TÉCNICO LISBOA

TLS in depth

Segurança Informática em Redes e Sistemas
2022/23

Miguel Pardal

Ack: Carlos Ribeiro, André Zúquete,
Miguel P. Correia, Ricardo Chaves

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			

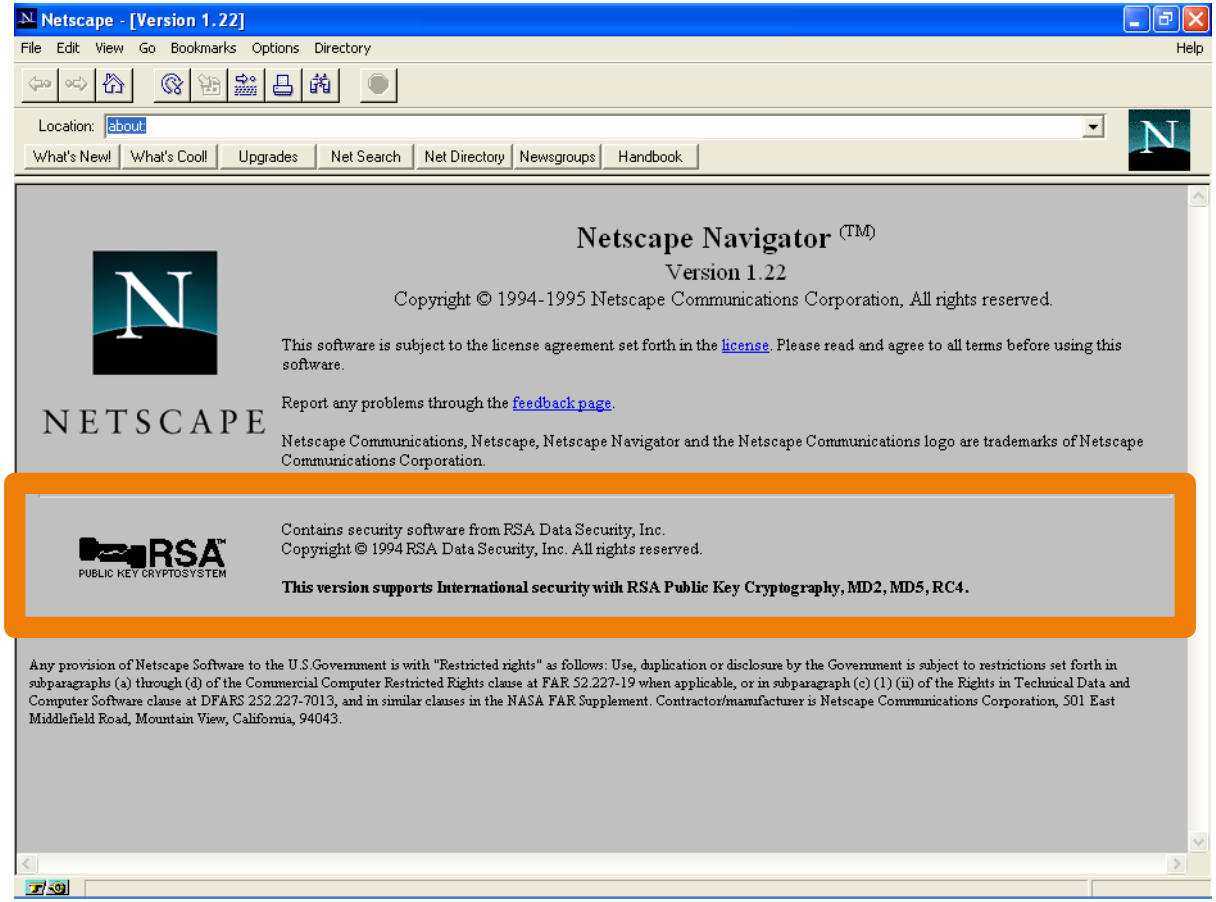
Roadmap

- TLS – Transport Layer Security
  - **HTTPS**

HTTPS

- **HTTPS** = **HTTP** over **S**SL (now TL**S**)
  - In fact, SSL was originally designed to be used with HTTP
    - Netscape Navigator (1994)
      - Forefather of Mozilla Firefox
  - Version history
    - 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

Netscape Navigator



- A few months later, in July 1995, a computer programmer ordered the first book ever sold by an online bookstore named after a river in South America…

Secure communication: *Transport* layer

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

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

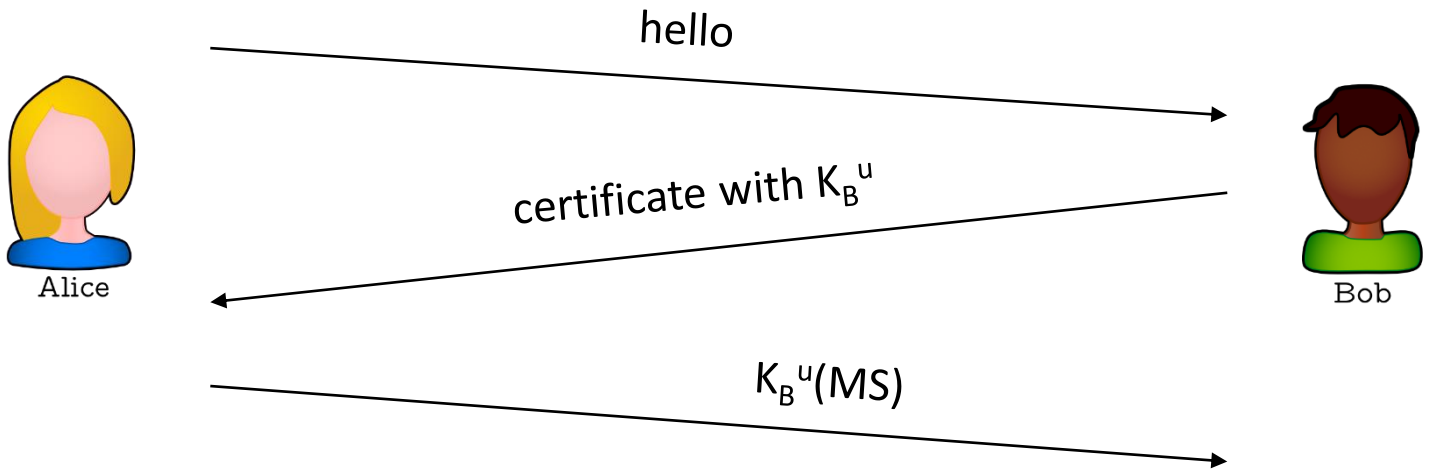
Roadmap

- 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

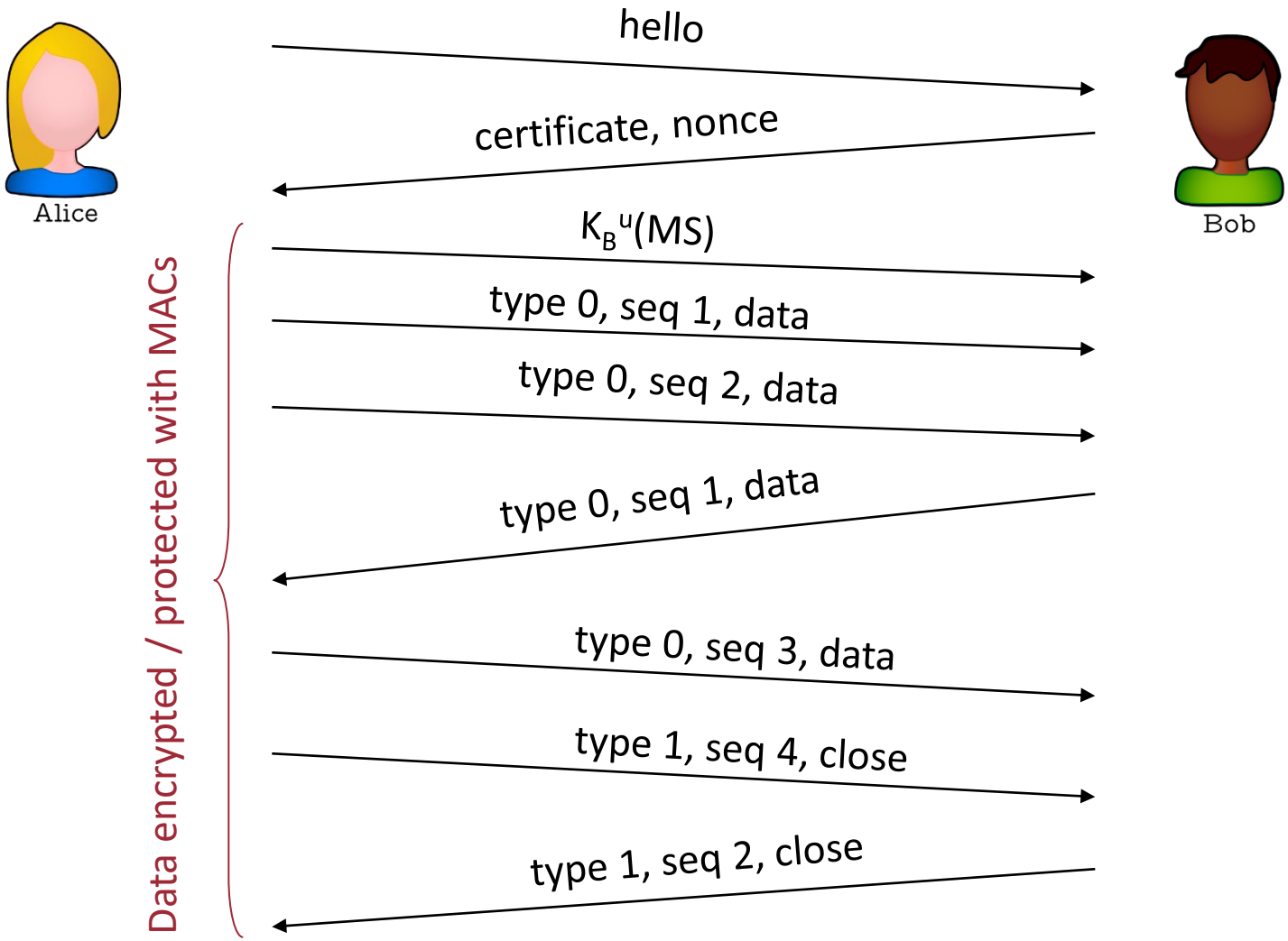
- Attacker can replay or re-order records
- Solution: put sequence number (**seq**) into MAC:
  - **seq** starts at 0; incremented for every record
  - MAC = MAC(M_x, seq||data) *key M_x= M_c or M_s*
  - Important: there is no sequence number field in the record, so **seq** is implicit, just like the **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 fields?
- Which encryption protocols?
- Negotiation?
  - Allow client and server to support different encryption algorithms
  - Allow client and server to choose together specific algorithm before data transfer

Roadmap

- **TLS – Transport Layer Security**

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 protocols

- **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
- **Record Protocol**
  - Creation and verification of secure messages
    - ~~Compression~~, cipher, integrity control

Removed in TLS 1.3

TLS handshake (1)

Purpose

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

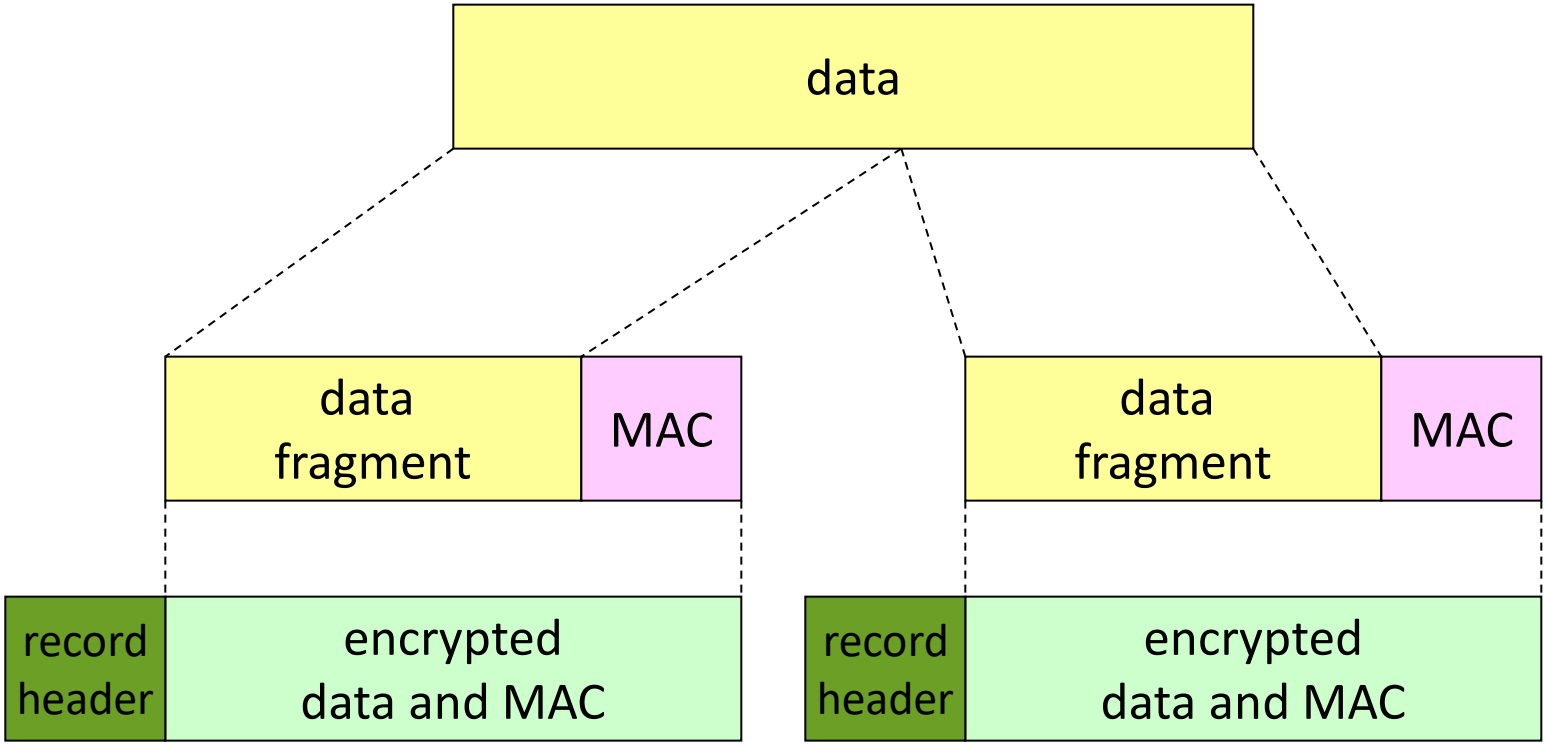
TLS handshake (2)

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 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 record protocol

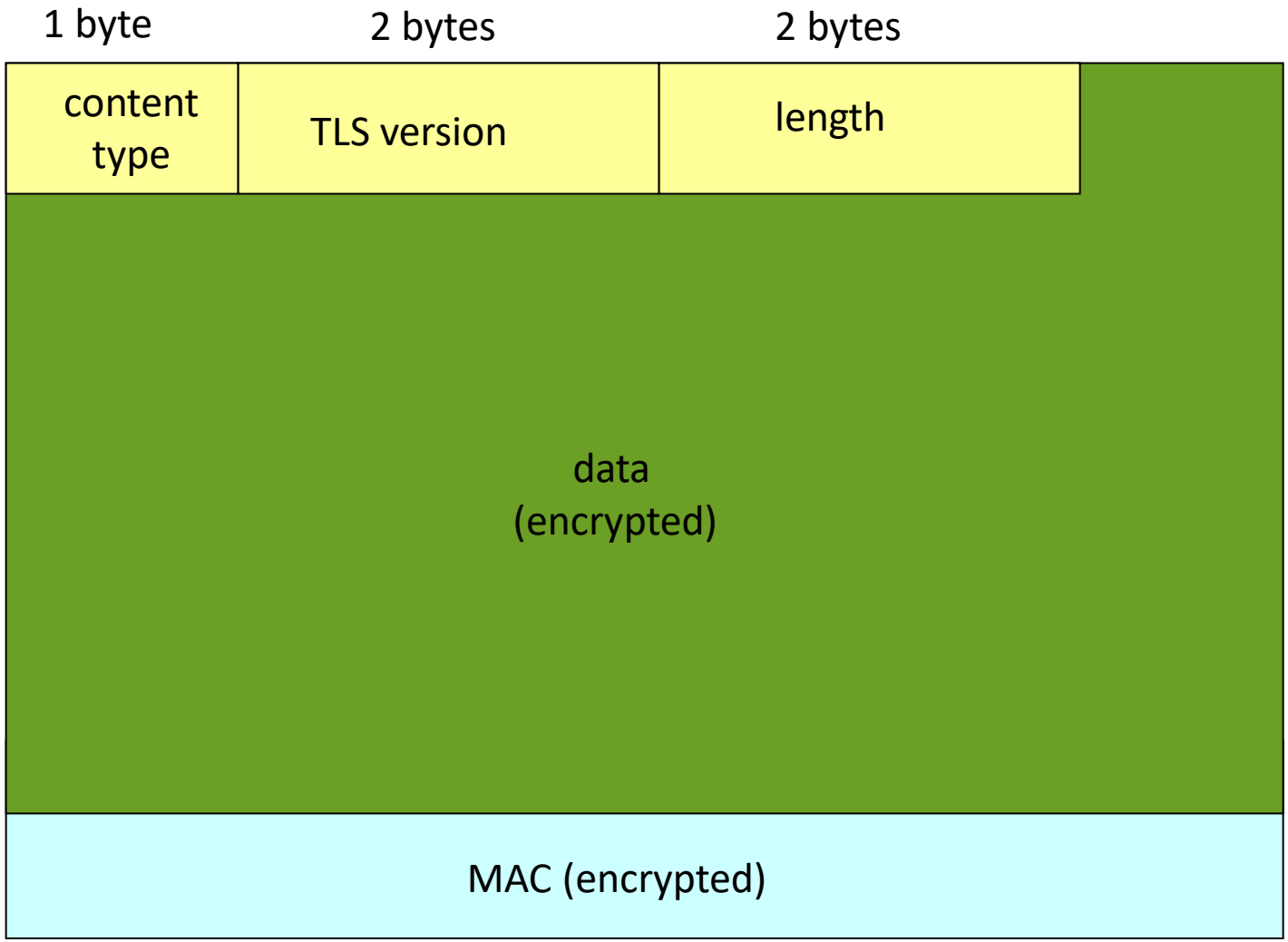


Record header: content type; TLS version; length

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 key distribution

- 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))

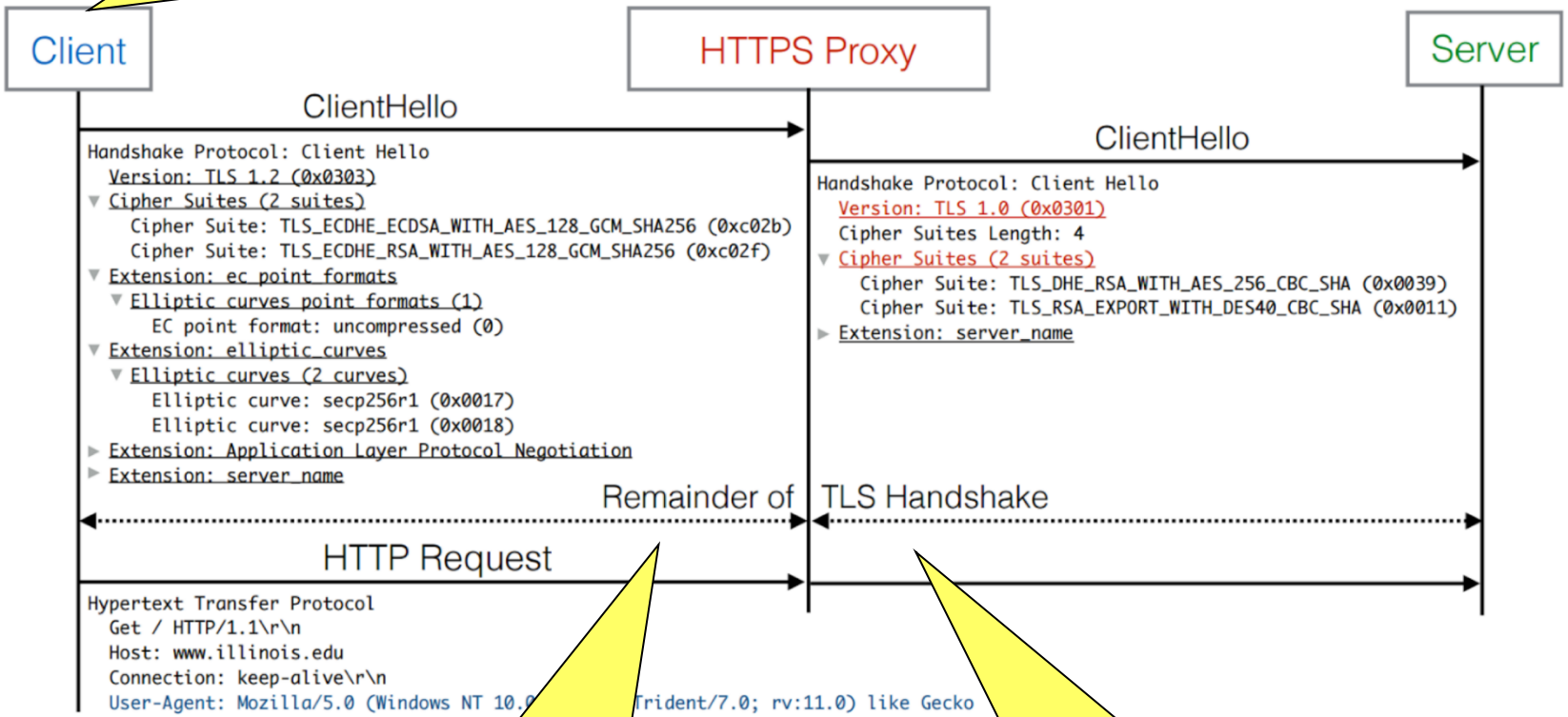
TLS performance

- First implementations were slow
- Currently, when properly configured, 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/

TLS interception attack

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



TLS session between client and proxy (client does not see any difference)

TLS session between proxy and server

TLS tool: 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!**

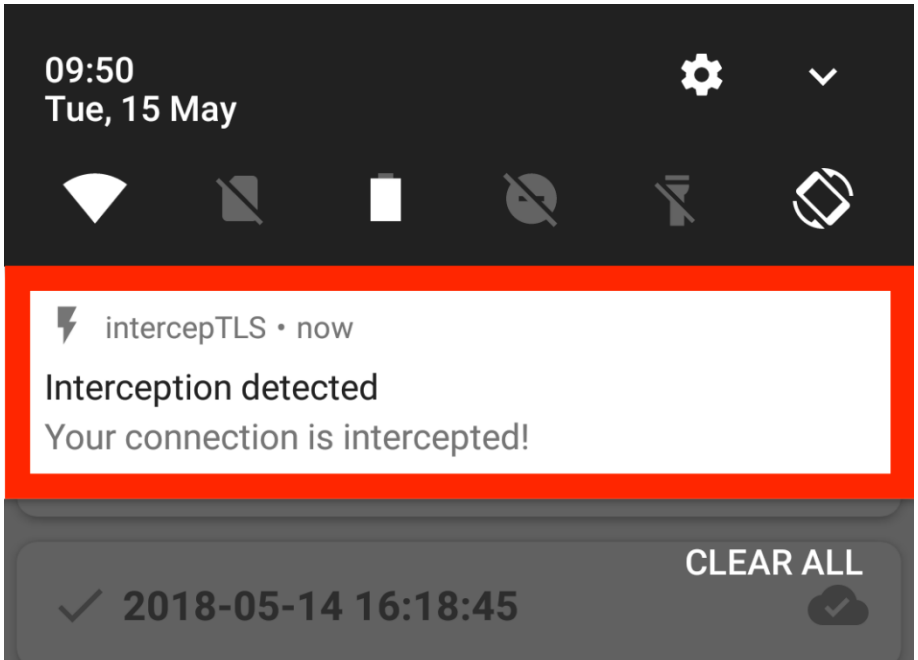
Domain Name	Certificate Name	EV	Security Certificate's **Authentic** Fingerprint
www.grc.com	grc.com	–	7A:85:1C:F0:F6:9F:D0:CC:EA:EA:9A:88:01:96:BF:79:8C:E1:A8:33
www.facebook.com	*.facebook.com	–	43:29:AB:C9:18:BB:BB:D5:B1:FC:8E:97:26:59:14:EB:92:B1:12:0D
www.paypal.com	www.paypal.com	●	BD:DE:B3:95:6B:86:79:B8:27:86:DA:4D:75:3C:53:AA:04:1E:08:92
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

TLS tool: intercepTLS

- Android app to detect TLS interception
  - https://tinyurl.com/interceptls
- Research project (TU Munich)
  - Goal: detect networks that intercept TLS connections



TLS 1.3

- Stronger protection against downgrade attack
- Cryptographic improvements:
  - Full deprecation of MD5
  - New functions:
    - ChaCha20, Poly1305, Ed25519, x25519, x448
- Faster and more private handshake
- Session continuation improvements
- Perfect Forward Secrecy options

Roadmap

- TLS – Transport Layer Security
  - **VPNs**

Layer 4 VPNs – TLS

- TLS can be used for setting layer 4 VPNs:
  - Datagrams encapsulated in TLS records, instead of IPsec datagrams
- Much used today for
  - VPNs, like those you used in the project to protect the communication between two facilities
  - So called “VPN services”: used to hide the origin of communication

OpenVPN

- **OpenVPN** – open implementation of TLS VPNs
  - Supports both *host-to-net VPNs* and *net-to-net VPNs*
  - Operates in client-server mode
- Provides mechanisms for:
  - **Access control**: allow defining which ports, websites, etc. can be accessed
  - **Load balancing**: sends messages to more than 1 destination server in a fair, equilibrated, way
  - **Failover**: if a destination server fails, sends messages to the alternative one(s)