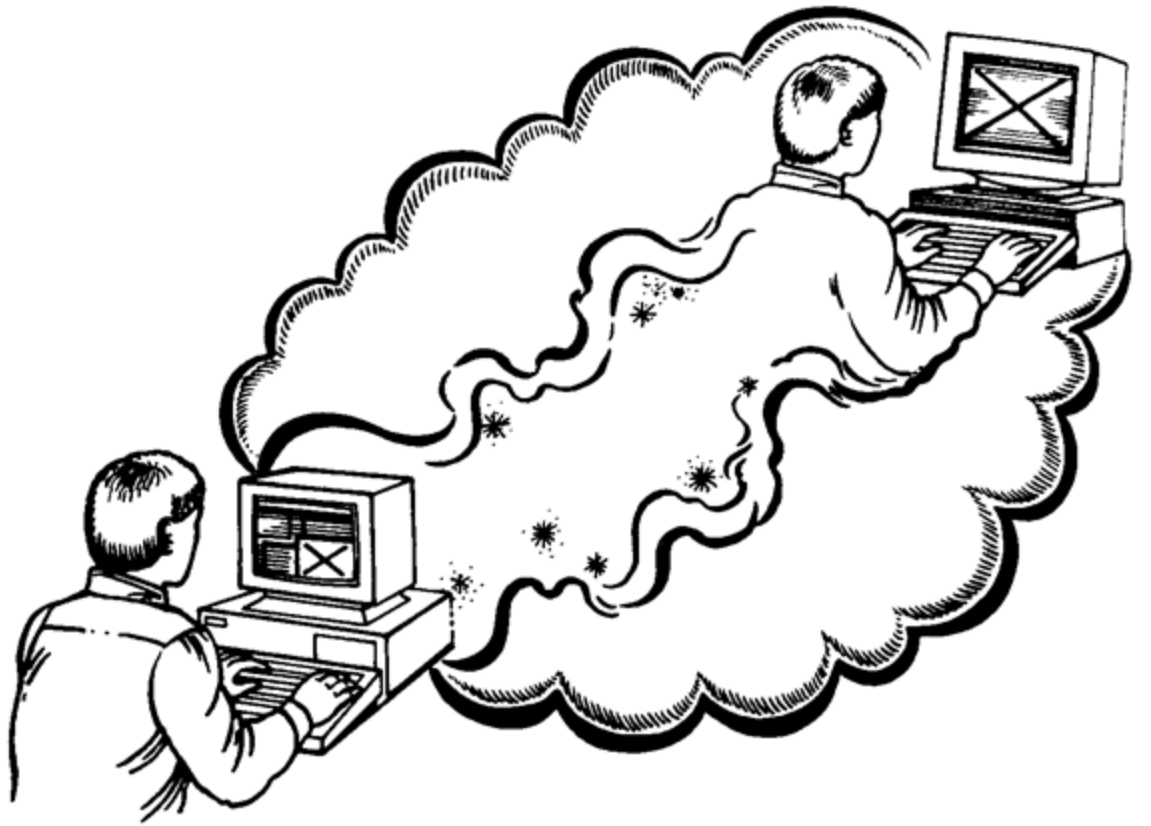


Figure 7.1 Remote login is a lot like astral projection.

SSH goals

- Secure communication application and protocol over TCP
  - Allows **secure remote sessions (~telnet)** and **file transfer (~ftp)**
  - Allows **tunneling** of TCP/IP traffic
- Security mechanisms
  - **Confidentiality** and **integrity** of the communication
  - **Distribution of keys**
  - Communicating parties' **authentication**
    - The server (or, usually, the server machine)
    - The user

SSH Keys

- SSH use **public-key cryptography**
- **Pair of keys**
  - One is **private**, personal, non-transmissible
  - One is **public**, can/should be widely known
- Allow for
  - **Confidentiality** with the exchange of secret keys
  - **Authentication and Integrity** with digital signatures

SSH protocols (1/2)

- Transport Layer Protocol
  - Server authentication
    - Signature of the exchanged DH ephemeral values
    - Server public key can be transferred at this time, if not already held by the client
      - ➔ **TOFU (Trust on First Use) model**
        - Vulnerable to man-in-the-middle
        - No certificates or CA
  - Distribution of keys
    - Diffie-Hellman key exchange
    - The session keys are computed with ephemeral DH values
  - Creation and analysis of secure messages
    - Compression, encryption, integrity control

SSH protocols (2/2)

- User authentication towards the remote machine:
  - Password or
  - Signature with private key of client
    - Server knows the public key of client
- Connection Protocol
  - Information/data flow **multiplexing** over a secure session

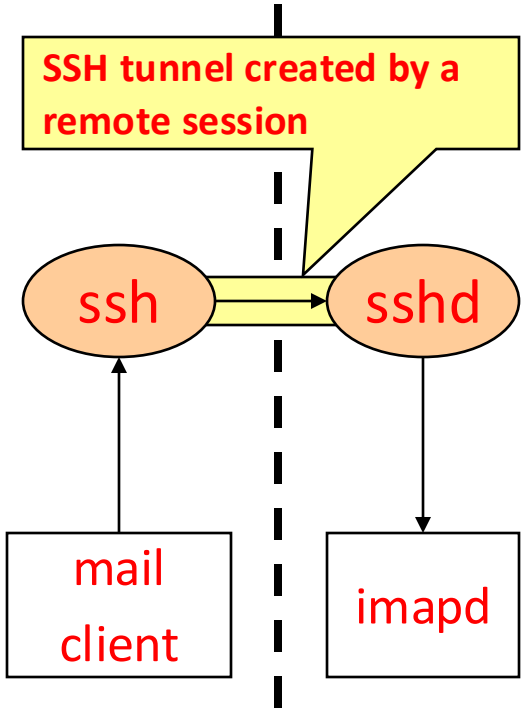
Roadmap

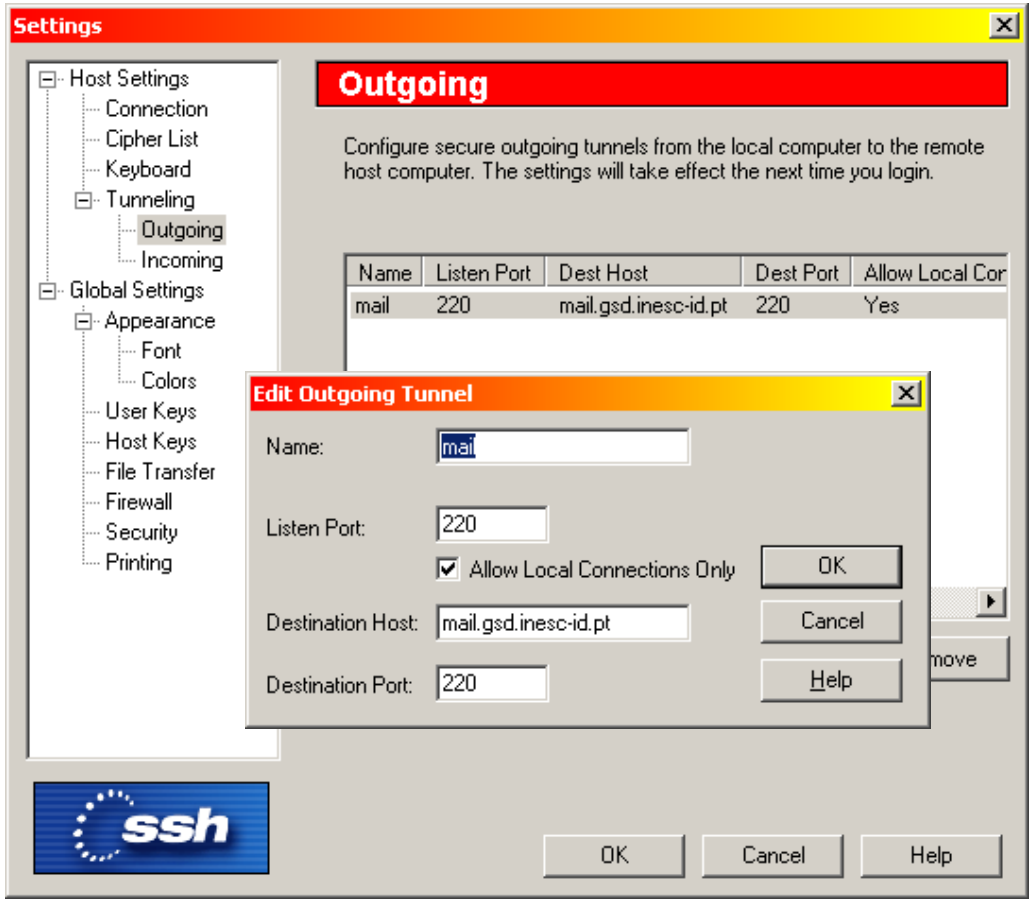
- SSH – Secure Shell
  - **SSH Tunnels**
- TLS – Transport Layer Security

Tunneling with SSH

- In the client machine, a **mapping** is created between a **local TCP port** and a **port in the remote machine**
  - e.g., *localhost:IMAP → mail.myorg.pt:IMAP*
- **SSH session / tunnel** is created to mail.myorg.pt
- Client application is configured to use the **local port**
  - When using the port, it will securely interact via SSH with the remote server *mail.myorg.pt*
  - Client SSH and server SSHd operate as a secure relay mechanism

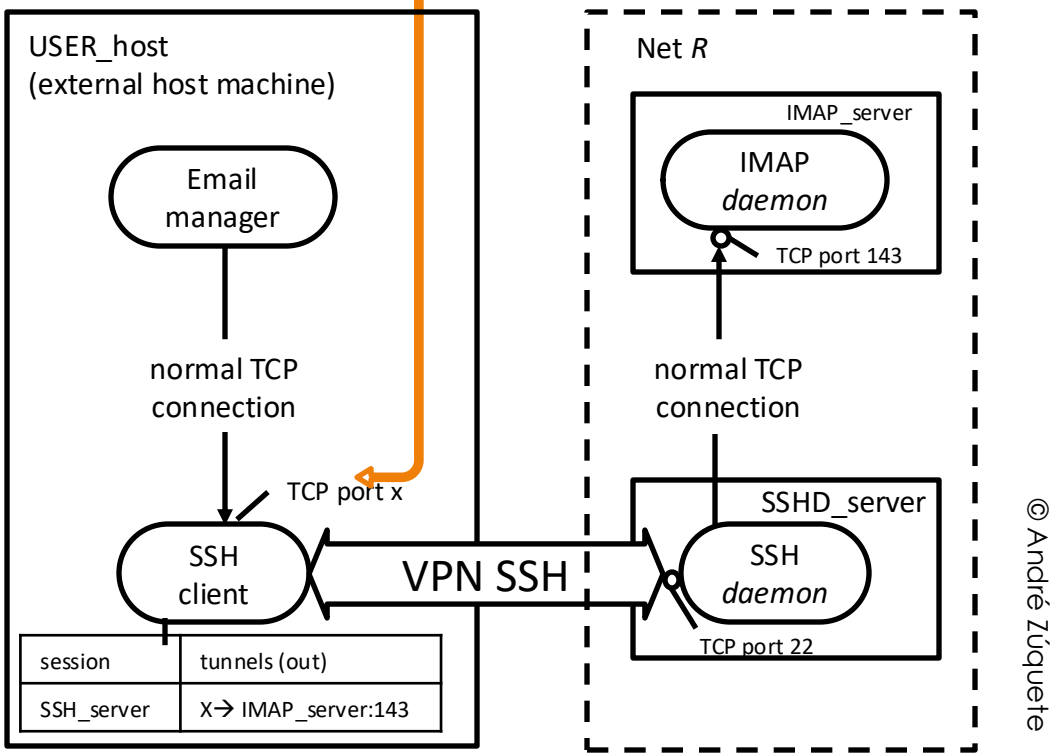
Configuration example of SSH tunnel for IMAP (port 220)





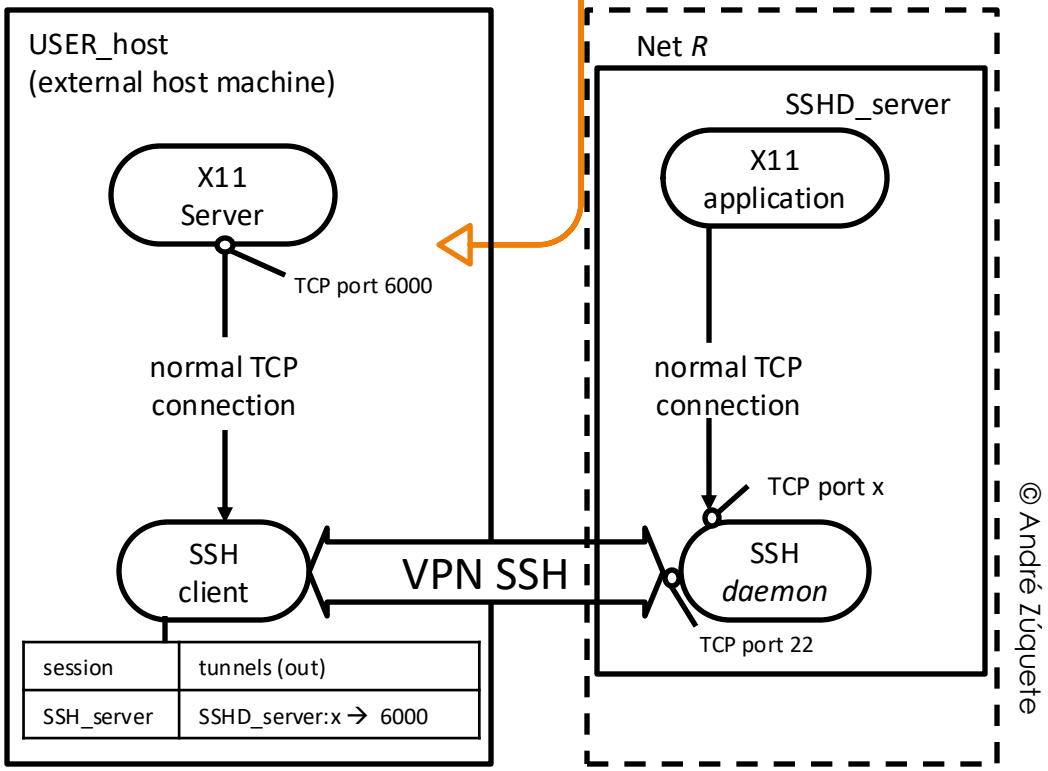
Output tunnel

- TCP connections started at the client, so **outbound**
  - **SSH client** opens **port X** and waits for connections from client application
  - Similar situation to remote login



Input tunnel

- TCP connection started at the server, so **inbound**
  - **SSH client** connects to port in local server
  - Often used to get graphical interface in the remote host



Tunneling

- Transparent for apps
- But complex/difficult for admins
  - Port management
- Low flexibility for developers and users

Roadmap

- SSH – Secure Shell
- **TLS – Transport Layer Security**


http


https

TLS is the S in HTTPS

- **HTTPS** = **HTTP** over **S**SL (now TL**S**)
  - SSL - Secure Sockets Layer
  - TLS – Transport Layer Security
- SSL was originally designed to be used with HTTP
  - Netscape Navigator (1994)
    - Forefather of Mozilla Firefox
  - Evolved from SSL in 1994 to TLS 1.3 by 2018
    - SSL 1.0 1994
    - SSL 2.0 1995
    - SSL 3.0 1996
    - TLS 1.0 1999
    - TLS 1.1 2006
    - TLS 1.2 2008
    - TLS 1.3 2018

TLS overview

- TLS provides secure channels over TCP/IP, authentication, integrity, confidentiality, and key distribution
- Many implementations:
  - SSLref, OpenSSL, GnuTLS, SSLeay, Mbed TLS, JSSE (Java Secure Socket Extension)
- TLS is not bound to HTTP
  - Can be used with: SMTP, IMAP, POP3
- TLS is very significant, as it ensures data in transit security across many critical Internet services

TLS goals

- Secure communication channels over TCP/IP
  - Current version: **TLS 1.3** – august 2018
    - Standard based on the deprecated **SSL (Secure Sockets Layer)**
    - Sometimes called **SSL** for that reason
  - Manages secure sessions over **TCP/IP** per application
    - Initially designed for the HTTP protocol (HTTPS)
    - Currently used by other protocols, e.g., SMTP, IMAP, POP3
- Security mechanisms
  - **Authentication** of the communicating parties
  - **Confidentiality** and **integrity** of the communication
  - **Key distribution**

TLS utilization

- TLS is just a protocol; not a standard API
- Common APIs:
  - Reference API for SSL: SSLref (Netscape)
  - Public implementations: OpenSSL, GnuTLS, SSLeay, Mbed TLS, Java Secure Socket Extension (JSSE)
- Remote interfaces:
  - Conventional: protocol/port
    - e.g., TCP/443 for HTTPS
  - STARTTLS: allows upgrading text connection to TLS
    - Defined for SMTP, IMAP, POP, SMTP, FTP, IRC,…

TLS operation management

- Client-Server model as in TCP
- The applications (e.g., web, mail) define the strategy
- Authentication
  - If it is needed and how it is performed
- Cryptographic algorithms
  - The client presents the **cipher suites** it supports
  - The server selects one
- Session key management
  - Lifetime of the **master secret** = lifetime of the **TLS session**
    - Used whenever the client and the server decide to communicate
    - Maximum of 24 hours recommended
  - Lifetime of the **session keys**: at most lifetime of the TCP connection

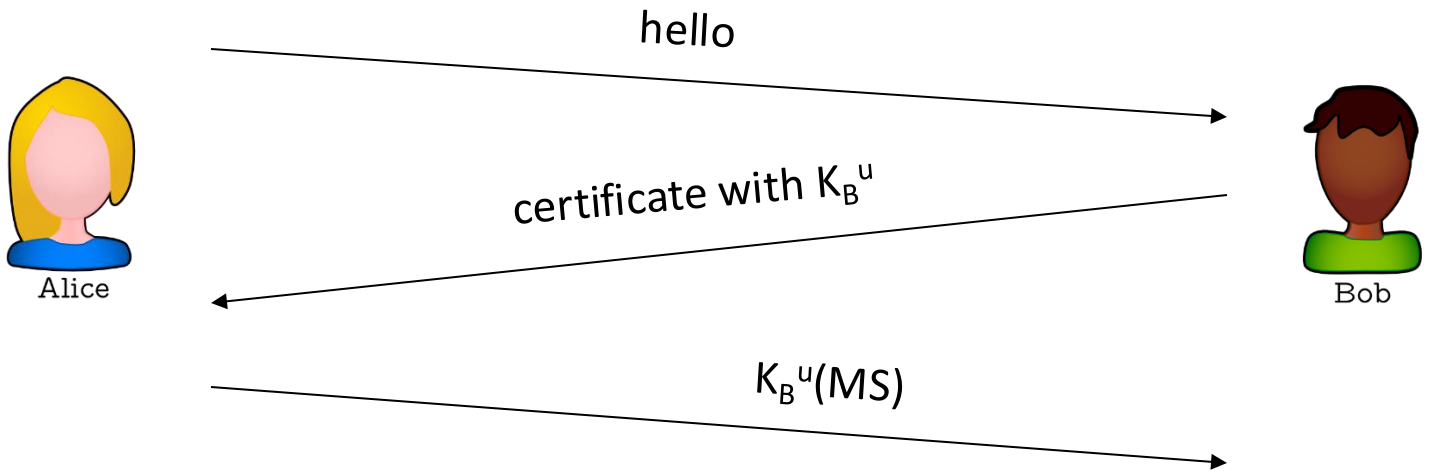
Roadmap

- SSH – Secure Shell
- TLS – Transport Layer Security
  - **Toy TLS**

Toy TLS: simplified secure channel

- **Handshake**
  - Alice and Bob use their certificates, private keys to authenticate each other and exchange a shared secret
- **Key derivation**
  - Alice and Bob use shared secret to derive set of keys
- **Data transfer**
  - Data to be transferred is broken up into series of records
- **Connection closure**
  - Special messages to securely close connection

Toy TLS: a simple handshake



- K_B^u = Bob's public key
- MS = master secret

Toy TLS: key derivation

- It is bad to use same the key for more than 1 cryptographic operation
  - So, use different keys for MACs and encryption
- Therefore **4 keys**:
  - K_c = encryption key for data sent from client to server
  - M_c = MAC key for data sent from client to server
  - K_s = encryption key for data sent from server to client
  - M_s = MAC key for data sent from server to client
- Keys derived using **key derivation function (KDF)**
  - Takes master secret (MS) and additional random data and creates the keys

Toy TLS: data records

- Why not encrypt data in stream as we write it to TCP?
  - Where would we put the MAC? If at end, no message integrity until the end of the connection!
- Instead, break stream in series of **records** (≃messages)
  - Each record carries a MAC
  - Receiver can act on each record as it arrives
- Issue: records have variable length and receiver needs to distinguish MAC from data
  - Solution:

record

Toy TLS: sequence numbers

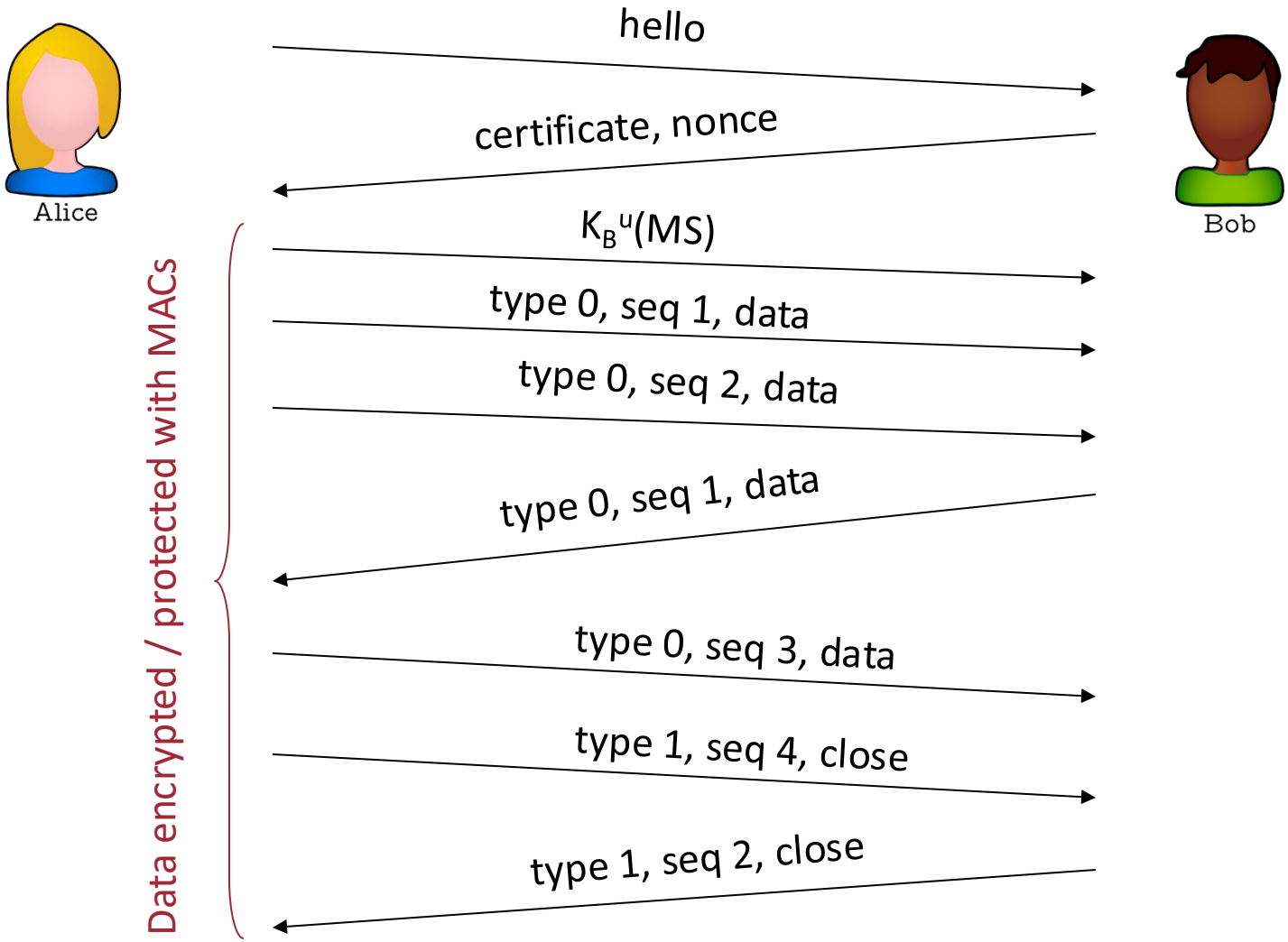
- Attacker can replay or re-order records
- Solution: put sequence number (**seq**) into MAC:
  - **seq** starts at 0; incremented for every record
  - MAC = MAC(M_x, seq||data) *key M_x= M_c or M_s*
  - Important: there is no sequence number field in the record, so **seq** is implicit, just like the **key** M_x
- Attacker could still replay all of the records
  - Use a **nonce** to prevent this attack

Toy TLS: control information

- Truncation attack
  - Attacker forges TCP connection close segment
  - One or both sides thinks there is less data than there actually is
- Solution: **record types**, with a type for closure
  - type 0 for data; type 1 for closure
- MAC = MAC(M_x, seq||type||data)

record

Toy TLS: summary



Toy TLS is not complete

- How long are the fields?
- Which encryption protocols to use?
- Negotiation?
  - Allow client and server to support different encryption algorithms
  - Allow client and server to choose together specific algorithm before data transfer

Roadmap

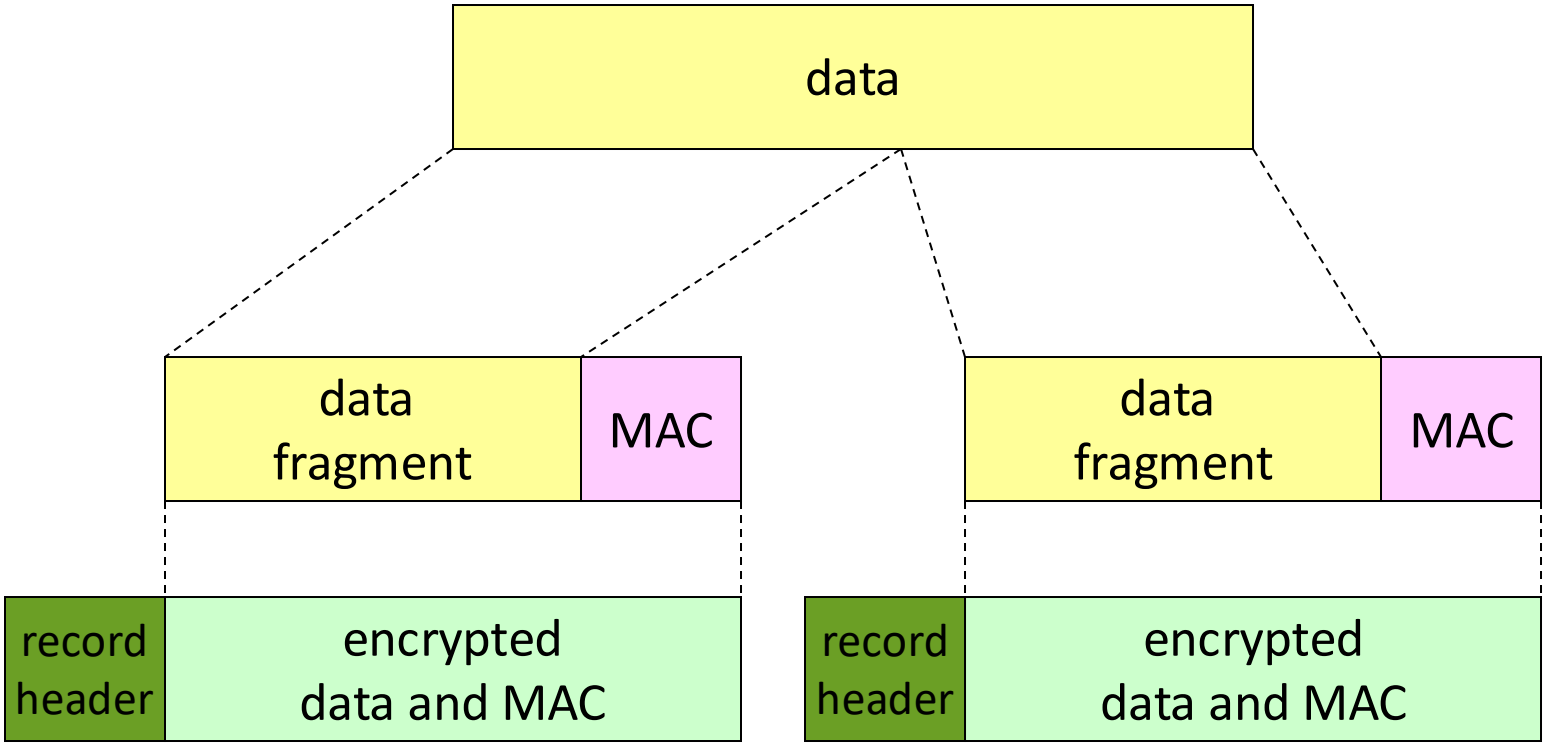
- SSH – Secure Shell
- **TLS – Transport Layer Security**

TLS protocols

- **Record Protocol**
  - Creation and verification of secure messages
    - Cipher, integrity control, ~~compression~~
- **Handshake Protocol**
  - Exchange of identity and supported cipher suites
  - Authentication of the communicating parties
    - Client challenges the server, that proves its identity with certificate(s) *[Optional, but mandatory if the next exists]*
    - Server challenges the client, that proves its identity with certificates *[Optional]*
  - Key distribution

Removed in TLS 1.3
(to avoid some attacks and because it was not widely used)

TLS record protocol

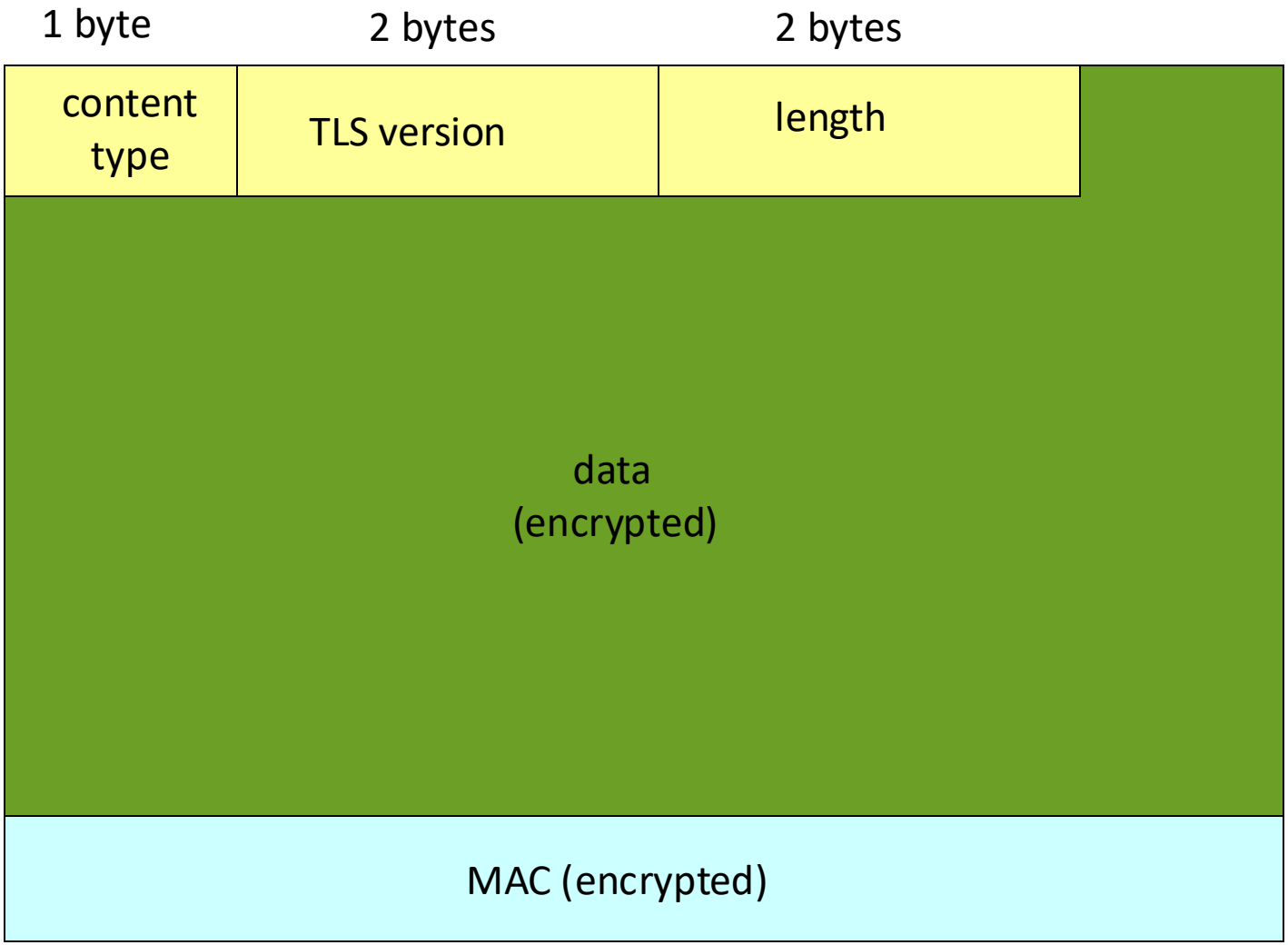


Record header: content type; TLS version; length

MAC: is function of the sequence number and the MAC key M_x

Fragment: each data fragment has up to 2^{14} bytes (~16 Kbytes)

TLS record format



TLS handshake (1)

Purpose

1. Server authentication
2. Negotiation: agree on crypto algorithms
3. Establish keys
4. Client authentication (optional)

TLS handshake (2)

1. **Client** sends list of algorithms it supports, along with client nonce
2. **Server** chooses algorithms from list; sends back: choice + certificate + server nonce
3. **Client** verifies certificate, extracts server's public key, generates pre_master_secret, encrypts with server's public key, sends to server
4. **Client and server** independently compute encryption and MAC keys from pre_master_secret and nonces
5. **Client** sends a MAC of all the handshake messages
6. **Server** sends a MAC of all the handshake messages

TLS Cipher Suites

- **Cipher suite**
  - Public-key algorithm
  - Symmetric encryption algo.
  - MAC algorithm
- TLS supports several
- **Negotiation**:
  - Client offers choice
  - Server picks 1
    - e.g., the most secure

- **Common algorithms**
- Public-key encryption
  - RSA, ECDSA
- Symmetric ciphers
  - AES: block
  - ChaCha20: stream
- Hash functions
  - SHA-2
  - SHA-3

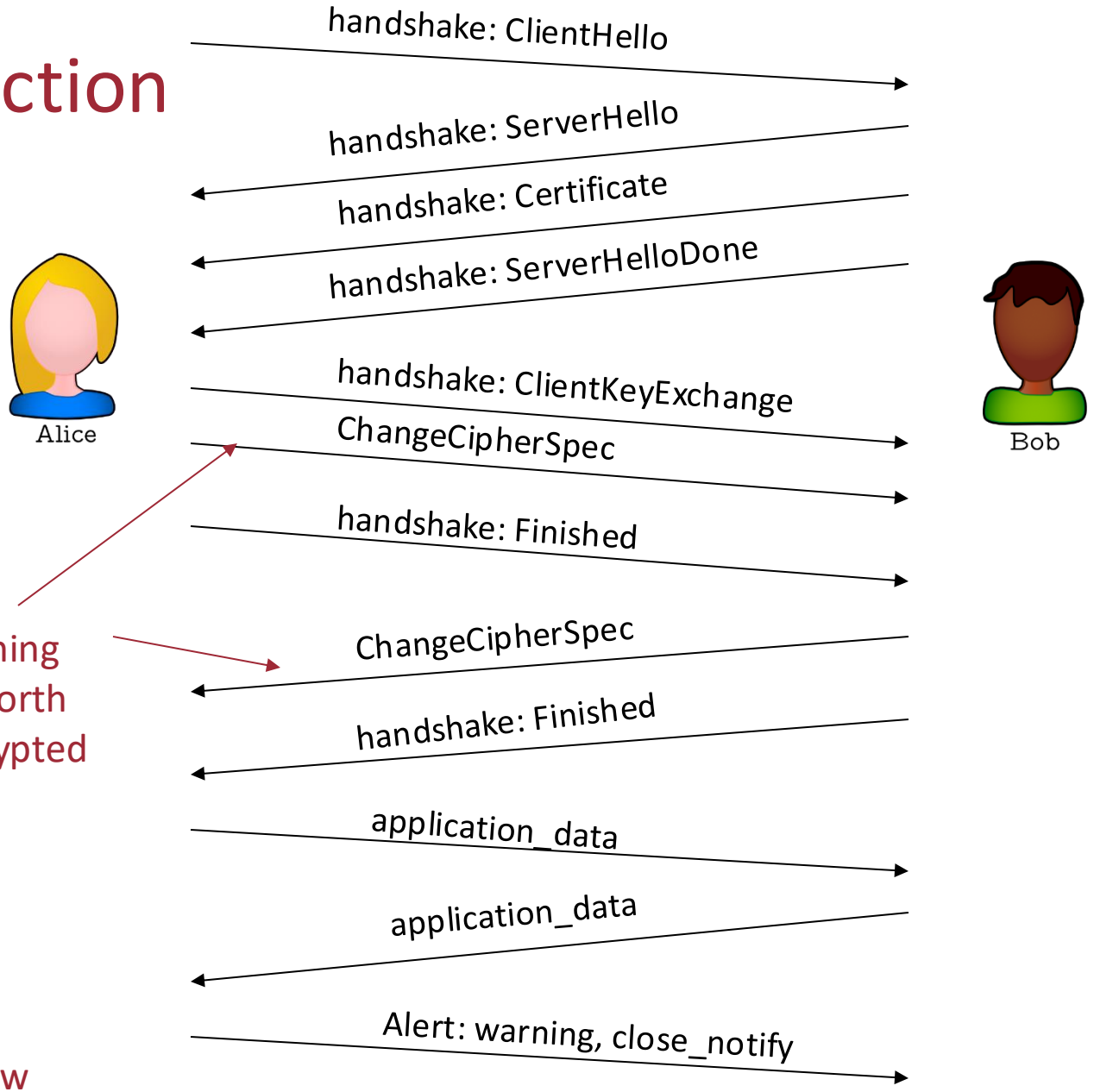
TLS handshake (3)

- Last two steps protect handshake from tampering:
  - Client typically offers range of algorithms, some strong, some weak
  - Man-in-the-middle could delete best algorithms from list
  - Last two message MACs allows detecting such an attack
- Why not simply sign or add MACs to the handshake messages?
  - Not possible: it is the handshake that sets up the keys!

TLS handshake (4)

- Why 2 nonces? (steps 1 and 2 of the handshake)
- Suppose Trudy sniffs all messages between Alice & Bob
- Next day, Trudy sets up TLS connection with Bob (server), sends exact same sequence of records
  - Bob (e.g., Amazon) thinks Alice made two separate orders for the same thing
  - Solution: Bob sends different nonce for each connection. This causes encryption keys to be different on the two days
  - Trudy's messages will fail Bob's integrity check

Real connection



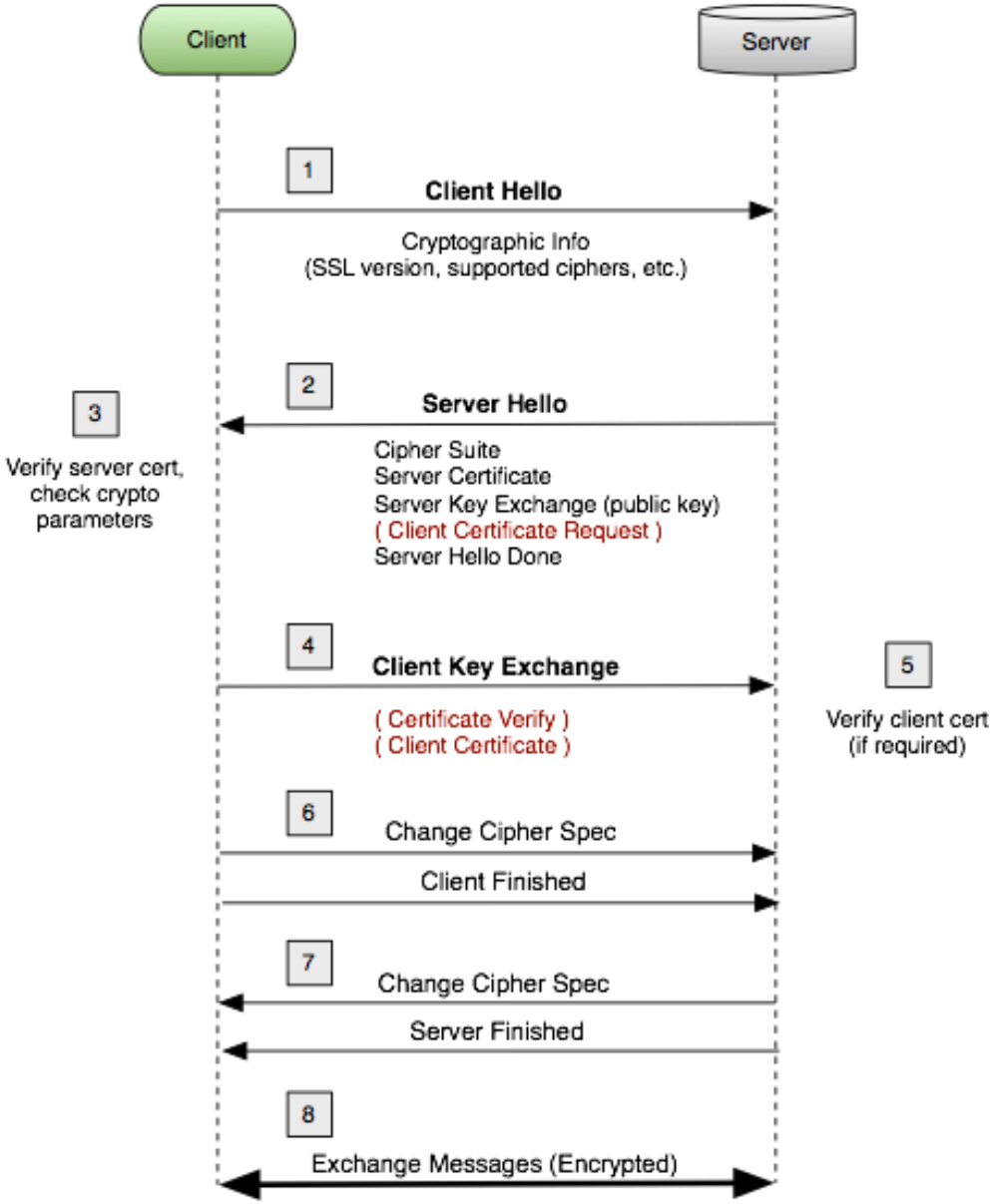
Key derivation

- Client nonce, server nonce, and pre-master secret input into pseudo random-number generator, produce master secret (MS)
  - pseudo random-number generator => same input at client and server produces the same result
- Master secret and nonces input into another pseudo random-number generator, produce key block, cut into:
  - Client MAC key
  - Server MAC key
  - Client encryption key
  - Server encryption key
  - Client initialization vector (IV)
  - Server initialization vector (IV)

TLS secrets and keys

- Master secrets (48 octets)
  - Pre-Master Secret (PMS)
    - Computed with DH; or
    - PMS is chosen by the client and sent to the server ciphered with its public key
  - Master secret (MS)
    - MS = MD5(PMS | SHA('A' | PMS | R1 | R2)) |
      MD5(PMS | SHA('BB' | PMS | R1 | R2)) |
      MD5(PMS | SHA('CCC' | PMS | R1 | R2)) | …
- Session keys
  - Computed from a master secret and the random values exchanged in the protocol
    - ClientMacKey = MD5(MS | SHA('A' | MS | R1 | R2))
    - ServerMacKey = MD5(MS | SHA('BB' | MS | R1 | R2))
    - ClientKey = MD5(MS | SHA('CCC' | MS | R1 | R2))
    - ServerKey = MD5(MS | SHA('DDDD' | MS | R1 | R2))

TLS


Client
Server
1
Client Hello
Cryptographic Info
(SSL version, supported ciphers, etc.)
2
Server Hello
Cipher Suite
Server Certificate
Server Key Exchange (public key)
(Client Certificate Request)
Server Hello Done
3
Verify server cert,
check crypto
parameters
4
Client Key Exchange
(Certificate Verify)
(Client Certificate)
5
Verify client cert
(if required)
6
Change Cipher Spec
Client Finished
7
Change Cipher Spec
Server Finished
8
Exchange Messages (Encrypted)

Example of client and server “talking”

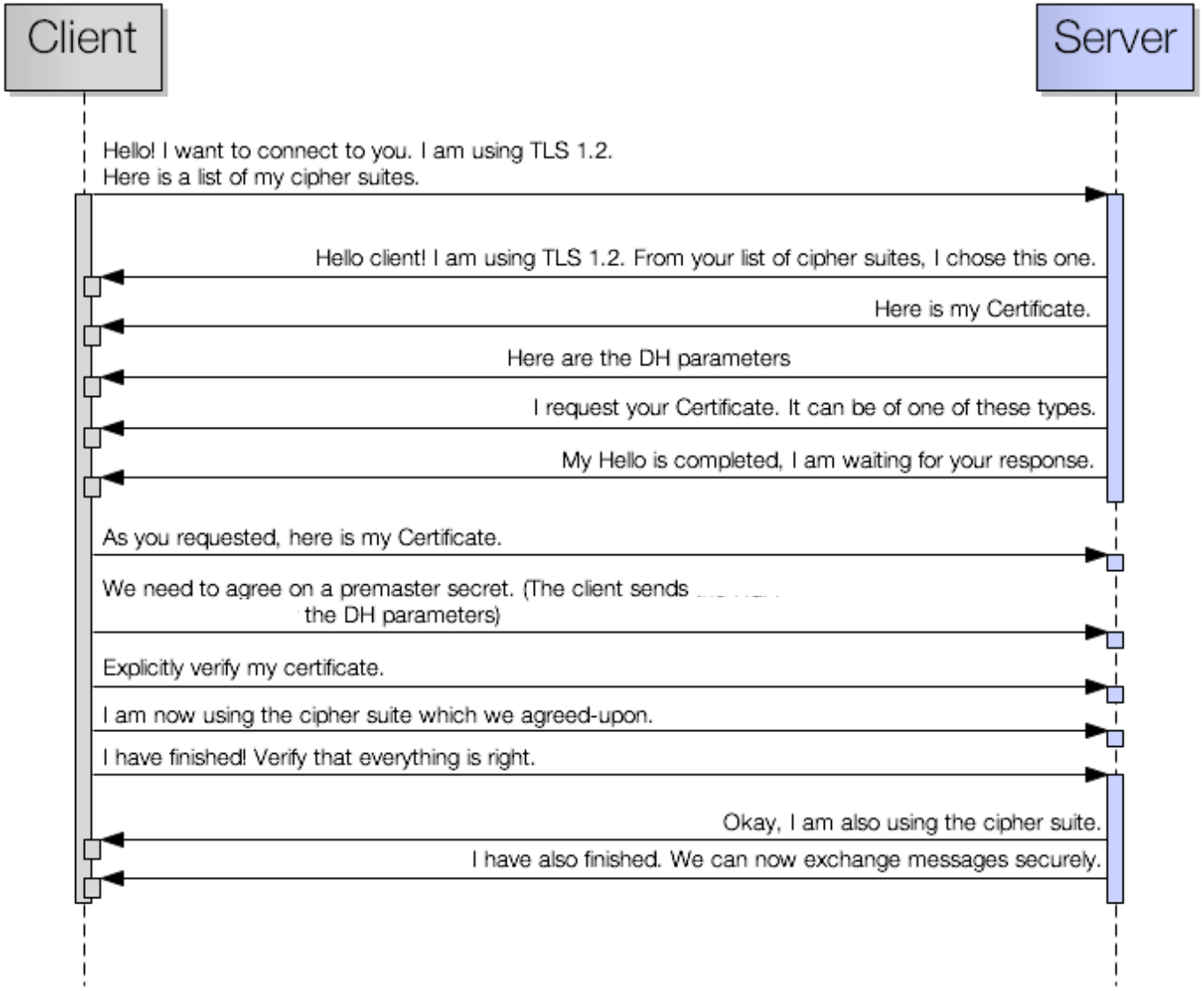


Figure credits: André Joaquim

TLS performance

- First implementations were slow
  - TLS was only used for some sensitive connections
- Currently, when properly configured, TLS can be fast
  - *"On our production frontend machines,*
    TLS accounts for
    less than 1% of the CPU load,
    less than 10 KB of memory per connection and
    less than 2% of network overhead."
    - Adam Langley, Google "Overclocking SSL"
      https://istlsfastyet.com/
- Nowadays, TLS can be the default for all connections

Roadmap

- SSH – Secure Shell
- TLS – Transport Layer Security
  - **Key management**

Key management

- TLS and SSH use **public-key cryptography**
  - (We will study this cryptography later, in more detail)
- **Pair of keys**
  - One is **private**, personal, non-transmissible
  - One is **public**, can/should be widely known
- Allow for
  - **Confidentiality** with the exchange of secret keys
  - **Authentication and Integrity** with digital signatures

Private key

- The private key represents its owner so:
  - The probability of it being compromised must be minimized
  - Backup copies must be physically secure
- Private key must be protected
  - The access path to the private key must be restricted
  - Password protected, e.g., JKS, PGP
  - Security of applications using the private key must be guaranteed

Private key confinement

- Storage and use of the private key in an autonomous device, e.g., a smartcard
  - The device generate the key pairs
  - The device ciphers/deciphers the data with the key pair controlled by on-chip access mechanisms
    - e.g., access PIN
  - Allows for qualified signatures
    - EU eIDAS Regulation

Public key certificate

- **Certificates** are documents signed by a certification entity
  - **Certification Authority (CA)**, public organization or company
  - Certificates are public documents
  - Certificates have a digital signature to assure authenticity
- Can distribute public key through unsecure channel
  - Receiver can validate the certificate signature using the CA public key
  - If it trusts the CA and the signature is valid, then it can trust the public key
- Certificate standard format: X.509 (RFC 3280)

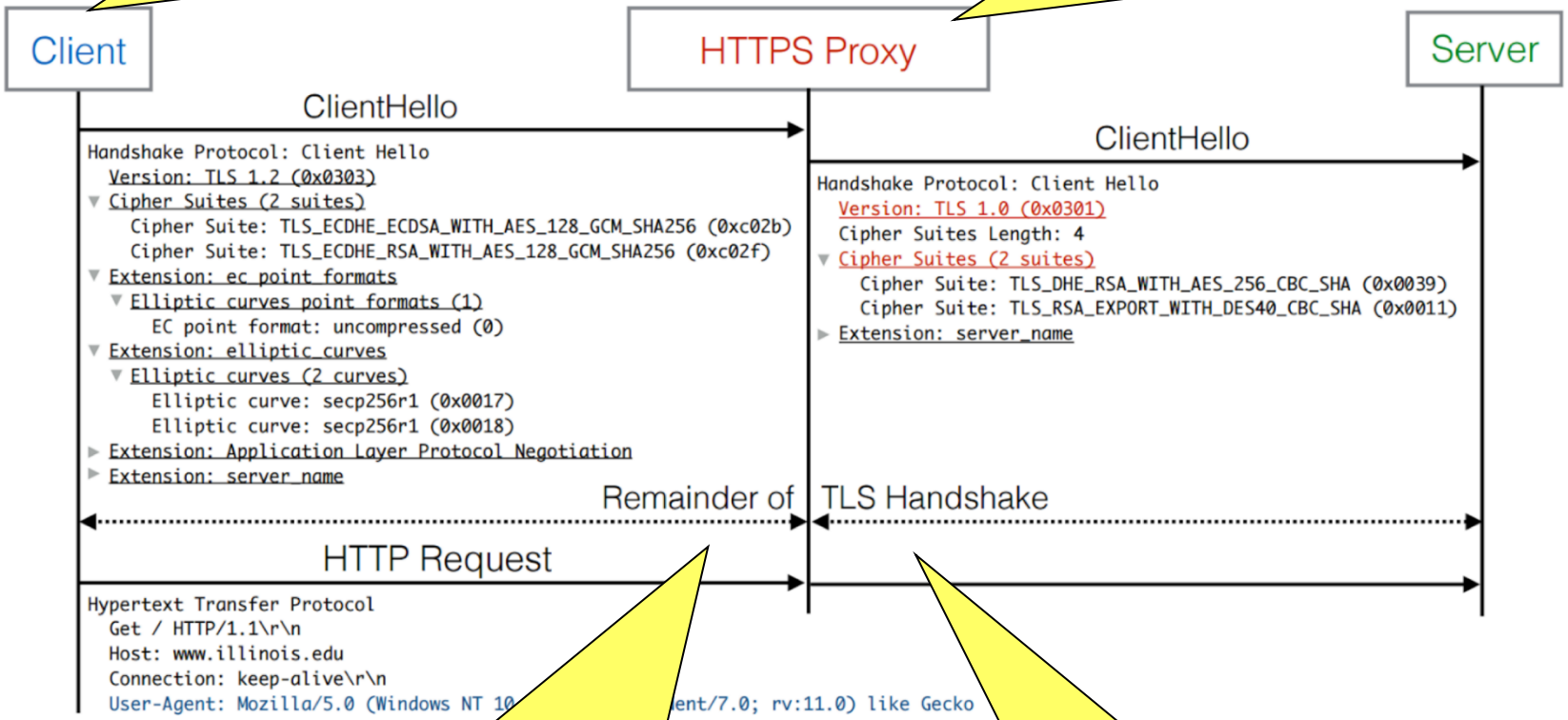
Roadmap

- SSH – Secure Shell
- TLS – Transport Layer Security
  - **Interception attacks**

TLS interception attack

Client is forced (or misled) to install a CA certificate controlled by the proxy

Proxy generates a certificate for the server "on the fly"



TLS session between client and proxy (client does not see any difference because it trusts the certificate generated by the malicious CA)

TLS session between proxy and server

TLS Fingerprints

Fingerprints

Is your employer, school, or Internet provider

eavesdropping on your **secure** connections?

370 sets of fingerprints checked per day
2,370,158 sets of fingerprints checked for our visitors

Secure browser connections can be intercepted and decrypted by authorities who spoof the authentic site's certificate. But **the authentic site's fingerprint CANNOT be duplicated!**

Domain Name	Certificate Name	EV	Security Certificate's **Authentic** Fingerprint
www.grc.com	grc.com	–	7A:85:1C:F0:F6:9F:D0:CC:EA:EA:9A:88:01:96:BF:79:8C:E1:A8:33
www.facebook.com	*.facebook.com	–	43:29:AB:C9:18:BB:BB:D5:B1:FC:8E:97:26:59:14:EB:92:B1:12:0D
www.paypal.com	www.paypal.com	●	BD:DE:B3:95:6B:86:79:B8:27:86:DA:4D:75:3C:53:AA:04:1E:08:92
twitter.com	twitter.com	–	73:33:BB:96:1D:DB:9C:0C:4F:E5:1C:FF:68:26:CF:5E:3F:50:AB:96
www.blogger.com	*.blogger.com	–	4D:8C:20:14:4A:3D:BB:CC:1E:62:36:9B:19:1A:3C:CF:E0:B4:CB:F1
www.linkedin.com	www.linkedin.com	–	2C:5C:AD:EB:6F:F3:9F:15:48:61:84:43:29:9C:B5:74:5A:BA:6E:A4
www.yahoo.com	*.www.yahoo.com	–	AD:D7:73:36:32:68:AB:33:71:3E:6E:1D:1B:73:93:1A:F4:2B:DC:D7
wordpress.com	*.wordpress.com	–	7A:C1:B2:7E:09:FF:88:03:C3:E9:B7:4F:31:F4:AC:75:79:BA:66:E6
www.wordpress.com	*.wordpress.com	–	7A:C1:B2:7E:09:FF:88:03:C3:E9:B7:4F:31:F4:AC:75:79:BA:66:E6

Each site's authentic security certificate fingerprint (shown above) was just now obtained by GRC's servers from each target web server. If your web browser sees a different fingerprint for the same certificate (carefully verify the Certificate Name is identical) that forms strong evidence that something is intercepting your web browser's secure connections and is creating fraudulent site certificates.

https://www.grc.com/fingerprints.htm

Summary

- SSH – Secure Shell
  - SSH Tunnels
- TLS – Transport Layer Security
  - Toy TLS
  - TLS record and handshake protocols
  - Key management
  - Interception attacks