Lecture 13: The Transport Layer Security (TLS) Protocol

TTM4135

Relates to Stallings Chapter 17

Spring Semester, 2025

Motivation

- Transport Layer Security (TLS) is a cryptographic protocol designed to provide communication security over a computer network
- TLS is probably the most widely used security protocol in use today in the real world
- TLS is used to secure communications with banks, online shops, email providers and much more
- TLS uses most of the mainstream cryptographic algorithms which we have studied in this course
- TLS is a very complex protocol and has been the subject of many attacks, and subsequent repairs

Outline

SSL/TLS history

- 1994: Netscape Communications developed Secure Sockets Layer (SSL) 2.0. Should no longer be used.
- 1995: Netscape release SSL 3.0. Should no longer be used.
- RFC 2246 issued in 1999 by IETF documenting Transport Layer Security (TLS) protocol 1.0, similar to SSL 3.0.
- TLS 1.1 specified in 2006 in RFC 4346. Fixes some problems with non-random IVs and exploitation of padding error messages.
- TLS 1.2 specified in 2008 in RFC 5246. Allows use of standard authenticated encryption (instead of separate encryption and MAC).

TLS 1.2 vs. TLS 1.3

- TLS 1.3 standardised in 2018 in RFC 8446 with significant differences from TLS 1.2
- TLS 1.2 is still the most widely supported version today so we focus on **TLS 1.2 in this lecture**
- We focus on TLS 1.3 in the next lecture and explain the main differences there

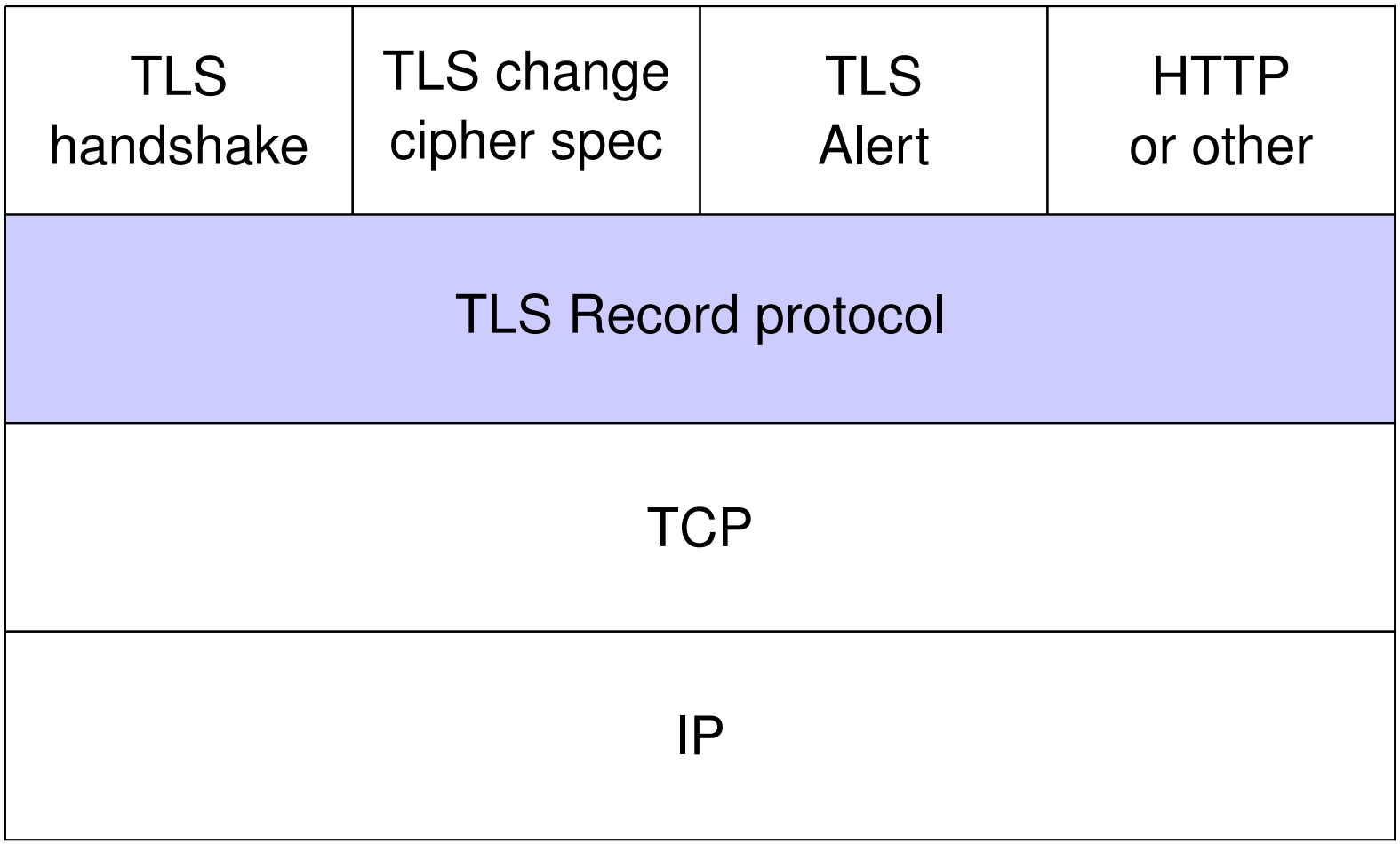
TLS uses

- TLS is a cryptographic services protocol based upon PKI and commonly used on the Internet
- Often used to allow browsers to establish secure sessions with web servers
- Many other application areas

TLS: Architecture overview

- TLS is not a single protocol but rather two layers of protcols (see next slide)
- Consists of 3 higher level protocols:
  - TLS handshake protocol to set up sessions
  - TLS alert protocol to signal events such as failures
  - TLS change cipher spec protocol to change the cryptographic algorithms
- The TLS record protocol provides basic services to various higher level protocols

TLS: Protocol stack



TLS Alert and TLS change cipher spec protocols

- The Alert protocol handles connections, by sending an "alert" message of various degrees of severity
- Three types of alerts:
  - warning alerts
  - `close_notify` alerts
  - fatal alerts
- Improper handling of alert messages can lead to truncation attacks
- The change cipher spec protocol is normally used after the handshake protocol to indicate commencement of secure traffic

TLS ciphersuites

- TLS ciphersuites specify the public key algorithms used in the handshake protocol and the symmetric algorithms used in the record protocol
- Over 300 standardized ciphersuites, many of which are weak and should no longer be used
- Full list is held by IANA
- An example ciphersuite, mandatory in TLS 1.0 and 1.1, is TLS_RSA_WITH_3DES_EDE_CBC_SHA. This means that:
  - the key exchange will use RSA to encrypt a secret chosen by the client (see following slides);
  - triple DES (Encrypt-Decrypt-Encrypt) in CBC mode will be used for encryption;
  - SHA-1 will be used for the HMAC for data integrity.

HMAC – reminder

- Proposed by Bellare, Canetti, Krawczyk in 1996
- Built from any iterated cryptographic hash function H, e.g., MD5, SHA-1, SHA-256, ...
- Standardized and used in many applications including TLS and IPsec
- Details in FIPS-PUB 198-1 (July 2008)

HMAC construction – reminder

Let H be any iterated cryptographic hash function. Then define:

$$\mathrm{HMAC}(M, K) = H\left((K \oplus \mathrm{opad}) \parallel H((K \oplus \mathrm{ipad}) \parallel M) \right)$$

where

- M: message to be authenticated
- K: key padded with zeros to be the block size of H
- opad: fixed string `0x5c5c5c...5c`
- ipad: fixed string `0x363636...36`
- $\parallel$ denotes concatenation of bit strings.
- HMAC is secure (unforgeable) if H is collision resistant or if H is a pseudorandom function.

HMAC – reminder

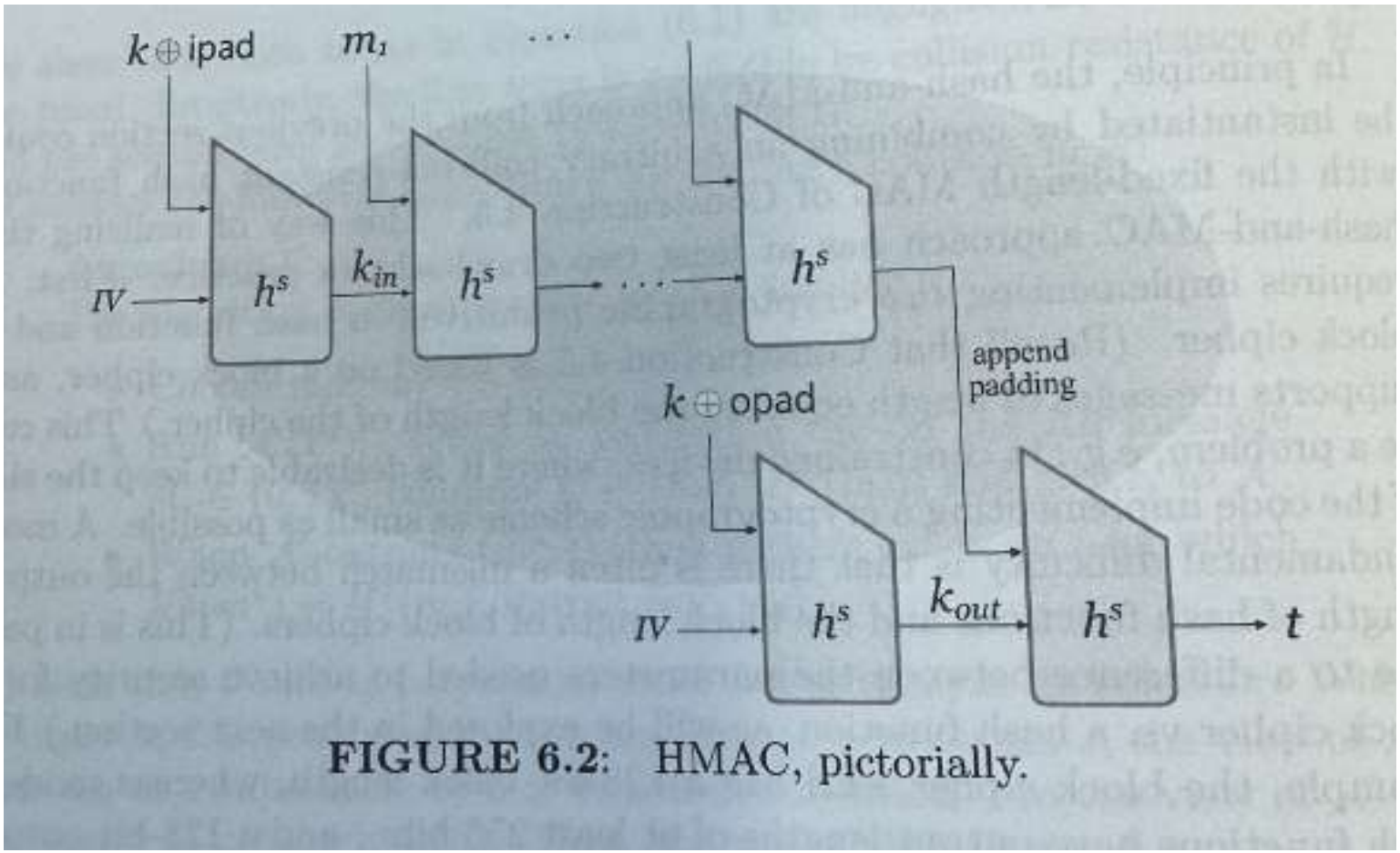


FIGURE 6.2: HMAC, pictorially.

Common TLS 1.2 ciphersuites

- Handshake algorithms (all use signed Diffie–Hellman)
  - DHE-RSA Ephemeral Diffie-Hellman with RSA signatures
  - ECDHE-RSA Elliptic curve DHE with RSA signatures
  - DHE-DSS DHE with DSS signatures
- Record algorithms
  - AES-GCM AES authenticated encryption with GCM mode
  - AES-CBC-SHA256 AES in CBC mode with HMAC from SHA256
  - CHACHA20-POLY1305 ChaCha stream cipher with Poly1305 MAC

Record protocol overview

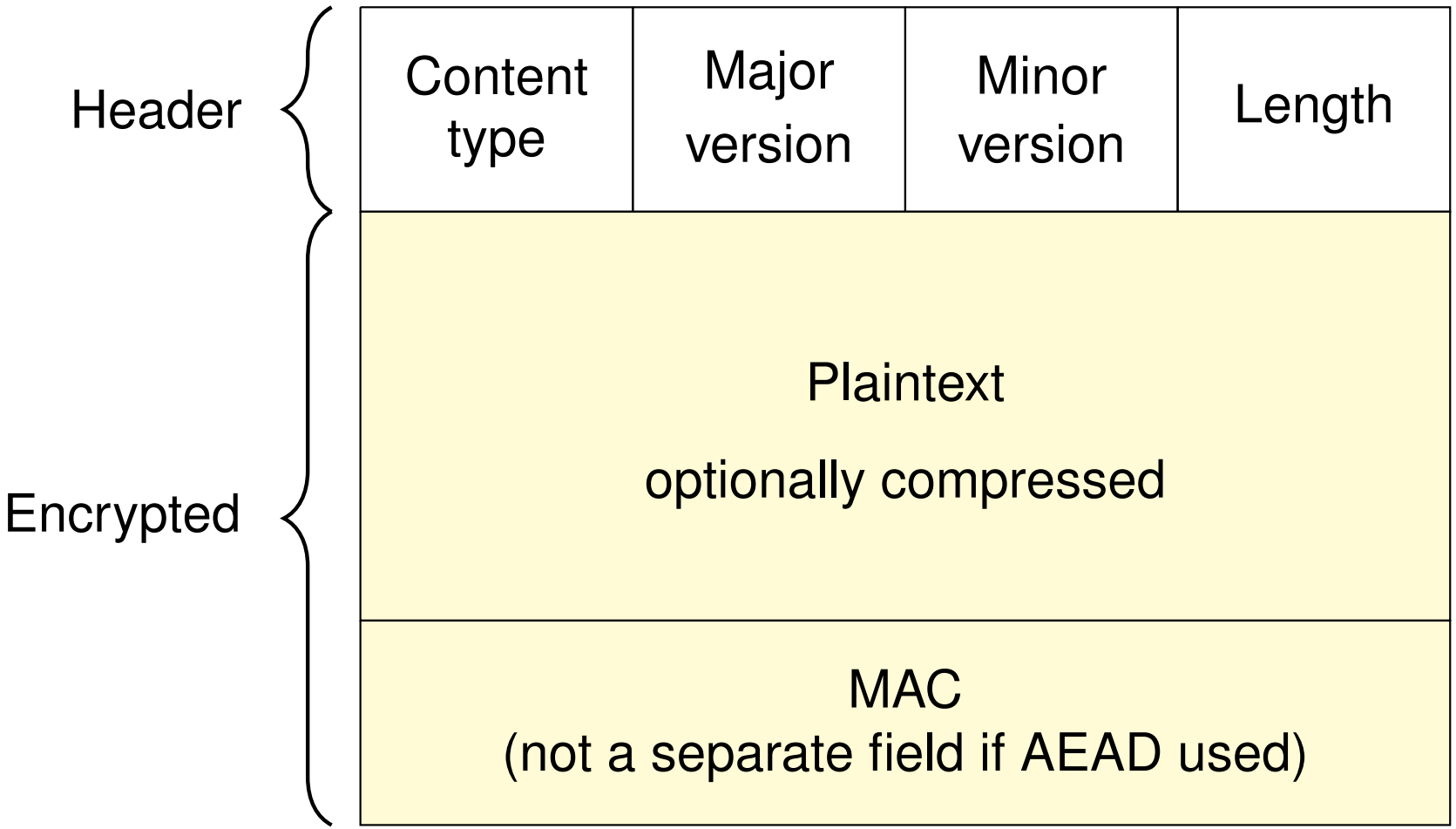
Provides two services for higher-layer protocols.

Message confidentiality: Ensure that the message contents cannot be read in transit

Message integrity: Ensure that the receiver can detect if a message is modified in transit

- These services can be provided by a symmetric encryption algorithm and a MAC
- From TLS 1.2, these services are often provided with authenticated encryption with associated data (AEAD) modes CCM or GCM
- The handshake protocol establishes symmetric keys (session keys) to use with these mechanisms

Record protocol format



TLS: Record protocol header

- Content Type: Defined content types are:
  - change-cipher-spec
  - alert
  - handshake
  - application-data
- Protocol Version
  - Major Version: 3 for TLS
  - Minor Version: 1 for TLS v1.0; 2 for TLS v1.1; 3 for TLS v1.2.
- Length: length in octets of the data

Record protocol operation

- Fragmentation: Each application layer message is fragmented into blocks of 2^{14} bytes or less
- Compression: Optionally applied – default compression algorithm is null
- Authenticated data: consists of the (compressed) data, the header, and an implicit record sequence number
- Plaintext: Compressed data and the MAC, if present
- Session keys for the MAC and encryption algorithms, or AEAD algorithm, are established during the handshake protocol
- The encryption and MAC algorithms are specified in the negotiated *ciphersuite*

Record protocol cryptographic algorithms

MAC: The algorithm used is HMAC in all TLS versions using a negotiated hash function. SHA-2 is allowed only from TLS 1.2.

Encryption: Either a negotiated block cipher in CBC mode or a stream cipher. Most common block ciphers are AES and 3DES. RC4 originally supported in TLS 1.2. For block ciphers, padding is applied after the MAC to make a multiple of the cipher block size.

AEAD: Allowed instead of encryption and MAC in TLS 1.2. Usually AES in CCM or GCM modes. Authenticated additional data is the header and implicit record sequence number.

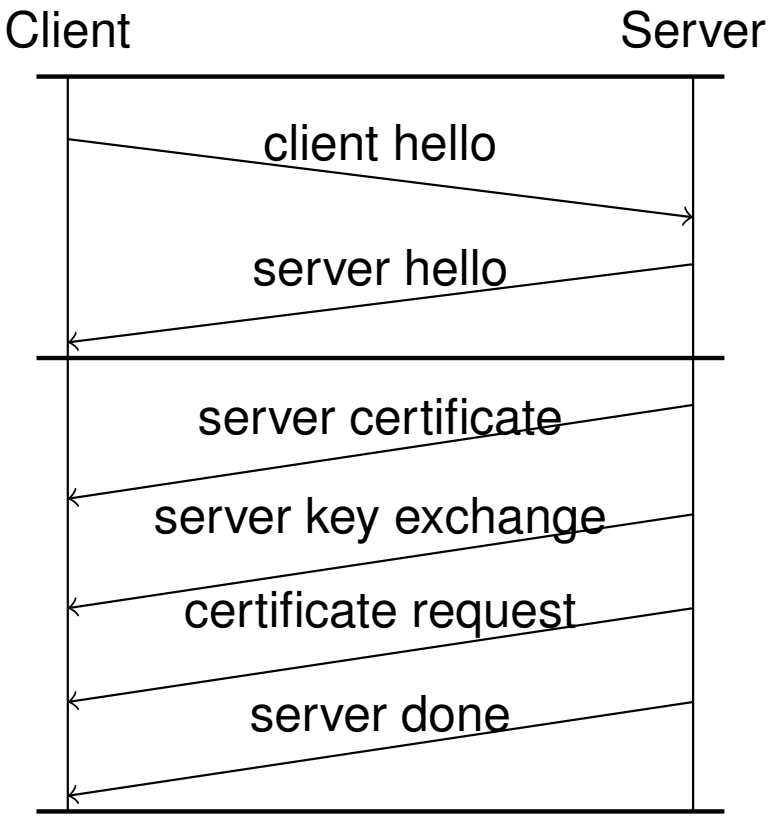
Handshake protocol purpose

- Negotiates the version of TLS and the cryptographic algorithms to be used
- Establishes a shared session key for use in the record protocol
- Authenticates the server
- Authenticates the client (optional)
- Completes the session establishment
- Several variations of the handshake:
  - RSA variant (still supported but not recommended)
  - Diffie-Hellman variant (recommended)
  - Pre-shared key variant
  - Mutual authentication or server-only authentication

Handshake protocol - four phases

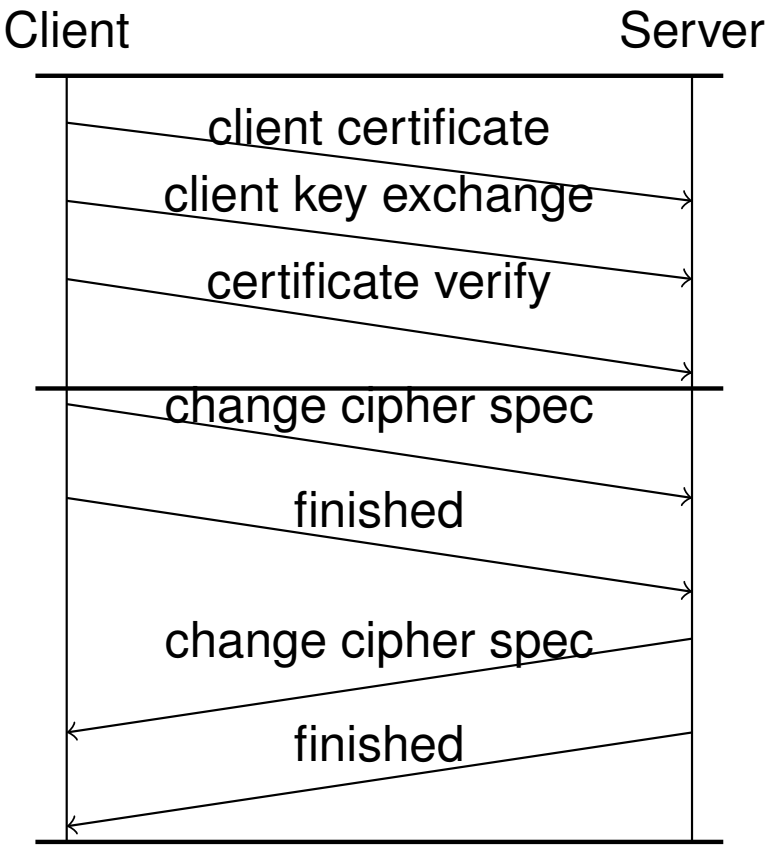
- Phase 1: Initiates the logical connection and establishes its security capabilities
- Phases 2 and 3: Performs key exchange with messages and message content depending on the handshake variant negotiated in phase 1
- Phase 4: Completes the setting up of a secure connection

Handshake protocol - phases 1 and 2



- Phase 1: Client and server negotiate version, cipher suite and compression and exchange nonces
- Phase 2: Server sends certificate and key exchange message (if needed)

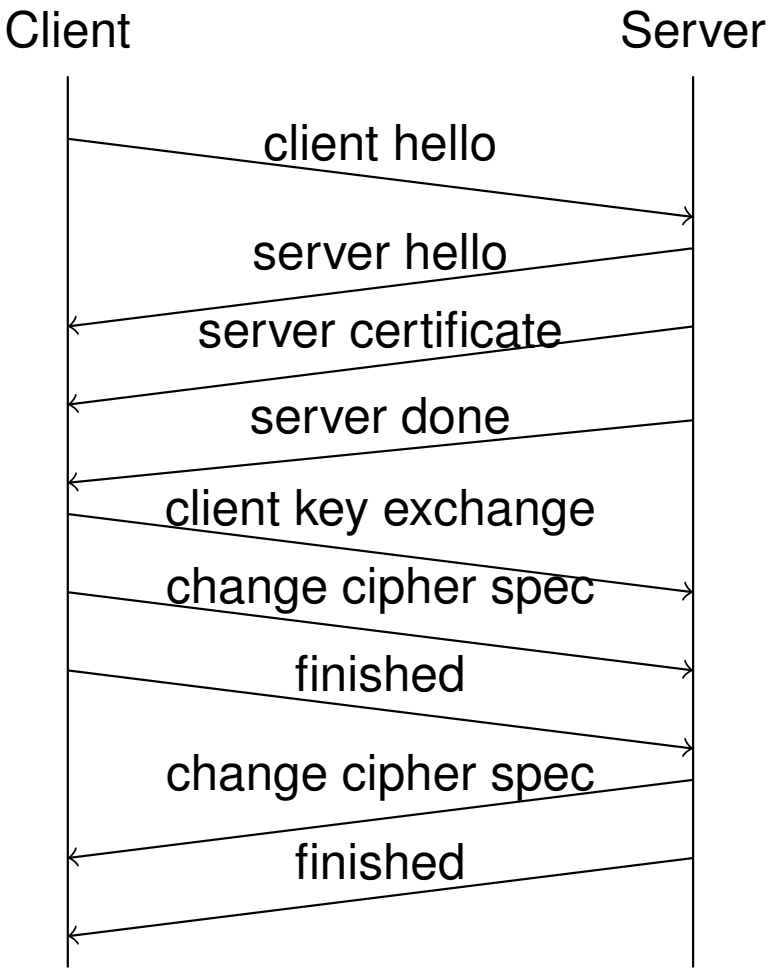
Handshake protocol - phases 3 and 4



- Phase 3: Client sends certificate and key exchange message
- Phase 4: Client and server start secure communications

Finished messages include a check value (pseudo-random function) of all the previous messages

RSA-based handshake protocol



- Simplest variation with server-only authentication and server public key suitable for RSA encryption.
- On completion of Phase 1, assume that RSA-based key exchange has been selected.

Main handshake messages

Client hello States highest version of TLS available, advertises ciphersuites available to the client and sends client nonce N_C

Server hello Returns the selected version and ciphersuite and sends server nonce N_S

Server key exchange Server's input to key exchange

Client key exchange Client input to key exchange

Change-cipher-spec Switch to newly negotiated ciphersuite for record layer

Ephemeral Diffie–Hellman handshake variant

Server key exchange Diffie–Hellman generator and group parameters and server ephemeral Diffie-Hellman value, all signed by server

Client key exchange Client ephemeral Diffie-Hellman value. This is optionally signed by the client if the client certificate is used

- Pre-master secret *pms* is the shared Diffie–Hellman secret
- Provides forward secrecy and therefore *recommended* today

RSA handshake variant

Server key exchange Not used

Client key exchange Key transport of pre-master secret *pms*

- Client selects a random *pre-master secret*, *pms*
- Client encrypts *pms* with the server's public key and sends the ciphertext to the server
- Server decrypts using its private key to recover *pms*

No forward secrecy and *not recommended* today

Generating session keys

- The master secret, *ms*, is defined using the premaster secret *pms*, as:

$$ms = PRF(pms, \text{“master secret”}, N_C \parallel N_S)$$

- As much keying material is generated as is required by the ciphersuite using:

$$k = PRF(ms, \text{“key expansion”}, N_S \parallel N_C)$$

- Independent session keys are partitioned from *k* in each direction (a write key and a read key on each side)
- Depending on ciphersuite, keying material may include:
  - encryption key
  - MAC key
  - IV

The pseudorandom function PRF

- The PRF (pseudo-random function) is built from HMAC with a specified hash function
  - The PRF in TLS 1.0 and TLS 1.1 is based on a combination of MD5 and SHA-1
  - In TLS 1.2 the PRF is based on SHA-2
- For example, in TLS 1.2:

$$\begin{aligned} PRF(K, label, r) \quad = \quad & HMAC(K, A(1) \parallel label \parallel r) \parallel \\ & HMAC(K, A(2) \parallel label \parallel r) \parallel \\ & \ldots \end{aligned}$$

where $A(0) = r$, $A(i) = HMAC(K, A(i-1))$ and HMAC uses a specified SHA-2 variant, typically SHA256, as its hash function, r is a seed.

Other handshake variants

Diffie-Hellman (DH) The parties use **static** Diffie–Hellman with certified keys — if the client does not have a certificate (usual on the Internet) then the client uses an ephemeral Diffie-Hellman key

Anonymous Diffie-Hellman (DH_Anon) The ephemeral Diffie-Hellman keys are not signed at all, so only protects against passive eavesdropping

The above methods for the handshake protocol are possible but not recommended today

Forward secrecy

- Forward secrecy is the property that compromise of long-term keys should not lead to compromise of session keys established before the long-term key compromise took place.
- A typical way to achieve forward secrecy is to use Diffie–Hellman key exchange with the exchange authenticated using signatures from the long-term keys.
- Using the RSA-based handshake in TLS does not achieve forward secrecy but is currently still used in many ciphersuites.
- Several ciphersuites using Diffie–Hellman, including elliptic curve Diffie–Hellman, are defined for TLS which provide forward secrecy.

SSL and TLS Limitations

- Higher layers should not be overly reliant on SSL or TLS always negotiating the strongest possible connection between two peers
- There are a number of ways a man-in-the-middle attacker can attempt to make two entities drop down to the least secure method they support.
  - Known as a downgrade attack.
- For example, an attacker could block access to the port a secure service runs on, or attempt to get the peers to negotiate an unauthenticated connection.

TLS protocol summary

- TLS includes two main protocols: the Handshake Protocol and the Record Layer Protocol
- New versions have been rolled out as understanding of cryptography and potential attacks increase
- Backward compatibility is a problem:
  - SSL 3.0 finally deprecated in 2015
  - TLS 1.0 more than 20 years old and still supported widely
- TLS assumes reliable delivery of messages, provided by TCP

TLS limitations

- In the past few years there have been many practical attacks on TLS
- Many servers do not support the latest versions of TLS and/or have not protected against known attacks
- SSL Pulse survey gives an up-to-date picture of current attacks
- Good coverage of some attacks is given on Matt Green's blog
- A selection of some previous attacks is on the following slides

The BEAST attack

- BEAST (Browser Exploit Against SSL/TLS) exploits non-standard use of IV in CBC mode encryption - IVs are chained from previous ciphertexts
- Allows attacker to recover plaintext byte by byte
- Known as a theoretical weakness since 2002, but only demonstrated in 2011
- From TLS 1.1 only random IV is allowed
- Most browsers now implement a mitigation strategy based on splitting plaintext into first byte + remainder to force a randomised IV including a MAC computation
- No longer considered a realistic threat

The CRIME and BREACH attacks

- Side channel attacks based on **compression** - different inputs result in different amounts of compression
- CRIME (Compression Ratio Info-leak Made Easy) exploits compression in TLS, while BREACH (Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext) exploits compression in HTTP
- Idea of attack known already in 2002 but practically demonstrated in 2012
- Commonly recommended to switch off compression in TLS but switching off in HTTP too results in big performance penalty

Padding oracles and the POODLE attack

- A padding oracle is a way for an attacker to know if a message in a ciphertext was correctly padded
- In 2002 it was shown how in theory CBC mode encryption can provide a padding oracle due to its error propagation properties
- Applied to TLS in a variety of attacks
- Main mitigation is a uniform error response, so that the attacker cannot distinguish padding errors from MAC errors
- POODLE (Padding Oracle On Downgraded Legacy Encryption) published in October 2014 forces downgrade to SSL 3.0 and then runs padding oracle attack

The Heartbleed bug

- An implementation error in OpenSSL
- Based on missing bounds check in *heartbeat* messages
- Allows memory leakage from server which is likely to include session keys and long-term keys
- Discovered April 2014 and required updating of many server keys after bug was fixed
- Is it reasonable that big companies use free software for securing important transactions?

TLS Timing (Padding) Oracle Attack

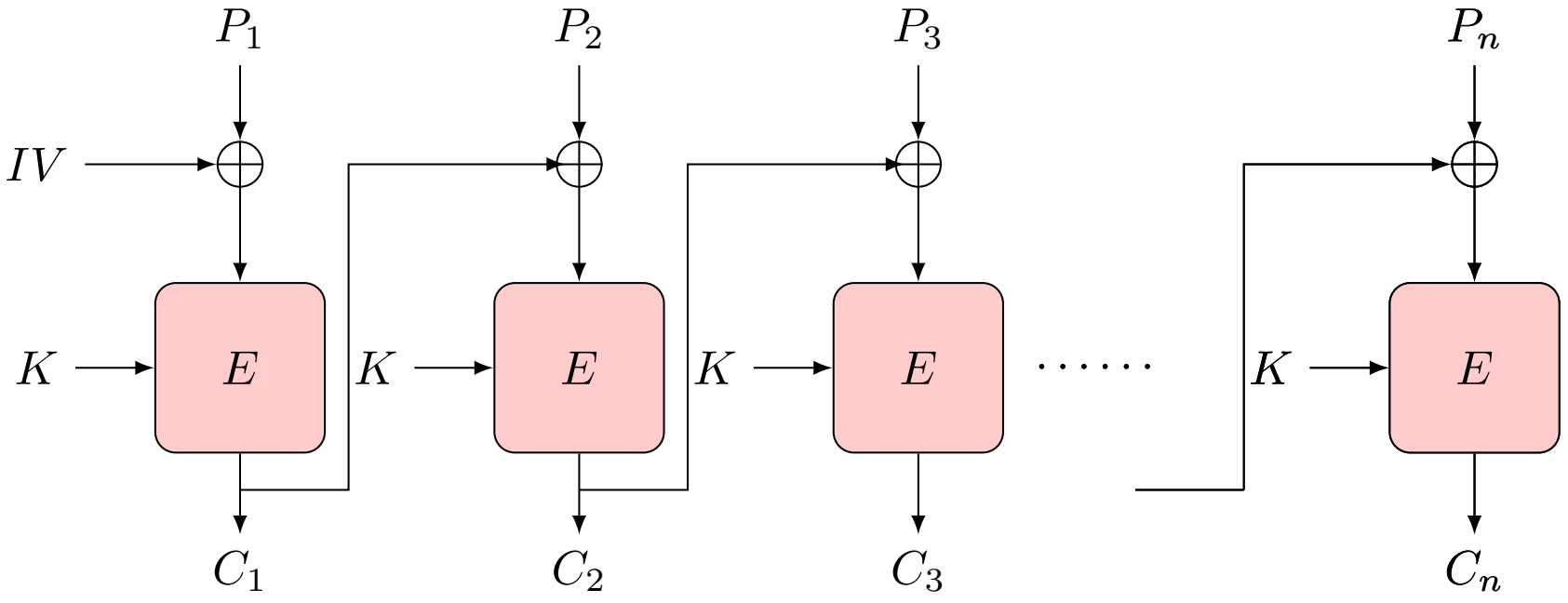
- Attack discovered by Nadhem AlFardan and Kenny Patterson from (formerly) Royal Holloway, University of London, UK (now at ETH)
- Dubbed “Lucky 13”
- TL;DR
  - *“There is a subtle timing bug in the way that TLS data decryption works when using the (standard) CBC mode ciphersuite. Given the right set of circumstances, an attacker can use this to completely decrypt sensitive information, such as passwords and cookies. ”*
  - Borderline practical in DTLS, more on the theoretical side if using TLS

https://blog.cryptographyengineering.com/2013/02/04/attack-of-week-tls-timing-oracles/
https://en.wikipedia.org/wiki/Lucky_Thirteen_attack

Cipher Block Chaining (CBC) mode – reminder

- CBC "chains" the blocks together
- A random initialisation vector IV is chosen and sent together with the ciphertext blocks
- *Encryption*:
  - $C_t = E(P_t \oplus C_{t-1}, K)$, where $C_0 = IV$
  - P_t is XOR'd with the previous ciphertext block C_{t-1}, and encrypted with key K to produce ciphertext block C_t
  - IV is used for the value C_0 and sent with $C_1, \ldots C_n$
- *Decryption*:
  - $P_t = D(C_t, K) \oplus C_{t-1}$, where $C_0 = IV$
  - C_t is decrypted with the key K, and XOR'd with the previous ciphertext block C_{t-1} to produce plaintext block P_t
  - As in encryption, C_0 is used as the IV

CBC mode encryption



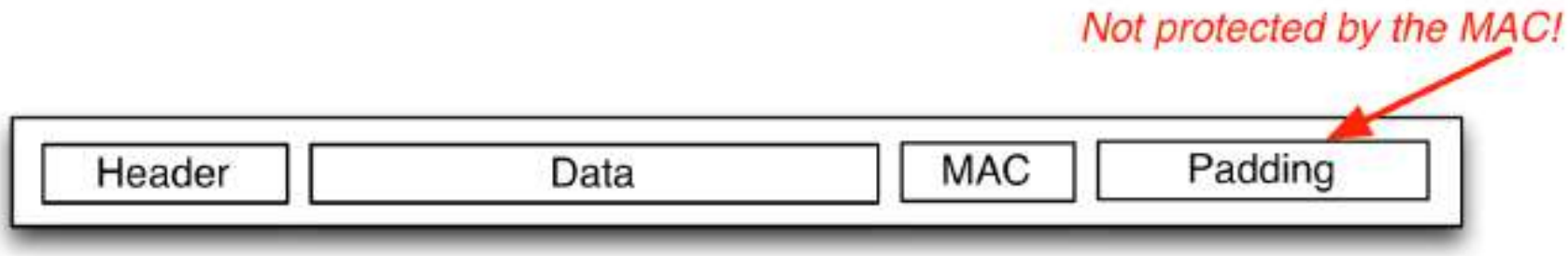
- IV and blocks $C_1, C_2, \ldots C_n$ are sent

The attack

- Recall that we saw that you should always encrypt a message first, then apply the MAC to the resulting ciphertext
- But TLS gets this backwards
- Upon encrypting a record,
  - The sender first applies a MAC to the plaintext
  - Then adds up to 255 bytes of padding to get the message up to a multiple of the cipher (e.g., AES) block size
  - Only then does it CBC-encrypt the record.

The attack

The critical point is that the padding is *not* protected by the MAC.



This means an attacker can tamper with the padding by flipping specific bits in the ciphertext, leading to a *padding oracle attack*. She can re-transmit to the server for decryption. If the attacker can learn whether her changes affected the padding, she can use this information to adaptively decrypt the whole record.

The attack

- These kinds of attacks were known to TLS designers
- But, instead of fixing, they patched it
- They did so by eliminating eliminate any error messages that could indicate whether the padding check failed
- But researchers realised they could run a *timing* attack instead
  - Time how long decryption took, deduce whether there was a padding failure or not
  - Implementations would first check padding, then return immediately (without checking the MAC) if this failed

The attack

"[T]he best way to do this is to compute the MAC even if the padding is incorrect, and only then reject the packet. For instance, if the pad appears to be incorrect, the implementation might assume a zero-length pad and then compute the MAC." (TLS 1.2 spec)

- When the padding check fails, the decryptor doesn't know how much padding to strip off
- Meaning doesn't know how much data to MAC
- Recommended countermeasure is to assume no padding, then MAC the whole blob
  - The results is that the MAC computation can take longer when the padding is damaged

The attack

"This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal."

- This remained true indeed for several years
- The attack showed that it *was* possible to distinguish, at least from a relatively short distance
- Partly due to advances in computer hardware
- The new technique was able to measure timing differentials of less than 1 microsecond over a LAN connection

TLS security summary

- Different kinds of attacks: implementation errors, poor choice of cryptographic primitives, flaws in protocol.
- Backward compatibility is a problem (Downgrade attacks).
- Several examples of the principle that “attacks only get better” over time.
- Complexity is a major problem. TLS 1.3 will remove many options, both in cipher suites and protocol options.
- TLS 1.3 will simplify the handshake
- TLS 1.3 adds new features (e.g. 0-RTT mode) which present new challenges.