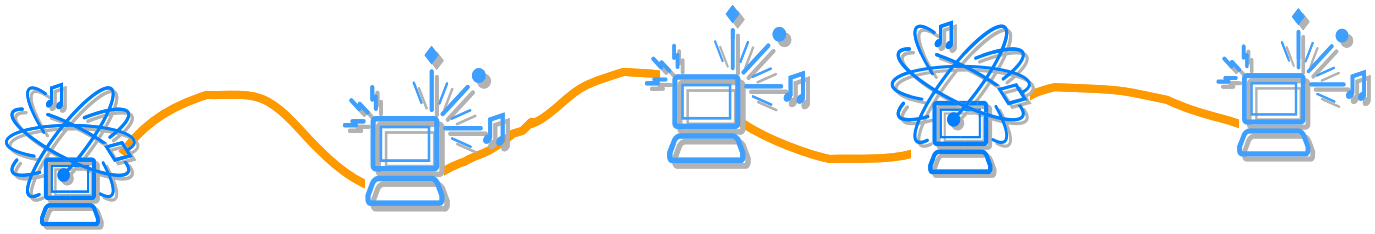
Segurança em Redes
SSL / TLS (*Secure Sockets Layer*)



Redes de Comunicação de Dados
Área Departamental de Engenharia da Eletrónica e das Telecomunicações e de Computadores
Instituto Superior de Engenharia de Lisboa

What is TLS

- TLS stands for Transport Layer Security and is the successor to SSL (Secure Sockets Layer).
- Many IP-based protocols, such as HTTP, SMTP, POP3, FTP uses TLS to protect data.
- TLS provides secure communication between web browsers and servers. The connection itself is secure because symmetric cryptography is used to encrypt the data transmitted.
- The keys are uniquely generated for each connection and are based on a shared secret negotiated at the beginning of the session, also known as a TLS handshake.

SSL/TLS – Introdução

- Objetivos principais
  - Autenticação do servidor perante o cliente e (opcional) vice-versa
  - Confidencialidade dos dados
  - Integridade de dados
  - Compressão dos dados (TLS 1.3 não suporta)
  - Geração e distribuição de chaves de sessão integrados no protocolo
  - Negociação dos parâmetros de segurança

SSL/TLS

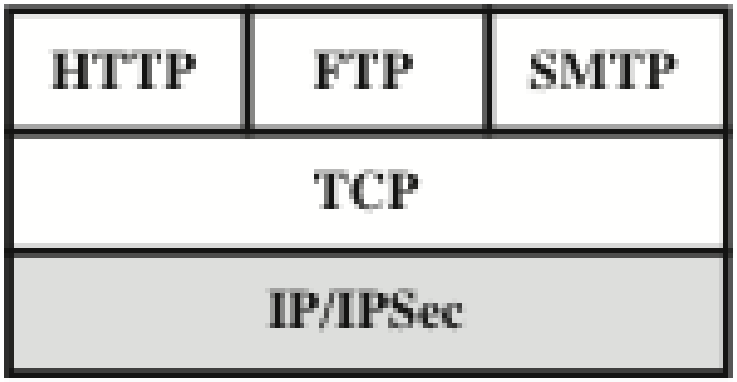
- One of the most widely used security services
- Defined in several RFCs
- An Internet standard that evolved from a commercial protocol known as Secure Sockets Layer (SSL)
- It is a general-purpose service implemented as a set of protocols that rely on **TCP**
- Could be provided as part of the underlying protocol suite and therefore be transparent to applications
- Can be embedded in specific packages
- Most browsers come equipped with TLS and most Web servers have implemented the protocol

SSL/TLS - Objectives

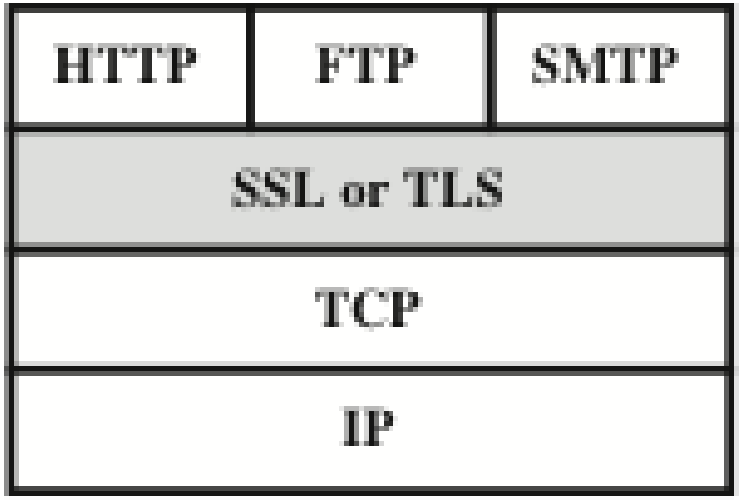
When secured by TLS, connections between a client (e.g., a web browser) and a server (e.g., wikipedia.org) should have one or more of the following properties:

- The **connection is *private*** (or ***secure***) because symmetric cryptography is used to encrypt the data transmitted. The keys for this symmetric encryption are generated uniquely for each connection and are based on a shared secret that was negotiated at the start of the session. The server and client negotiate the details of which encryption algorithm and cryptographic keys to use before the first byte of data is transmitted. The negotiation of a shared secret is both secure (the negotiated secret is unavailable to eavesdroppers and cannot be obtained, even by an attacker who places themselves in the middle of the connection) and reliable (no attacker can modify the communications during the negotiation without being detected).
- The **identity of the communicating parties can be *authenticated*** using public-key cryptography. This authentication can be made optional but is generally required for at least one of the parties (typically the server).
- The **connection is *reliable*** because each message transmitted includes a message integrity check using a message authentication code to prevent undetected loss or alteration of the data during transmission.

Security level in the TCP/IP protocol stack



(a) Network Level



(b) Transport Level

Historial do SSL/TLS

Foi desenvolvido pela **Netscape**, em meados da década de **1990** (SSL 2.0), para permitir garantir a segurança no transporte de HTTP, LDAP, POP3 e outros.

Foi desenhado para utilizar o **TCP como camada de transporte** e providenciar confidencialidade, integridade e autenticação entre dois extremos da camada de transporte.

Embora o SSL e o TLS não sejam compatíveis entre si, este último pode ser entendido como uma evolução do SSL. Assim sendo vamos estudá-los juntos fazendo referência, quando for pertinente, às respetivas diferenças.

Historial do SSL/TLS

- SSL 1.0
  - Projetado pela Netscape por volta de 1994 (??)
  - Há muito esquecido!
- SSL 2.0
  - Publicado pela Netscape, Novembro de 1994. Possui várias fraquezas
- SSL 3.0
  - Desenhado pela Netscape e por Paul Kocher, Novembro de 1996
- TLS 1.0
  - Desenvolvido sobre a égide do IETF (RFC 2246), baseado no SSL 3.0, Janeiro de 1999. Não é compativel com o SSL 3.0, poderia ser entendido como um SSL 3.1
    - Ex. O TLS usa HMAC em vez do MAC; pode correr em qualquer porto
- TLS 1.1 (RFC 4346) – Evolução do TLS 1.0
- TLS 1.2 (RFC 5246 / 2008)
- **TLS 1.3 (RFC 8446 / 2018)**

TLS 1.3

While TLS 1.3 is not **directly** compatible with previous versions, all versions of TLS incorporate a versioning mechanism which allows clients and servers to interoperably negotiate a common version if one is supported by both peers.

You can read more about the TLS 1.3 in https://kinsta.com/blog/tls-1-3/#, including a presentation by Filippo Valsorda from Cloudflare [https://www.cloudflare.com, https://vimeo.com/177333631] where he resumes the diferences between TLS 1.2 and TLS 1.3.

SSL/TLS Support

Website protocol support (May 2022)		
Protocol version	Website support[71]	Security[71][72]
SSL 2.0	0.3%	Insecure
SSL 3.0	2.5%	Insecure[73]
TLS 1.0	37.1%	Deprecated[8][9][10]
TLS 1.1	40.6%	Deprecated[8][9][10]
TLS 1.2	99.7%	Depends on cipher[n 1] and client mitigations[n 2]
TLS 1.3	54.2%	Secure

Notes

see § Cipher table at article bellow

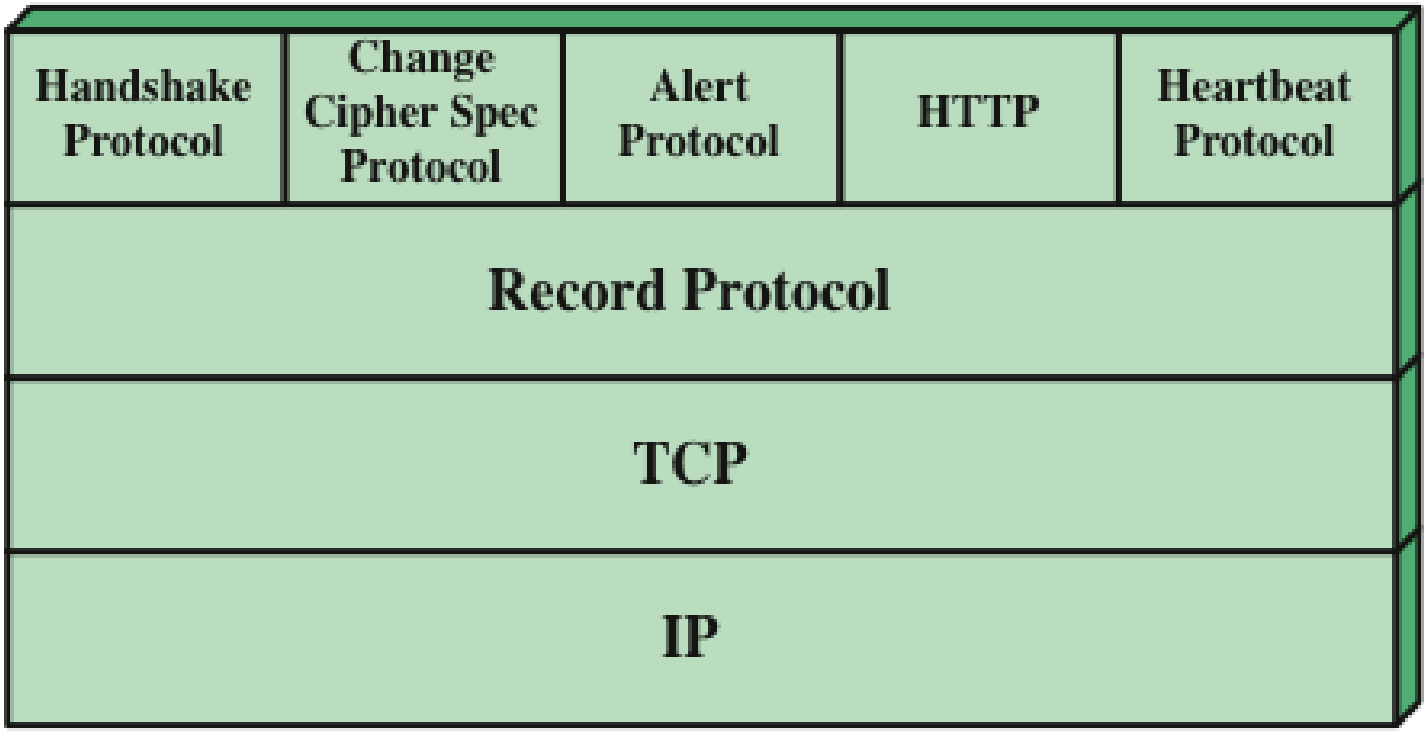
see § Web browsers and § Attacks against TLS/SSL sections at article bellow

Wikipedia (June 2022): https://en.wikipedia.org/wiki/Transport_Layer_Security

SSL/TLS Características básicas

- SSL/TLS é usado nos *browsers* Web e em servidores para dar suporte ao *e-commerce* seguro sobre HTTP.
  - Suportado por Microsoft IE, Netscape, Mozilla, Apache, IIS, …
- A arquitetura SSL inclui duas camadas:
  - **Camada inferior**
    - Record Protocol
      - **Fornece canal seguro e fiável à camada superior**. Assenta sobre o protocolo de transporte (**TCP**), sendo responsável por garantir a confidencialidade e integridade da comunicação transmitida. Opcionalmente pode realizar compressão (no TLS 1.3 não).
  - **Camada superior**
    - SSL Handshake Protocol
    - Change Cipher Spec. Protocol
    - Alert Protocol
    - Heartbeat Protocol
    - HTTP
    - outros protocolos de aplicação

SSL/TLS: Arquitetura do protocolo



SSL/TLS: Funções dos vários componentes

- **Record Protocol**
  - Fragmentação
  - Cifra usando cifras simétricas
  - Autenticação das mensagens e proteção da integridade
  - Compressão (TLS 1.3 já não suporta)
- **Handshake Protocol**
  - Assenta sobre o Record Protocol e é responsável por:
    - Estabelecer os parâmetros de segurança da sessão de comunicação, incluindo os algoritmos e as chaves partilhadas para confidencialidade, autenticação e integridade (tipicamente usando criptografia de chave pública);
    - Autenticar a identidade de um ou de ambos os intervenientes e sinalizar a ocorrência de situações de erro.
- **Change Cipher Spec Protocol**
  - Uma única mensagem que indica o fim do SSL Handshake
- **Alert Protocol**
  - Mensagens de erro (alertas fatais e avisos)
- **Heartbeat Protocol**
  - Mantem a atividade na ligação

SSL/TLS: Record Protocol

- Fornece um canal seguro e fiável à camada superior
- É suportado sobre o **TCP**
- Transporta as mensagens das aplicações acima e de outros protocolos do SSL
- Fragmentação
- Cifra usando cifras simétricas
- Autenticação das mensagens e proteção da integridade
- Compressão (TLS 1.3 já não suporta)
- Estão-lhe associados os conceitos de sessão e de ligação

SSL/TLS: Record Protocol

- O SSL *Record Protocol* fornece:
  - **Autenticação** dos dados e **integridade**
    - **MAC usando um algoritmo semelhante ao HMAC**.
    - Baseado nos algoritmos de *hash* **MD5** ou **SHA-1**.
    - O MAC protege um **número de sequência a 64 bits para protecção anti-repetição**.
  - **Confidencialidade** (limitados os protocolos possíveis no TLS 1.3)
    - Cifra utilizando algoritmos de chave simétrica.
      - IDEA, RC2-40, DES-40 (exportável), 3DES, … [bloco]
      - RC4-40 e RC4-128. [*stream*]
- Os dados da aplicação/camadas superiores ao protocolo SSL são divididos em fragmentos (máx. dimensão: 2^{14} bytes = **16 Kbytes**).
- Primeiro o MAC, depois o PAD (se necessário) e finalmente cifra.
- Adiciona o *header*
  - Tipo de conteúdo, versão, comprimento do fragmento.
- Submetido ao TCP.

SSL/TLS: Record Protocol

- **O *record protocol* divide-se em três camadas:**
  - **formatação** da mensagem em fragmentos
  - **compressão** (opcional, não suportada em TLS 1.3)
  - **MAC e cifra**
- Na **primeira camada** a informação é dividida em pacotes

```
struct {
      ContentType type;
      ProtocolVersion version;
      uint16 length;
      opaque fragment[TLSPlaintext.length];
}TLSPlaintext;
```

- Na **segunda camada** os pacotes são comprimidos

```
struct {
      ContentType type;
      ProtocolVersion version;
      uint16 length;
      opaque fragment[TLSCompressed.length];
 }TLSCompressed;
```

SSL/TLS: Record Protocol

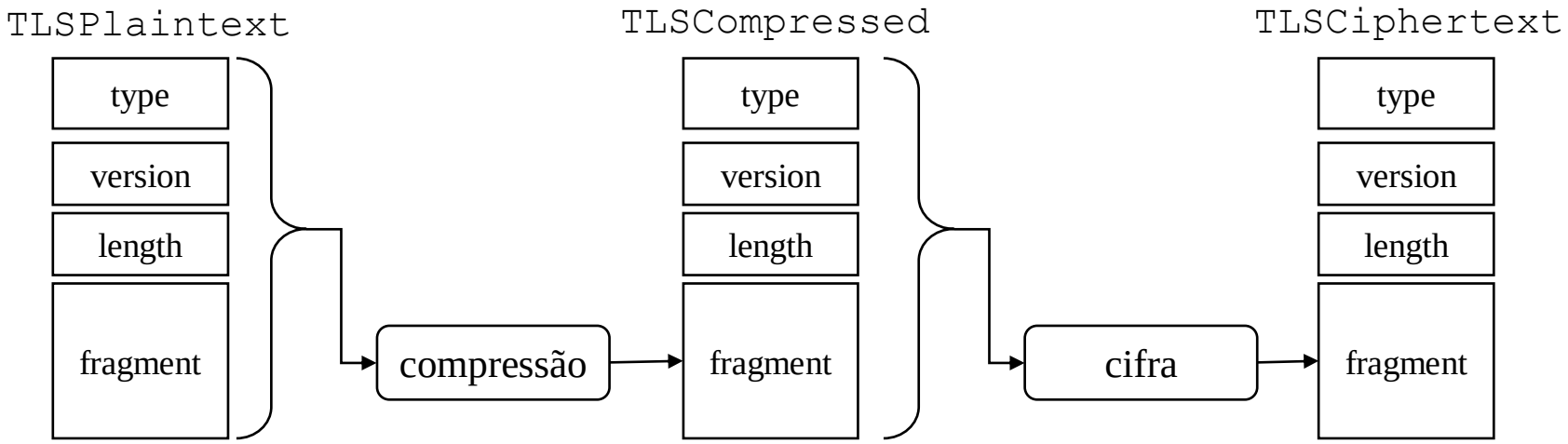
A **terceira camada** cifra e insere códigos de autenticação

```
struct {
    ContentType type;
    ProtocolVersion version;
    uint16 length;
    select (CipherSpec.cipher_type) {
        case stream: GenericStreamCipher;
        case block: GenericBlockCipher;
    } fragment;
 } TLSCiphertext;

block-ciphered struct {
    opaque content[TLSCompressed.length];
    opaque MAC[CipherSpec.hash_size];
    uint8 pad[GenericBlockCipher.pad_len];
    uint8 pad_len;
 } GenericBlockCipher;
```

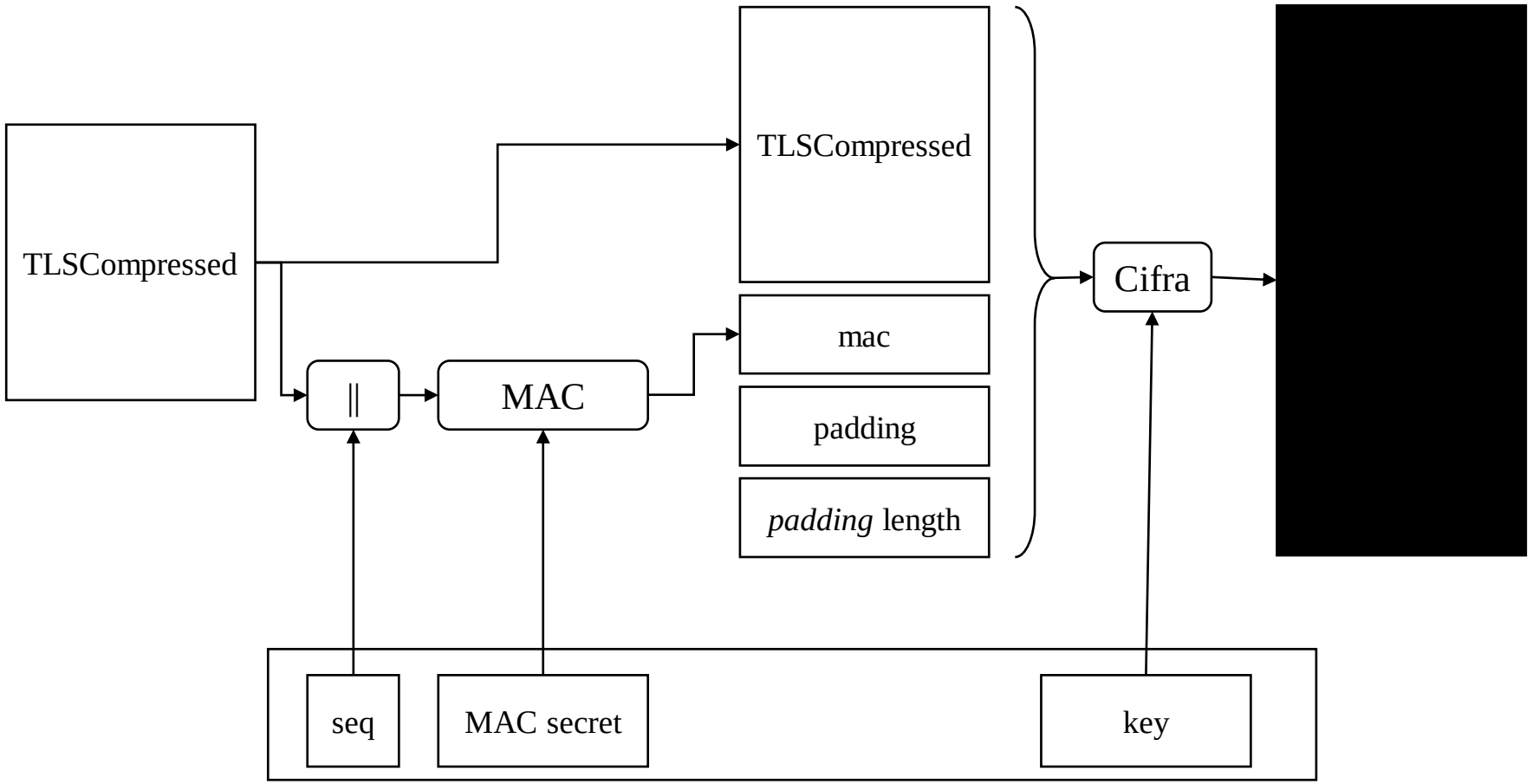
O membro *fragment* da estrutura TLSCiphertext contém a cifra da estrutura GenericBlockCipher

SSL/TLS: Record Protocol

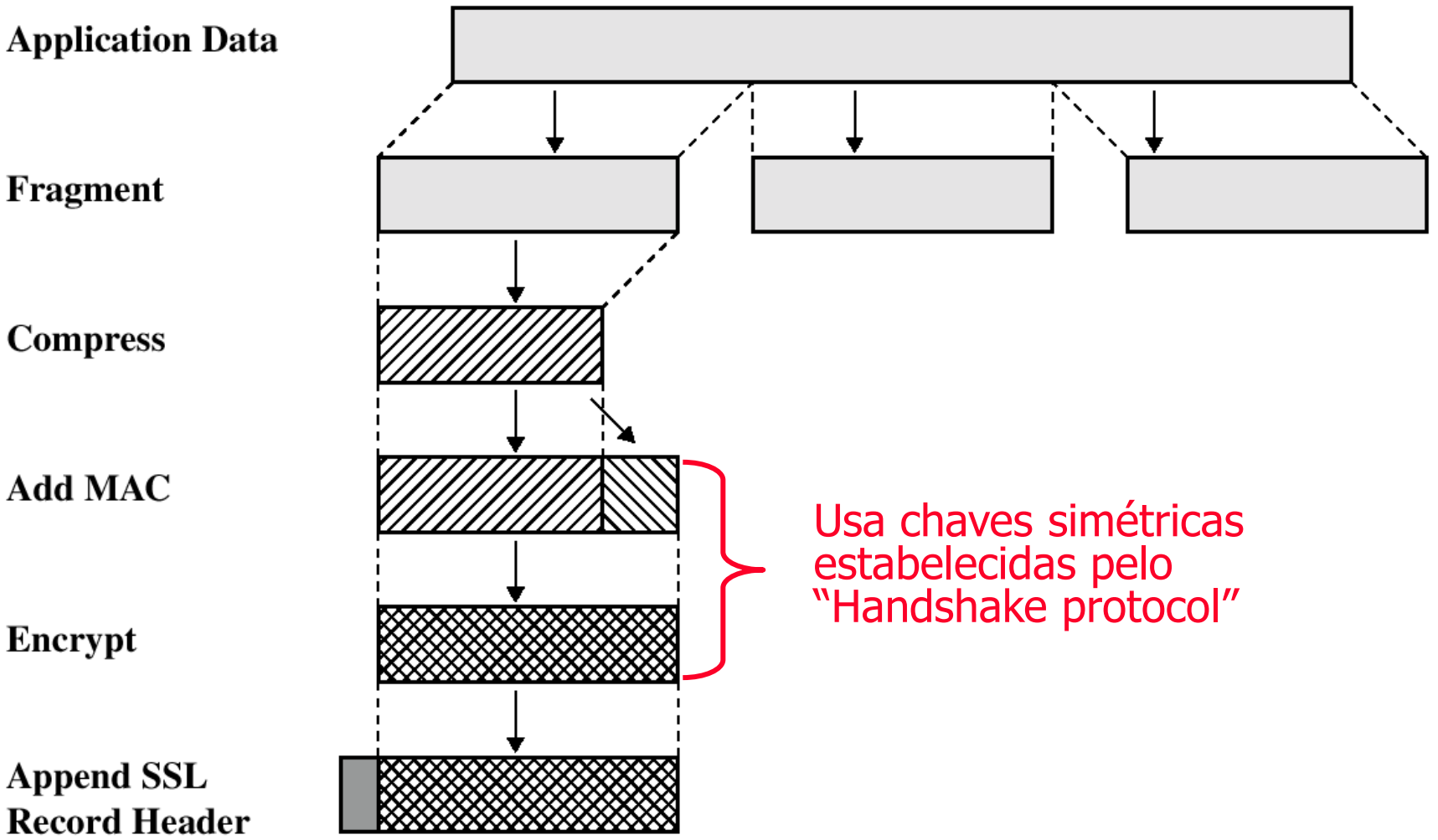


SSL/TLS: Record Protocol

Número de sequência, MAC e cifra



SSL/TLS: Record Protocol



SSL/TLS: Record Protocol – Message format

General format of all TLS records

+	Byte +0	Byte +1	Byte +2	Byte +3
Byte 0	Content type			
Bytes 1..4	Legacy version		Length	
	(Major)	(Minor)	(bits 15..8)	(bits 7..0)
Bytes 5..(*m*−1)	Protocol message(s)			
Bytes *m*..(*p*−1)	MAC (optional)			
Bytes *p*..(*q*−1)	Padding (block ciphers only)			

SSL/TLS: Record Protocol – Message format

Content types		
Hex	**Dec**	**Type**
0x14	20	ChangeCipherSpec
0x15	21	Alert
0x16	22	Handshake
0x17	23	Application
0x18	24	Heartbeat

SSL/TLS: Record Protocol – Message format

Legacy version

Major version	Minor version	Version type
3	0	SSL 3.0
3	1	TLS 1.0
3	2	TLS 1.1
3	3	TLS 1.2
3	4	TLS 1.3

Length

The length of "protocol message(s)", "MAC" and "padding" fields combined, not to exceed 2^{14} bytes (16 KiB).

Protocol message(s)

One or more messages identified by the Protocol field. Note that this field may be encrypted depending on the state of the connection.

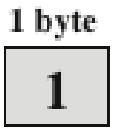
SSL/TLS: Record Protocol – Message format

MAC and padding

A message authentication code computed over the "protocol message(s)" field, with additional key material included. Note that this field may be encrypted, or not included entirely, depending on the state of the connection.

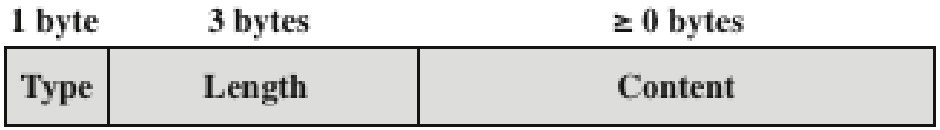
No "MAC" or "padding" fields can be present at end of TLS records before all cipher algorithms and parameters have been negotiated and handshaked and then confirmed by sending a CipherStateChange record for signaling that these parameters will take effect in all further records sent by the same peer.

SSL/TLS: Record Protocol payload



(a) Change Cipher Spec Protocol

(b) Alert Protocol



(c) Handshake Protocol

(d) Other Upper-Layer Protocol (e.g., HTTP)

Algoritmos suportados (vários eliminados no TLS 1.3)

Cifra de bloco (no modo CBC)

- RC2_40, DES 40, DES_56, 3DES_168, IDEA_128, Fortezza_80, …

Cifra de fluxo (*stream*)

- RC4_40, RC4_128

Se for usada cifra de bloco é realizado *padding*

O último byte do *padding* indica o comprimento do *padding*

TLS 1.3 bans all non-AEAD ciphers

- Current AEAD ciphers for TLS:
  - AES-GCM, AES-CCM, ARIA-GCM, Camellia-GCM, ChaCha/Poly (coming soon)

Algoritmos suportados (Até TLS 1.2 inclusive)

Autenticação

MAC = *hash*(MAC_wr_secret | pad_2 |

** *hash*(MAC_wr_secret | pad_1 | seq_num | type | length | frag))**

- Semelhante ao HMAC mas os *pads* são concatenados
- Funções de *hash* suportadas:
  - **MD5**
  - **SHA-1**

- O pad_1 é **0x36** repetido 48 vezes (MD5) ou 40 vezes (SHA-1)
- O pad_2 is **0x5C** repetido 48 vezes (MD5) ou 40 vezes (SHA-1)

Optimizing Through Optimism

- **TLS 1.2 assumed that the client knew nothing**
  - First round trip mostly consumed by learning server capabilities
- **TLS 1.3 narrows the range of options**
  - Only (EC)DHE
  - Limited number of groups
  - Client can make a good guess at server's capabilities
    - Pick its favorite groups and send a DH share

Sessões e ligações

- Uma sessão SSL é uma associação entre um cliente e um servidor
- As sessões têm estado; o estado de uma sessão inclui algoritmos de segurança e parâmetros
- Uma sessão pode incluir múltiplas ligações seguras entre o mesmo cliente e servidor
- Ligações da mesma sessão partilham o estado da sessão
- As sessões são utilizadas para evitar a dispendiosa negociação de novos parâmetros para cada nova ligação
- Podem existir múltiplas sessões em simultâneo entre as mesmas duas entidades; não é utilizado na prática.

SSL Record Protocol

- **Conceito de sessão**:
  - As sessões são criadas pelo protocolo Handshake.
  - É uma **associação entre cliente e servidor que define um conjunto de parâmetros criptográficos** (algoritmos de cifra e de *hash*, segredo mestre, certificados).
  - **Uma sessão pode suportar múltiplas ligações** para evitar uso repetido do protocolo de *handshake* o qual é dispendioso.

SSL *Record Protocol*

- **Conceito de ligação**:
  - **É uma ligação lógica cliente/servidor cujo estado é definido por *nonces*, chaves secretas para MAC e cifra, IVs (vectores de inicialização), números de sequência**.
  - As chaves utilizadas nas ligações são derivadas de um único segredo mestre criado para a sessão durante o protocolo Handshake.

Estados das sessões e ligações

Estado da sessão

- Identificador da sessão (*session identifier*)
  - Sequência aleatória de bytes escolhida pelo servidor para identificar a sessão
- Certificado dos pares
  - Certificados X.509v3 dos pares, podem ser nulos
- Método de compressão
  - Normalmente é nula (não é utilizada compressão)
- Especificações da cifra
  - Algoritmo de cifra dos dados (ex.: *null*, DES, 3DES, …)
- Algoritmo de *hash* (MAC) (ex.:MD5, SHA-1)
- Atributos criptográficos (ex.: dimensão do *hash*, dimensão do vector de inicialização (IV), …)
- Segredo mestre (*master secret*)
  - segredo a 48 bytes (384 bits) partilhado entre o cliente e o servidor e a partir do qual serão gerados todas as chaves restantes (*key block*)
- – *is resumable*
  - uma *flag* que indica quando a sessão pode ser usada para se iniciarem novas ligações
- Estado das ligações

Estados das sessões e ligações

Estado da ligação

- Números aleatórios do cliente e servidor
  - sequencias aleatórias de bytes escolhidos pelo servidor e pelo cliente para cada ligação
- Segredo do servidor para escrita MAC (*server write MAC secret*)
  - Chave secreta utilizada em operações MAC sobre dados enviados pelo servidor
- Segredo do cliente para escrita MAC (*client write MAC secret*)
  - Chave secreta utilizada em operações MAC sobre dados enviados pelo cliente
- Chave de escrita do servidor
  - Chave secreta de escrita para a cifragem de dados pelo servidor
- Chave de escrita do cliente
  - Chave secreta de escrita para a cifragem de dados pelo cliente
- Vetores de inicialização (IV)
  - Um IV por cada chave de cifra se o modo CBC for usado
  - Inicializado pelo SSL *Handshake Protocol*
- Números de sequência de envio e receção
  - Os números de sequência são a 64 bit
  - Colocados a zero após cada mensagem Change Cipher Spec

SSL Handshake Protocol

- O SSL necessita de chaves simétricas:
  - Para MAC e cifra na camada *Record*.
  - Chaves diferentes em cada direção.
- Estas chaves são criadas como parte do SSL *Handshake Protocol*.
- O SSL *Handshake Protocol* é um protocolo complexo com muitas opções … simplificado no TLS 1.3

SSL Handshake Protocol: Objectivos de segurança

- Autenticação das entidades participantes.
  - Os participantes são designados ‘cliente’ e ‘servidor’.
  - O servidor é sempre autenticado, o cliente nem sempre.
  - Apropriado para a maioria das aplicações de *e-commerce*.
- Estabelecimento de um segredo partilhado, “fresco”.
  - O segredo partilhado é utilizado para derivar novas chaves de sessão para confidencialidade e autenticação no SSL *Record Protocol*.
- Negociação segura do que respeita à segurança.
  - Algoritmos de cifra e autenticação
  - Métodos de estabelecimento de chaves e de autenticação.

SSL Handshake Protocol

Most messages exchanged during the setup of the TLS session are based on this record, unless an error or warning occurs and needs to be signaled by an Alert protocol record, or the encryption mode of the session is modified by another record (see ChangeCipherSpec protocol).

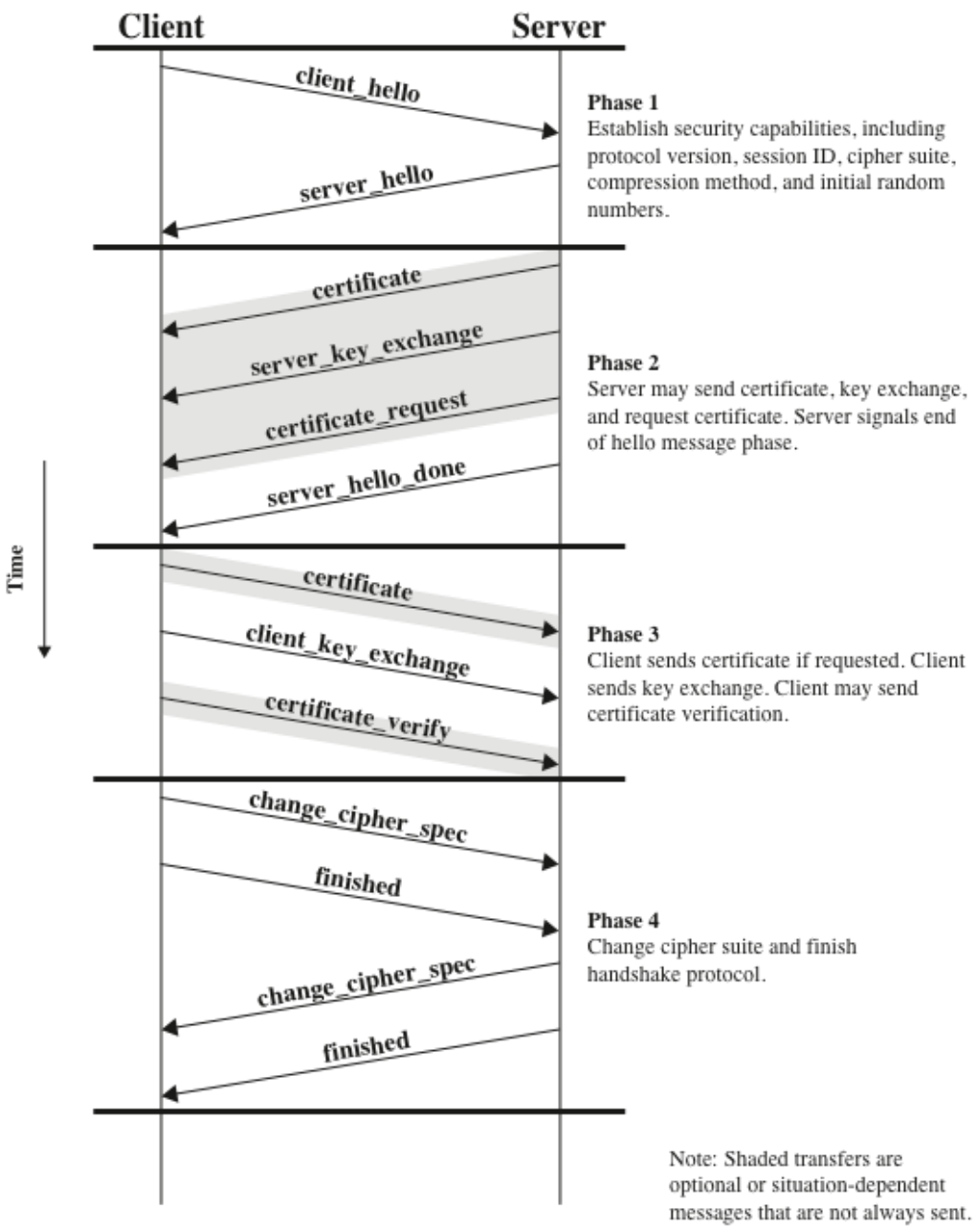
+	Byte +0	Byte +1	Byte +2	Byte +3
Byte 0	22			
Bytes 1..4	Version		Length	
	(Major)	*(Minor)*	*(bits 15..8)*	*(bits 7..0)*
Bytes 5..8	Message type	Handshake message data length		
		(bits 23..16)	*(bits 15..8)*	*(bits 7..0)*
Bytes 9..(*n*−1)	Handshake message data			
Bytes *n*..(*n*+3)	Message type	Handshake message data length		
		(bits 23..16)	*(bits 15..8)*	*(bits 7..0)*
Bytes (*n*+4)..	Handshake message data			

Note that multiple handshake messages may be combined within one record.

SSL Handshake Protocol: Message type

Handshake message types	
Code	**Description**
0	HelloRequest
1	ClientHello
2	ServerHello
4	NewSessionTicket
8	EncryptedExtensions (TLS 1.3 only)
11	Certificate
12	ServerKeyExchange
13	CertificateRequest
14	ServerHelloDone
15	CertificateVerify
16	ClientKeyExchange
20	Finished

Handshake Protocol Action



Cipher Suites

- Os algoritmos criptográficos utilizados são negociados através de *Cipher Suites*, que incluem:
  - Algoritmo para estabelecimento de chaves (ex. cifra assimétrica).
  - Algoritmo para cifra simétrica.
  - Função de *hash* para garantir a integridade dos dados e derivar chaves
- **Exemplos**:
  - SSL_NULL_WITH_NULL_NULL} - algoritmos nulos (usado pelo *Record Protocol* antes do *handshake* ser realizado).
  - SSL_RSA_WITH_3DES_EDE_CBC_SHA
    - cifra assimétrica - RSA;
    - cifra simétrica - 3DES_EDE_CBC;
    - função de *hash* - SHA1.
  - SSL_RSA_EXPORT_WITH_RC4_40_MD5}
    - cifra assimétrica - RSA com chaves de dimensão menor ou igual a 512 bit;
    - cifra simétrica - RC4_40, algoritmo RC4 com chaves de 40 bit;
    - função de *hash* - MD5.

SSL Handshake Protocol: Troca de chaves

- O SSL suporta vários mecanismos de estabelecimento de chaves.
- O mais comum utiliza a cifra RSA.
  - O cliente escolhe o `pre_master_secret`, cifra usando a chave pública do servidor e envia-o para o servidor.
- Também pode criar o `pre_master_secret` a partir de:
  - Diffie-Hellman fixo
    - O certificado do servidor (e possivelmente o do cliente) contém os parâmetros DH.
  - Diffie-Hellman efémero
    - O servidor e o cliente escolhem novos componentes Diffie-Hellman. São trocada chaves efémeras assinadas, necessita de chaves para assinar e de certificados em ambos os lados
  - Diffie-Hellman anónimo
    - Cada lado envia valores Diffie-Hellman, mas sem autenticação.
    - Vulnerável a ataques *man-in-middle*.

SSL Handshake Protocol: Autenticação da entidade

- O SSL suporta vários mecanismos de autenticação de entidades.
- O mais comum é baseado no RSA.
  - A habilidade para decifrar o *pre_master_secret* e gerar o MAC correcto a partir de mensagens, utilizando chaves derivadas do *pre_master_secret,* **autentica o servidor perante o cliente**.
- Menos comum: Assinaturas DSS ou RSA e *nonces* (e outros campos, e.g. valores Diffie-Hellman).

SSL: Derivação de chaves

- As chaves usadas para autenticação e para a cifra na *Record Layer* são derivadas do ***pre_master_secret***:

  - Derivar o ***master_secret*** do *pre_master_secret* e de *nonces* dos clientes/servidores usando funções de *hash* MD5 e SHA-1.
  - Derivar o material para a chave ***key_block*** do *master_secret* e dos *nonces* cliente/servidor, através do uso repetido de funções de *hash*.
  - Dividir o *key_block* nas chaves para o MAC e para a cifra para o *Record Protocol* conforme necessário.

Pre master secret

- **48 bytes de dimensão**

- **Obtida via:**
  - **RSA**
    - Gerado pelo cliente
    - Enviado cifrado com a chave pública do servidor e enviado para este
  - **DH**
    - Gerado via Diffie-Hellman

SSL: Derivação de chaves

SSL: **master_secret** = **MD5**(pre_master_secret +
SHA('A' + pre_master_secret + ClientHello.random + ServerHello.random)) +
MD5(pre_master_secret + **SHA**('BB' + pre_master_secret +
ClientHello.random + ServerHello.random)) +
MD5(pre_master_secret + SHA('CCC' + pre_master_secret +
ClientHello.random + ServerHello.random))

TLS: **master_secret** = **PRF**(pre_master_secret, "master secret", ClientHello.random + ServerHello.random) [0..47];

- Pre_master_secret: 48 bytes
- A PRF é implementada usando HMAC

Key block

- **SSLv3**
- **key_block** =

  MD5(master_secret + SHA(`A' + master_secret + ServerHello.random + ClientHello.random)) +

  MD5(master_secret + SHA(`BB' + master_secret + ServerHello.random + ClientHello.random)) +

  MD5(master_secret + SHA(`CCC' + master_secret + ServerHello.random + ClientHello.random)) + [...];

- **TLS:**

  PRF(secret, label,seed) = P_**MD5**(S1, label + seed) XOR P_**SHA-1**(S2, label + seed);

  master_secret = **PRF**(pre_master_secret, "master secret", ClientHello.random + ServerHello.random) [0..47];

  key_block = **PRF**(SecurityParameters.master_secret, "key expansion", SecurityParameters.server_random + SecurityParameters.client_random)

SSL *Handshake Protocol*: Funcionamento

- Escolheu-se o uso mais comum do SSL:
  - Sem autenticação do cliente.
  - O cliente envia o *pre_master_secret* usando a chave pública de cifra RSA do certificado do servidor.
  - O servidor é autenticado pela habilidade de decifrar para obter o *pre_master_secret* e construir a mensagem final correta.

SSL *Handshake Protocol*: Funcionamento

Sumário:

M1: C → S: `ClientHello`

M2: S → C: `ServerHello, ServerCertChain,ServerHelloDone`

M3: C → S: `ClientKeyExchange, ChangeCipherSpec, ClientFinished`

M4: S → C: `ChangeCipherSpec, ServerFinished`

SSL *Handshake Protocol*: Funcionamento

M1: C → S: `ClientHello`

- O cliente inicia a ligação.
- Envia o número da versão do cliente.
  - 3 (*major*) e 0 (*minor*) para o SSL; 3 e 1 ou 3 e 2 para o TLS.
- Envia o `ClientNonce`.
  - 32 bytes; 4 bytes do tempo mais 28 bytes aleatórios.
- Oferece uma lista de parâmetros de cifra à escolha (*ciphersuites*).
  - Opções para troca de chave e autenticação, algoritmos de cifra, funções de *hash*.
  - Ex.. `TLS_RSA_WITH_3DES_EDE_CBC_SHA`.

SSL *Handshake Protocol*: Funcionamento

M2: S → C: `ServerHello, ServerCertChain, ServerHelloDone`

- Envia o número de versão do servidor.
- Envia o `ServerNonce` e o `SessionID`.
- Selecciona uma conjunto de parâmetros de cifra da lista oferecida pelo cliente.
  - Ex. `TLS_RSA_WITH_3DES_EDE_CBC_SHA`.
- Envia a mensagem `ServerCertChain`.
  - Permite ao cliente validar a chave pública do servidor através da análise até ao *root of trust* (CA).
- Mensagem (opcional) `CertRequest`.
  - Omitida neste exemplo – não há autenticação do cliente.
- Finalmente, `ServerHelloDone`.

SSL *Handshake Protocol*: Funcionamento

M3: C → S: `ClientKeyExchange, ChangeCipherSpec, ClientFinished`

- `ClientKeyExchange` contém o `pre_master_secret` cifrado utilizando a chave pública do servidor.
- `ChangeCipherSpec` indica quais os parâmetros de cifra que o cliente suporta e que podem ser usados nesta sessão.
  - Enviado o SSL *Change Cipher Spec. Protocol*.
- Mensagens (opcional) `ClientCertificate`, `ClientCertificateVerify`.
  - Apenas quando o cliente é autenticado.
- Finalmente, a mensagem `ClientFinished`.
  - **Um MAC de todas as mensagens trocadas entre ambos os lados**.
  - **O MAC é calculado usando o `master_secret`.**

SSL *Handshake Protocol*: Funcionamento

M4: S → C: `ChangeCipherSpec, ServerFinished`

- `ChangeCipherSpec` indica que o servidor está a modificar os parâmetros de cifra a serem usados nesta sessão de acordo com os suportados pelo cliente.
  - Enviado usando SSL *Change Cipher Spec. Protocol*.
- Finalmente, a mensagem `ServerFinished`.
  - **Um MAC de todas as mensagens trocadas entre ambos os lados.**
  - **O MAC é calculado usando o `master_secret`.**
  - **O servidor pode apenas calcular o MAC se conseguir decifrar o `pre_master_secret` recebido em M3.**

SSL Handshake Protocol: Funcionamento

- **O cliente foi autenticado perante o servidor?**

  Não!

- **Um adversário pode aprender o valor do pre_master_secret?**

  Não! O cliente validou a chave pública do servidor; Só o possuidor da chave privada é que consegue decifrar o ClientKeyExchange para aprender a *pre_master_secret*.

- **O servidor é autenticado perante o cliente?**

  Sim! O ServerFinished inclui os MAC dos *nonces* calculados usando a chave derivada do *pre_master_secret*.

Outros exemplos de funcionamento do SSL *Handshake Protocol*

- Muitas opções/situações dependem de mensagens do protocolo:
  - M2 (S→C) pode incluir:
    - `ServerKeyExchange` (ex. para a troca de chaves DH).
    - `CertRequest` (para a autenticação do cliente).
  - M3 (C→S) pode incluir:
    - `ClientCert` (para autenticação do cliente),
    - `ClientCertVerify` (para autenticação do cliente).

- Para detalhes ver RFC 2246 (TLS).

SSL *Handshake Protocol*: Características adicionais

- O SSL *Handshake Protocol* suporta ***session resumption*** e ***ciphersuite re-negotiation***.
  - Permite que a autenticação e os segredos partilhados possam ser reusados através de múltiplas ligações.
    - Ex. próxima página Web do mesmo *site*.
  - Permite nova troca de chaves (*re-keying*) da ligação actual utilizando novos *nonces*.
  - Permite troca de parâmetros de cifra durante a sessão.
  - `ClientHello` cita o anterior `SessionID`.
  - Ambos os lados contribuem com novos *nonces*. Actualizam o `master_secret` e o `key_block`.
  - Tudo isto protegido pelo *Record Protocol*.

Outros protocolos do SSL

- ***Alert protocol***
  - Gestão da sessão SSL, mensagens de erro.
  - Erros fatais e avisos.
- ***Change cipher spec protocol***
  - É um protocolo separado do *SSL Handshake Protocol*.
  - Usado para indicar que uma entidade está a mudar para um conjunto de parâmetros acordados recentemente.
- ***Heartbeat protocol***
  - Mantém tráfego na ligação

- Os protocolos acima correm sobre o *Record Protocol* (tal como o *Handshake Protocol*).

SSL Alert Protocol

Cada mensagem deste protocolo consiste em dois campos (bytes)

- Primeiro campo: “warning” ou “fatal”
- Segundo campo (byte):
  - fatal
    - unexpected_message
    - bad_record_MAC
    - decompression_failure
    - handshake_failure
    - illegal_parameter
  - warning
    - close_notify
    - no_certificate
    - bad_certificate
    - unsupported_certificate
    - certificate_revoked
    - certificate_expired
    - certificate_unknown

No caso de um alerta fatal

- A ligação é terminada
- O session ID é invalidado
- Nenhuma outra sessão/ligação pode ser estabelecida no âmbito desta sessão

SSL Alert Protocol

This record should normally not be sent during normal handshaking or application exchanges. However, this message can be sent at any time during the handshake and up to the closure of the session. If this is used to signal a fatal error, the session will be closed immediately after sending this record, so this record is used to give a reason for this closure. If the alert level is flagged as a warning, the remote can decide to close the session if it decides that the session is not reliable enough for its needs (before doing so, the remote may also send its own signal).

+	Byte +0	Byte +1	Byte +2	Byte +3
Byte 0	21			
Bytes 1..4	Version		Length	
	(Major)	*(Minor)*	0	2
Bytes 5..6	Level	Description		
Bytes 7..(p−1)	MAC (optional)			
Bytes p..(q−1)	Padding (block ciphers only)			

SSL Alert Protocol

Level

This field identifies the level of alert. If the level is fatal, the sender should close the session immediately. Otherwise, the recipient may decide to terminate the session itself, by sending its own fatal alert and closing the session itself immediately after sending it. The use of Alert records is optional, however if it is missing before the session closure, the session may be resumed automatically (with its handshakes).

Normal closure of a session after termination of the transported application should preferably be alerted with at least the *Close notify*Alert type (with a simple warning level) to prevent such automatic resume of a new session. Signalling explicitly the normal closure of a secure session before effectively closing its transport layer is useful to prevent or detect attacks (like attempts to truncate the securely transported data, if it intrinsically does not have a predetermined length or duration that the recipient of the secured data may expect).

Alert level types

Code	Level type	Connection state
1	warning	Connection or security may be unstable.
2	fatal	Connection or security may be compromised, or an unrecoverable error has occurred.

SSL Alert Protocol

Description - Alert description types

This field identifies which type of alert is being sent.

Alert description types			
Code	**Description**	**Level types**	**Note**
0	Close notify	warning/fatal	
10	Unexpected message	fatal	
20	Bad record MAC	fatal	Possibly a bad SSL implementation, or payload has been tampered with e.g. FTP firewall rule on FTPS server.
21	Decryption failed	fatal	TLS only, reserved
22	Record overflow	fatal	TLS only
30	Decompression failure	fatal	
40	Handshake failure	fatal	
41	No certificate	warning/fatal	SSL 3.0 only, reserved
42	Bad certificate	warning/fatal	
43	Unsupported certificate	warning/fatal	e.g. certificate has only Server authentication usage enabled and is presented as a client certificate
44	Certificate revoked	warning/fatal	
45	Certificate expired	warning/fatal	Check server certificate expire also check no certificate in the chain presented has expired
46	Certificate unknown	warning/fatal	
47	Illegal parameter	fatal	

SSL Alert Protocol

Description - Alert description types (cont.)

This field identifies which type of alert is being sent.

Alert description types			
Code	**Description**	**Level types**	**Note**
48	Unknown CA (Certificate authority)	**fatal**	TLS only
49	Access denied	**fatal**	TLS only – e.g. no client certificate has been presented (TLS: Blank certificate message or SSLv3: No Certificate alert), but server is configured to require one.
50	Decode error	**fatal**	TLS only
51	Decrypt error	**warning**/**fatal**	TLS only
60	Export restriction	**fatal**	TLS only, reserved
70	Protocol version	**fatal**	TLS only
71	Insufficient security	**fatal**	TLS only
80	Internal error	**fatal**	TLS only
86	Inappropriate Fallback	**fatal**	TLS only
90	User canceled	**fatal**	TLS only
100	No renegotiation	**warning**	TLS only
110	Unsupported extension	**warning**	TLS only
111	Certificate unobtainable	**warning**	TLS only

SSL Alert Protocol

Description - Alert description types (cont.)

This field identifies which type of alert is being sent.

Alert description types			
Code	**Description**	**Level types**	**Note**
112	Unrecognized name	warning/fatal	TLS only; client's Server Name Indicator specified a hostname not supported by the server
113	Bad certificate status response	fatal	TLS only
114	Bad certificate hash value	fatal	TLS only
115	Unknown PSK identity (used in TLS-PSK and TLS-SRP)	fatal	TLS only
120	No Application Protocol	fatal	TLS only, client's ALPN did not contain any server-supported protocols

Mensagens de alerta do SSL

- close_notify(0),
- unexpected_message(10),
- bad_record_mac(20),
- decryption_failed(21),
- record_overflow(22),
- decompression_failure(30),
- handshake_failure(40),
- bad_certificate(42),
- unsupported_certificate(43),
- certificate_revoked(44),
- certificate_expired(45),
- certificate_unknown(46),

illegal_parameter(47),
unknown_ca(48),
access_denied(49),
decode_error(50),
decrypt_error(51),
export_restriction(60),
protocol_version(70),
insufficient_security(71),
internal_error(80),
user_canceled(90),
no_renegotiation(100),

SSL ChangeCipherSpec protocol

+	Byte +0	Byte +1	Byte +2	Byte +3
Byte 0	20			
Bytes 1..4	Version		Length	
	(Major)	*(Minor)*	0	1
Byte 5	CCS protocol type			

CCS protocol type
Currently only 1.

SSL Heartbeat protocol

Is a periodic signal generated by hardware or software to indicate normal operation or to synchronize other parts of a system

Typically used to monitor the availability of a protocol entity

In the specific case of TLS, a **Heartbeat protocol was defined in 2012 in RFC 6250** (Transport Layer Security (T L S) and Datagram Transport Layer Security (DTLS) Heartbeat Extentsion)

Runs on top of the T L S Record Protocol

Consists of two message types:

- heartbeat_request
- heartbeat_response

SSL Heartbeat protocol

The use of the Heartbeat protocol is established during Phase 1 of the Handshake protocol

The heartbeat serves two purposes:

- It assures the sender that the recipient is still alive.
- The heartbeat generates activity across the connection during idle periods, which avoids closure by a firewall that does not tolerate idle connections

The requirement for the exchange of a payload was designed into the Heartbeat protocol to support its use in a connectionless version of TLS known as Datagram Transport Layer Security (DTLS)

SSL Application protocol

+	Byte +0	Byte +1	Byte +2	Byte +3
Byte 0	23			
Bytes 1..4	Version		Length	
	(Major)	*(Minor)*	*(bits 15..8)*	*(bits 7..0)*
Bytes 5..(*m*−1)	Application data			
Bytes *m*..(*p*−1)	MAC (optional)			
Bytes *p*..(*q*−1)	Padding (block ciphers only)			

Length

Length of application data (excluding the protocol header and including the MAC and padding trailers)

MAC

20 bytes for the SHA-1-based HMAC, 16 bytes for the MD5-based HMAC.

Padding

Variable length; last byte contains the padding length.

Portos por protocolos que utilizam o SSL

- https 443
- ssmtp 465
- snntp 563
- sldap 636
- spop3 995
- ftp-data 889
- ftps 990
- imaps 991
- telnets 992
- ircs 993

Captura

Frame 15 (156 bytes on wire, 156 bytes captured)
Internet Protocol CLIENT ==>>> SERVER
Secure Socket Layer
SSLv3 Record Layer: Client Hello
Content Type: Handshake (22)
Version: SSL 3.0 (0x0300)
Length: 97
Handshake Protocol: Client Hello
Handshake Type: Client Hello (1)
Length: 93
Version: SSL 3.0 (0x0300)
Random.gmt_unix_time: Jul 20, 2030 16:25:01.000000000
Random.bytes
Session ID Length: 32
Session ID (32 bytes)
Cipher Suites Length: 22
Cipher Suites (11 suites)
Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
Cipher Suite: TLS_RSA_WITH_DES_CBC_SHA (0x0009)
Cipher Suite: TLS_RSA_EXPORT1024_WITH_RC4_56_SHA (0x0064)
Cipher Suite: TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA (0x0062)
Cipher Suite: TLS_RSA_EXPORT_WITH_RC4_40_MD5 (0x0003)
Cipher Suite: TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5 (0x0006)
Cipher Suite: TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA (0x0013)
Cipher Suite: TLS_DHE_DSS_WITH_DES_CBC_SHA (0x0012)
Cipher Suite: TLS_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA (0x0063)
Compression Methods Length: 1
Compression Methods (1 method)
Compression Method: null (0)

Frame 17 (1427 bytes on wire, 1427 bytes captured)
Internet Protocol CLIENT <<<=== SERVER
Secure Socket Layer
SSLv3 Record Layer: Multiple Handshake Messages
Content Type: Handshake (22)
Version: SSL 3.0 (0x0300)
Length: 1368
Handshake Protocol: Server Hello
Handshake Type: Server Hello (2)
Length: 70
Version: SSL 3.0 (0x0300)
Random.gmt_unix_time: Not representable
Random.bytes
Session ID Length: 32
Session ID (32 bytes)
Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
Compression Method: null (0)
Handshake Protocol: Certificate
Handshake Type: Certificate (11)
Length: 1286
Certificates Length: 1283
Certificates (1283 bytes)
Certificate Length: 720
Certificate (720 bytes)
Certificate Length: 557
Certificate (557 bytes)
Handshake Protocol: Server Hello Done
Handshake Type: Server Hello Done (14)
Length: 0

Frame 18 (258 bytes on wire, 258 bytes captured)
Internet Protocol CLIENT ==>>> SERVER
Secure Socket Layer
SSLv3 Record Layer: Client Key Exchange
Content Type: Handshake (22)
Version: SSL 3.0 (0x0300)
Length: 132
Handshake Protocol: Client Key Exchange
Handshake Type: Client Key Exchange (16)
Length: 128
SSLv3 Record Layer: Change Cipher Spec
Content Type: Change Cipher Spec (20)
Version: SSL 3.0 (0x0300)
Length: 1
Change Cipher Spec Message
SSLv3 Record Layer: Encrypted Handshake Message
Content Type: Handshake (22)
Version: SSL 3.0 (0x0300)
Length: 56
Handshake Protocol: Encrypted Handshake Message

Frame 23 (115 bytes on wire, 115 bytes captured)
Internet Protocol CLIENT <<<=== SERVER
Secure Socket Layer
SSLv3 Record Layer: Encrypted Handshake Message
Content Type: Handshake (22)
Version: SSL 3.0 (0x0300)
Length: 56
Handshake Protocol: Encrypted Handshake Message

SSL e TLS

TLS 1.0 = SSL 3.0 com diferenças menores.

- Uso do HMAC como algoritmo MAC.
- Métodos diferentes de derivar o material das chaves (`master-secret` e `key-block`).
  - Função geração de números pseudo-aleatórios baseados no HMAC com MD5 e SHA-1.
- Códigos adicionais de alerta.
- Mais tipos de certificados de clientes.
- *Padding* de dimensão variável.
  - Pode ser usado para esconder o comprimento de mensagens pequenas e frustrar assim a análise de tráfego.
- E mais ….

SSL/TLS: Aplicações

- *E-commerce* seguro usando SSL/TLS.
  - A autenticação do cliente não é necessária até o cliente decidir comprar alguma coisa.
  - O SSL fornece um canal seguro para enviar informação dos cartões de crédito, detalhes pessoais, etc.
  - O cliente pode ser autenticado através da informação do cartão de crédito, o comerciante vê assim diminuído (a maior parte) do risco.
  - Muito sucesso (amazon.com, supermercados *on-line*, companhias aéreas, …)

SSL/TLS: Aplicações

- *E-commerce* seguro: Alguns pontos.
  - Não dá garantias sobre o que acontece aos dados do cliente (incluindo os detalhes dos cartões de crédito) depois da sessão: podem ser guardados num servidor inseguro.
  - O cliente entende o significado da expiração do certificado e outros avisos de segurança?
  - O software do cliente testa toda a cadeia de certificados?
  - O nome no certificado condiz com o URL do *site* de *e-commerce*? O utilizador verifica isto?
  - O *site* é aquele que o cliente pensa que é?
  - O software do cliente propõe os parâmetros de cifra adequados?

SSL/TLS: Aplicações

- **Banca electrónica segura.**
  - Autenticação do cliente pode ser possível usando certificados de cliente.
    - Pormenores de registo, armazenamento seguro das chaves privadas, revogação.
  - De outra forma o SSL fornece um canal seguro para o envio de nome, *password*, nome do meio da mãe, …
    - Para que é que o cliente utiliza a mesma *password*?
  - O cliente percebe o significado de expiração do certificado e outros avisos de segurança?
  - O software do cliente propõe os parâmetros de segurança apropriados?
    - Forçados pelo servidor..

Algumas falhas na segurança do SSL/TLS

- (Histórica) falhas na geração de números aleatórios para o SSL.
  - Baixa qualidade dos RNG leva a chaves de sessão previsíveis.
  - Goldberg and Wagner, Dr. Dobb's Journal, Jan. 1996.
  - http://www.ddj.com/documents/s=965/ddj9601h/
- Falhas no relatar de erros.
  - (tempos de resposta diferentes pelo servidor no caso de falha no *padding* e no MAC) + (análise do método de *padding* para o modo CBC) = recuperação do texto em claro do SSL.
  - Canvel, Hiltgen, Vaudenay and Vuagnoux, Crypto2003.
  - http://lasecwww.epfl.ch/php_code/publications/search.php?ref=CHVV03
- Ataques temporais (*timing attacks*).
  - Análise dos tempos de resposta do servidor OpenSSL permite aos atacantes no mesmo segmento de LAN derivar a chave privada do servidor
    - Boneh and Brumley, 12th Usenix Security Symposium, 2003.
    - http://crypto.stanford.edu/~dabo/abstracts/ssl-timing.html

TLS 1.3 Backwards compatibility

- TLS 1.3 **abandons backwards compatibility** in favor of a proper security design. It has been designed from scratch to provide functionality similar (yet not compatible) to TLS 1.2, but with significantly improved performance, privacy and security.

SSL/TLS 1.3 – Introduction

The primary goal of TLS is to provide a secure channel between two communicating peers; the only requirement from the underlying transport is a reliable, in-order data stream. Specifically, the secure channel should provide the following properties:

- **Authentication**: The server side of the channel is always authenticated; the client side is optionally authenticated. Authentication can happen via asymmetric cryptography (e.g., RSA [RSA], the Elliptic Curve Digital Signature Algorithm (ECDSA) [ECDSA], or the Edwards-Curve Digital Signature Algorithm (EdDSA) [RFC8032]) or a symmetric pre-shared key (PSK).
- **Confidentiality**: Data sent over the channel after establishment is only visible to the endpoints. TLS does not hide the length of the data it transmits, though endpoints are able to pad TLS records in order to obscure lengths and improve protection against traffic analysis techniques.
- **Integrity**: Data sent over the channel after establishment cannot be modified by attackers without detection.

These properties should be true even in the face of an attacker who has complete control of the network, as described in [RFC3552].

TLS 1.3

TLS consists of two primary components:

- A handshake protocol that authenticates the communicating parties, negotiates cryptographic modes and parameters, and establishes shared keying material.
  - The handshake protocol is designed to resist tampering; an active attacker should not be able to force the peers to negotiate different parameters than they would if the connection were not under attack.
- A record protocol that uses the parameters established by the handshake protocol to protect traffic between the communicating peers.
  - The record protocol divides traffic up into a series of records, each of which is independently protected using the traffic keys.

TLS 1.2 vs TLS 1.3

TLS 1.3 is a major overhaul and has two main advantages over previous versions:

- Enhanced security
  - TLS 1.3 now **removes obsolete and insecure features from TLS 1.2**, including the following:
    - RSA key transport — Doesn't provide forward secrecy
    - CBC mode ciphers — Responsible for BEAST, and Lucky 13
    - RC4 stream cipher — Not secure for use in HTTPS
    - SHA-1 hash function — Deprecated in favor of SHA-2
    - Arbitrary Diffie-Hellman groups — CVE-2016-0701
    - EXPORT-strength ciphers – Responsible for FREAK and LogJam
- Improved speed
  - TLS 1.2 needs two round-trips to complete the TLS handshake. TLS 1.3 requires only one round-trip. This cuts the encryption latency in half.
  - TLS 1.3 remembers! On sites visited previously, it is possible now to send data on the first message to the server. This is called a **"zero round trip." (0-RTT)**. The result is improved load time times.

TLS 1.3 Major Differences from TLS 1.2 (1/2)

The list of supported symmetric encryption algorithms has been pruned of all algorithms that are considered legacy. Those that remain are all Authenticated Encryption with Associated Data (AEAD) algorithms. The cipher suite concept has been changed to separate the authentication and key exchange mechanisms from the record protection algorithm (including secret key length) and a hash to be used with both the key derivation function and handshake message authentication code (MAC).

A **zero round-trip time (0-RTT) mode was added**, saving a round trip at connection setup for some application data, at the cost of certain security properties.

Static RSA and Diffie-Hellman cipher suites have been removed; all public-key based key exchange mechanisms now provide forward secrecy.

All handshake messages after the ServerHello are now encrypted. The newly introduced EncryptedExtensions message allows various extensions previously sent in the clear in the ServerHello to also enjoy confidentiality protection.

The key derivation functions have been redesigned. The new design allows easier analysis by cryptographers due to their improved key separation properties. The HMAC-based Extract-and-Expand Key Derivation Function (HKDF) is used as an underlying primitive.

TLS 1.3 Major Differences from TLS 1.2 (2/2)

The handshake state machine has been significantly restructured to be more consistent and to remove superfluous messages such as ChangeCipherSpec (except when needed for middlebox compatibility).

Elliptic curve algorithms are now in the base spec, and new signature algorithms, such as EdDSA, are included. TLS 1.3 removed point format negotiation in favor of a single point format for each curve.

Other cryptographic improvements were made, including changing the RSA padding to use the RSA Probabilistic Signature Scheme (RSASSA-PSS), and the removal of compression, the Digital Signature Algorithm (DSA), and custom Ephemeral Diffie-Hellman (DHE) groups.

The TLS 1.2 version negotiation mechanism has been deprecated in favor of a version list in an extension. This increases compatibility with existing servers that incorrectly implemented version negotiation.

Session resumption with and without server-side state as well as the PSK-based cipher suites of earlier TLS versions have been replaced by a single new PSK exchange.

ECC Key Exchange Algorithms (rfc4492/TLS 1.3)

The table below summarizes the new key exchange algorithms, which mimic DH_DSS, DHE_DSS, DH_RSA, DHE_RSA, and DH_anon, respectively. TLS 1.3 only uses predefined groups.

Key Exchange Algorithm	Description
ECDH_ECDSA	Fixed ECDH with ECDSA-signed certificates.
ECDHE_ECDSA	Ephemeral ECDH with ECDSA signatures.
ECDH_RSA	Fixed ECDH with RSA-signed certificates.
ECDHE_RSA	Ephemeral ECDH with RSA signatures.
ECDH_anon	Anonymous ECDH, no signatures.

TLS 1.3 Server Certificate Types/Key Exchange Algorithm

Server Certificate	Type
ECDH_ECDSA	Certificate MUST contain an ECDH-capable public key. It MUST be signed with ECDSA.
ECDHE_ECDSA	Certificate MUST contain an ECDSA-capable public key. It MUST be signed with ECDSA.
ECDH_RSA	Certificate MUST contain an ECDH-capable public key. It MUST be signed with RSA.
ECDHE_RSA	Certificate MUST contain an RSA public key authorized for use in digital signatures. It MUST be signed with RSA.

TLS 1.2 performance issues

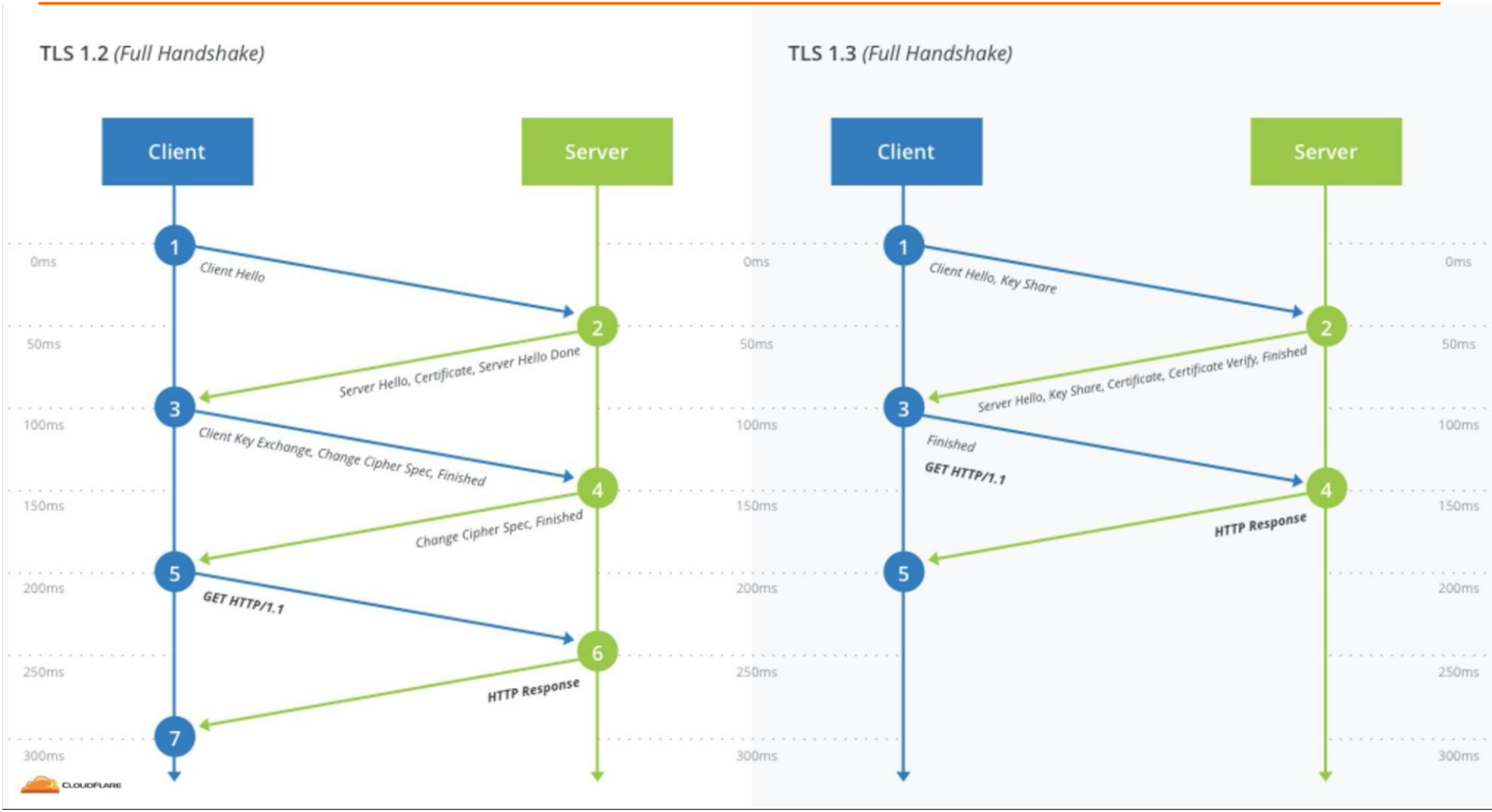
- TLS 1.2's need to negotiate numerous TLS parameters that can impose a performance overhead on HTTPS (or other TLS protected) communications.

- TLS 1.2's 4-step handshake requires two round-trip exchanges, first to select the cipher-suite, and then to exchange the certificates and symmetric keys (or key shares).

- For every TLS 1.2's connection to be established, **two additional transactions** with the server are required. As a result, TLS connections require more bandwidth and power than (unencrypted) HTTP, which can be particularly costly to Internet-of-Things (IoT) applications, where low power and bandwidth consumption are hard constraints.

- **This usually represents more 100 ms to establish a connection with TLS 1.2 than with TLS 1.3.** It is worse with long connections (e.g. far web sites) or when using wireless connections (e.g. 4G or 3G).

TLS1.2 vs TLS 1.3 Handshake



[https://blog.cloudflare.com/introducing-tls-1-3/]

TLS 1.3 performance (1/2)

- Besides improved security, the reduced set of parameters and the simplified handshake in TLS 1.3 also contributes to improving overall performance.
  - Since there is only one key exchange algorithm (with baked-in parameters) and just a handful of supported ciphers, the absolute bandwidth required to set up a TLS 1.3 channel is considerably less than earlier versions.
- **TLS 1.3 now supports a new handshake protocol, called *1-RTT mode*.**
  - **In 1-RTT, the client can send DH key shares in the first handshake message, because it can be fairly certain of the parameters the server will use**. In the rare case that the server does not support them, it can produce an error so that the client will send a different configuration.
  - Instead of negotiating the parameters first and then exchanging keys or key shares, TLS 1.3 allows a client to set up a TLS channel with only one round-trip transaction (instead of two, as it was previously done). This can have a great cumulative effect in the processing, power, and network resources that are required for a client to communicate with a server over TLS 1.3.

TLS 1.3 performance (2/2)

- **TLS 1.3 feature, called the *0-RTT Resumption mode*.**
  - **When a browser visits a server for the first time and successfully completes a TLS handshake, both the client and the server can store a pre-shared encryption key locally. If the browser visits the server again, it can use this resumption key to send encrypted application data in its first message to the server. This has the same latency as unencrypted HTTP, because initial handshakes are not required.**
- It should be noted that there have been some security concerns about 0-RTT mode in the past.

TLS 1.2 Security concerns

- There have been many concerns about TLS security, but one of the most impactful was the realization that TLS 1.2 (and all earlier versions, including SSL) is vulnerable to downgrade attacks due to a flaw in its handshake protocol design. More specifically, **TLS 1.2 does not use digital signatures to protect the integrity of the handshake**. Signatures protect part of the handshake, only after the cipher-suite negotiation. As a consequence, attackers can manipulate any third-party cipher-suite negotiation that occurs in the same computer network (e.g. airport wifi), and force the use of an insecure cipher. They can then break that vulnerable cipher and gain unwarranted access to the entire conversation.

TLS 1.2 privacy issues

- TLS 1.2 has been criticized for compromising web user privacy.
- TLS offers an extension known as ***Server Name Indication* or SNI**. SNI allows the hostname of a server to be included in the initial SSL handshake. This extension is used for virtual hosting, where servers may serve multiple domains on the same IP address and port, while presenting a different certificate for each domain.
- **In TLS 1.2, SNIs are sent unencrypted**, so despite the use of HTTPS, a network attacker can leak this information and track the web pages a user visits.

TLS 1.3 Security

- A core tenet of TLS 1.3 is simplicity. **All key exchange algorithms, except the *Diffie-Hellman* (DH) key exchange, were removed. TLS 1.3 has also defined a set of tried and tested DH parameters, eliminating the need to negotiate parameters with the server.**
- **TLS 1.3 no longer supports unnecessary or vulnerable ciphers, such as CBC-mode and the RC4 cipher.**
  - These ciphers are known to be susceptible to attacks, but were still supported in most TLS implementations for legacy compatibility. Fortunately, the recent rush of downgrade attacks affecting early TLS versions motivated IETF to entirely remove such ciphers from TLS 1.3.
- **TLS 1.3 requires servers to cryptographically sign the entire handshake, including the cipher negotiation**, which prevents attackers from modifying any handshake parameters.
  - TLS 1.3 is architecturally impervious to the downgrade attacks that affected earlier TLS versions.
- The signatures themselves were also improved by implementing a new standard, called RSA-PSS. RSA-PSS signatures are immune to cryptographic attacks affecting the signature schemes employed in earlier TLS versions.

[https://security.stackexchange.com/questions/183179/what-is-rsa-oaep-rsa-pss-in-simple-terms]