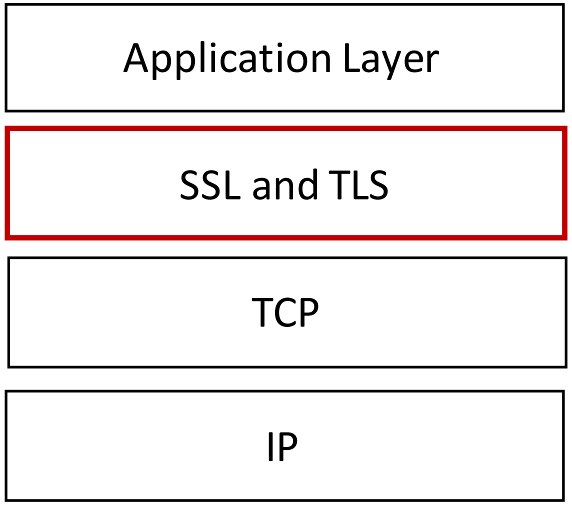
IT-Security 1

Chapter 9: SSL/TLS

Prof. Dr.-Ing. Ulrike Meyer

WS 15/16

SSL and TLS

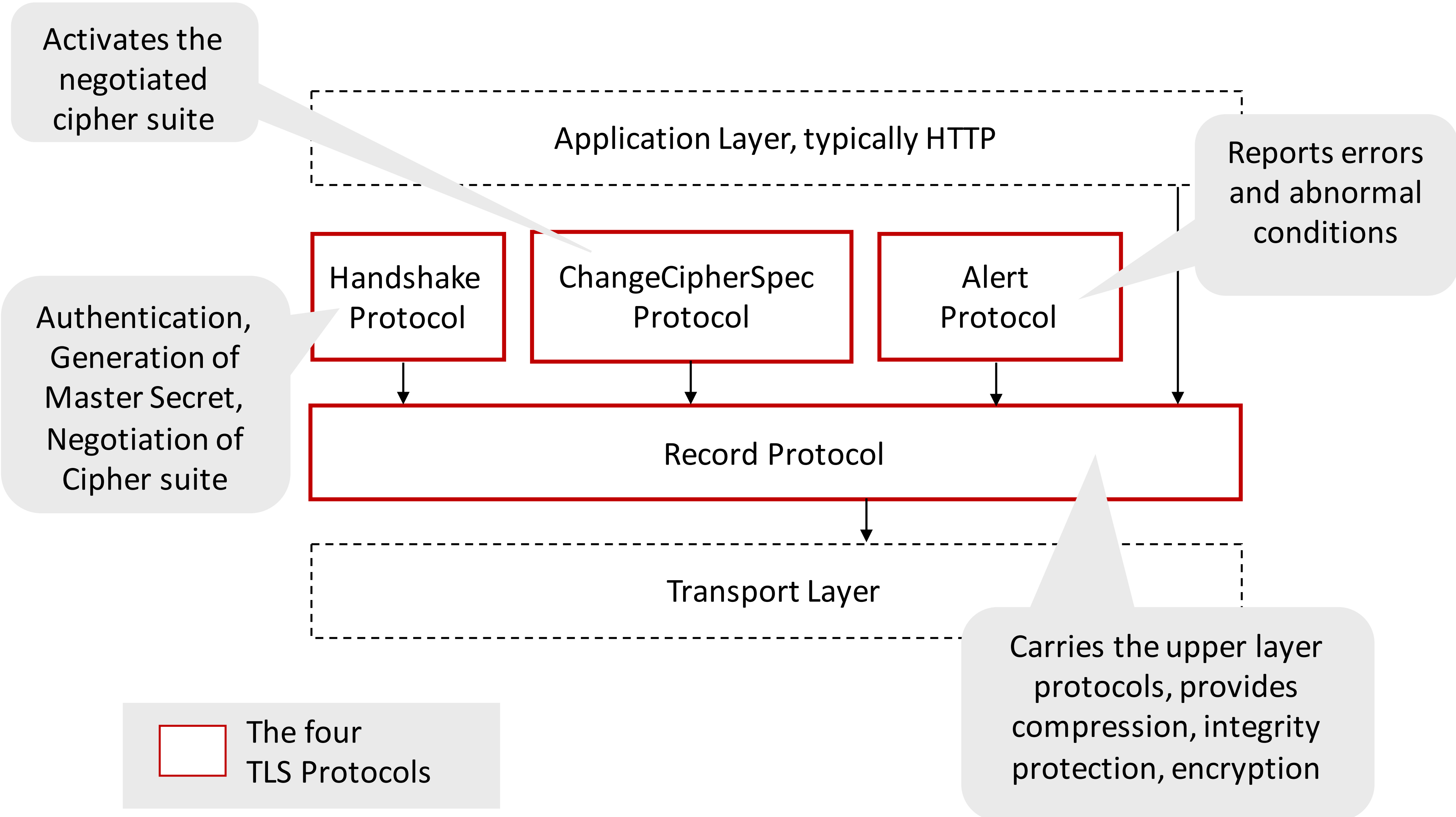


- Protocol suite for
  - Server authentication only or server and client authentication
  - Data integrity and optional confidentiality on the transport layer between client and server
  - Session-based protection that can be invoked from application (e.g. https)

SSL vs. TLS

- SSL = Secure Socket Layer
  - First version developed by Netscape in 1994
  - Latest version: v3
- TLS = Transport Layer Security
  - Standardized version of SSL
  - Standardized by the Internet Engineering Task Force (IETF)
  - Latest version: RFC 5246 *The Transport Layer Security (TLS) Protocol Version 1.2*
  - Updated e.g. by RFC 7465 which forbids the use of RC4
  - Version 1.0 backward compatible with SSL v3
- Here we will discuss:
  - TLS 1.0 as basis (in 2013 nearly 80% of all webservers still only supported 1.0 and all major browsers supported 1.0 only!)
  - Changes in TLS 1.1 (minor) and TLS 1.2 (bigger but not fundamental)

TLS Architecture



Connections and Sessions

- A TLS Session
  - Is an association between a client and a server
  - Is created by the handshake protocol
  - Defines a set of security parameters
  - May be shared by multiple connections

- A TLS Connection is
  - a transport layer connection that provides a suitable type of service
  - is a peer-to-peer relationship
  - is transient
  - is associated with a session

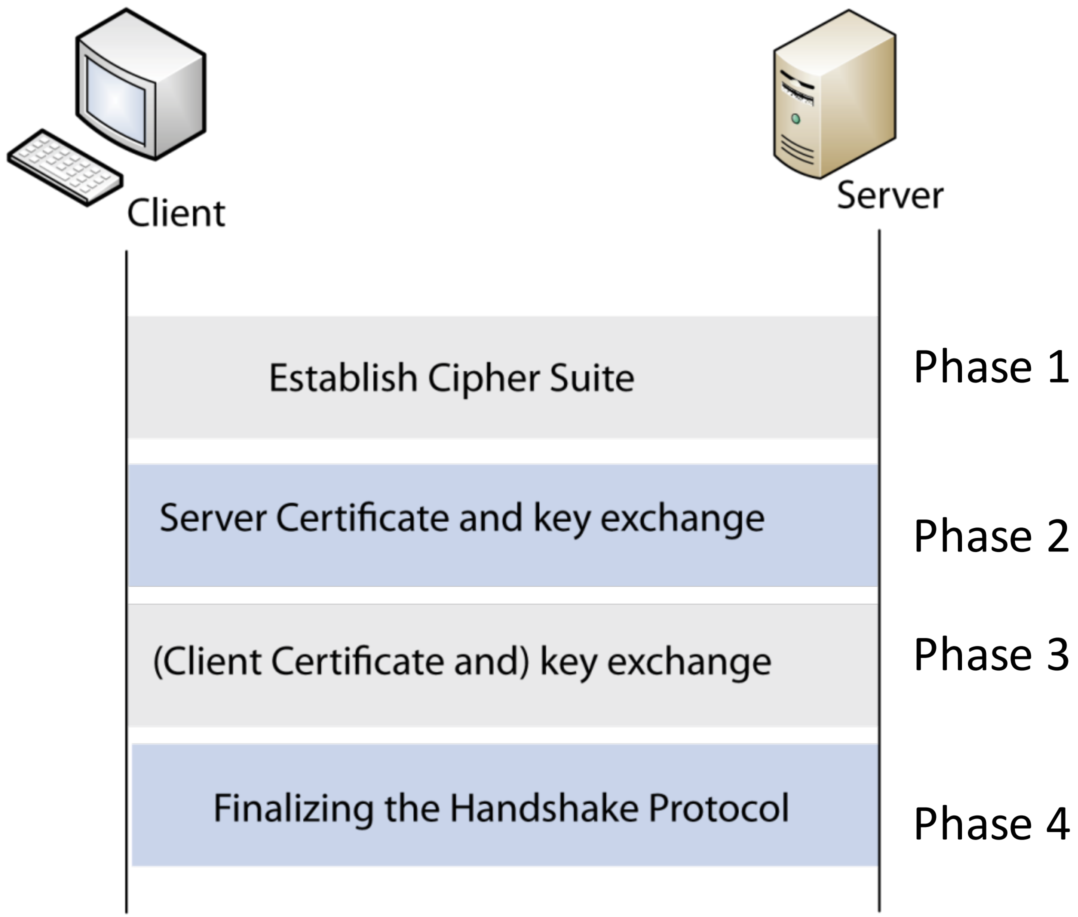
Session State

- Session Identifier – Chosen by the server to identify an active or resumable session state
- Peer certificate – An X509.v3 certificate of the peer. This state may be null
- Compression method – Algorithms used to compress data prior to encryption
- Cipher spec – Encryption algorithm and hash algorithm, corresponding attributes like hash_size
- Master secret – 48-byte secret shared between client and server
- Is resumable – Flag indicating if session can be used to initiate new connection

Connection State

- Server and client random – Byte sequences chosen by server and client for each connection
- Server write MAC secret – Secret key used for MAC operation by server
- Client write MAC secret – Secret key used for MAC operation by client
- Server write key – Encryption key for data encrypted by server, decrypted by client
- Client write key – Encryption key for data encrypted by client, decrypted by server
- Initialization vector – For CBC mode
- Sequence numbers – Sequence numbers for replay protection, reset on change cipher spec message

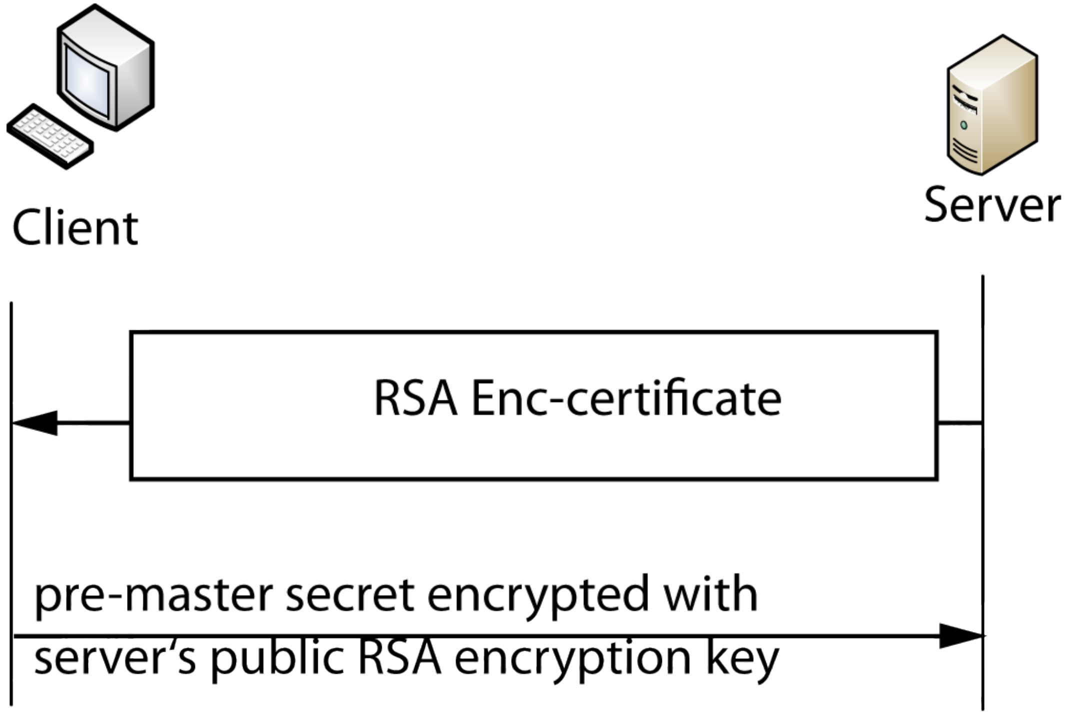
Handshake Protocol: Overview



Handshake: Key Exchange Methods

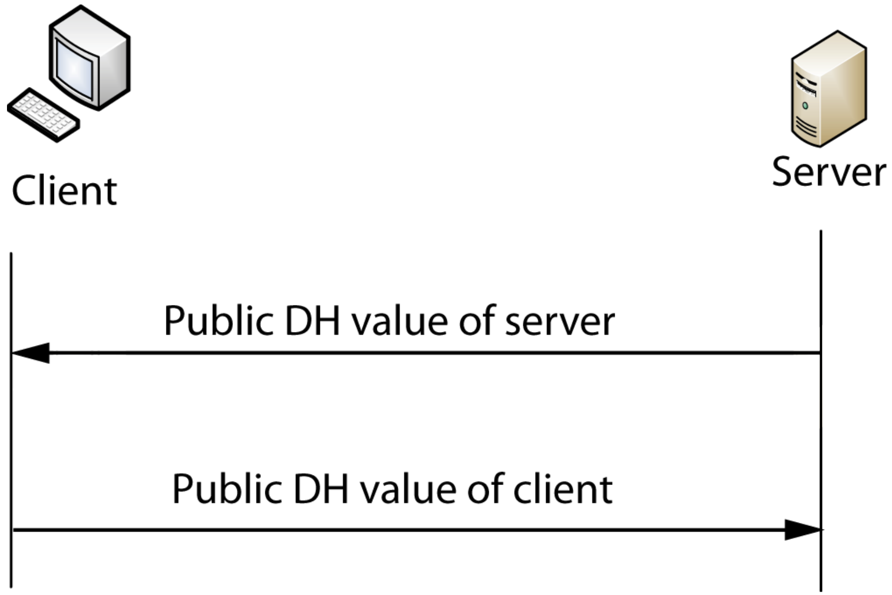
- After the handshake protocol, client and server share a secret key called master secret
- From the master secret other session keys for encryption and integrity protection are derived
- TLS supports four different key exchange methods to agree upon the master secret, these are called
  - “RSA”
  - “Anonymous DH”
  - “Ephemeral DH”
  - “Fixed DH”
- Depending on the selected method, the content of the messages in Phase 2 and 3 of the handshake protocol differs

Key Exchange Methods: RSA



- Client obtains a certificate for RSA encryption key from server
- Client generates a random key called pre-master secret
- Client encrypts it with server's public key and sends it to server
- The server decrypts the pre-master secret with its private key

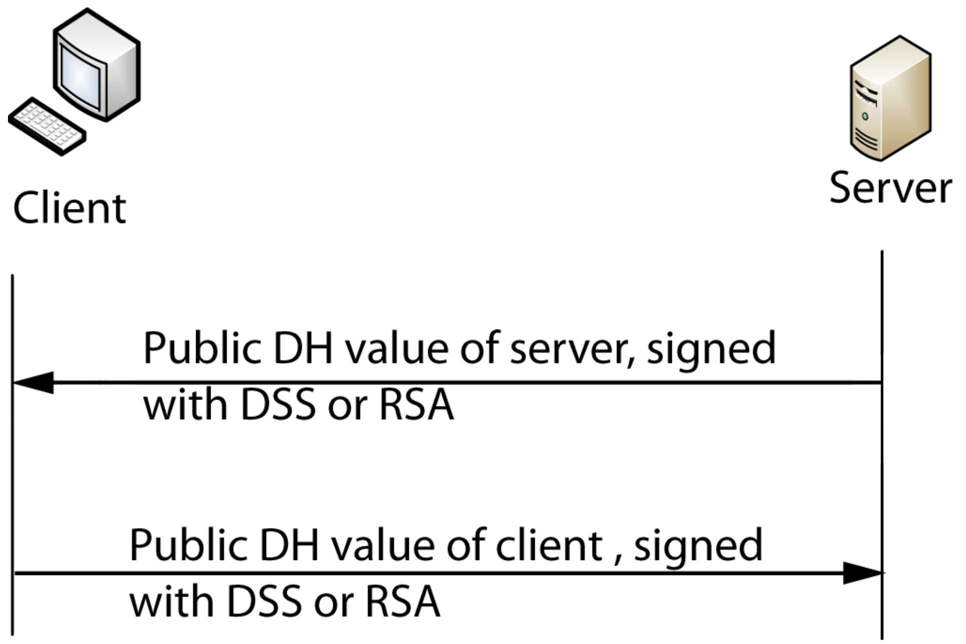
Key Exchange Methods: DH-Anonymous



- Client and server exchange un-signed public DH values
- Both compute the pre-master secret as the DH key from the public DH value of the other party and the own private DH value

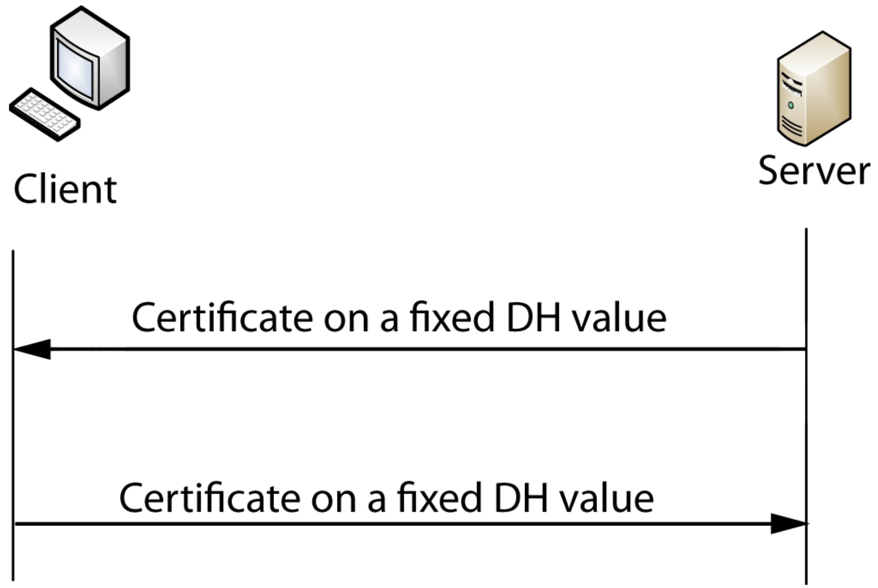
➢ Vulnerable to Man-in-the-Middle attacks but anonymous

Key Exchange Methods: Ephemeral DH (DHE)



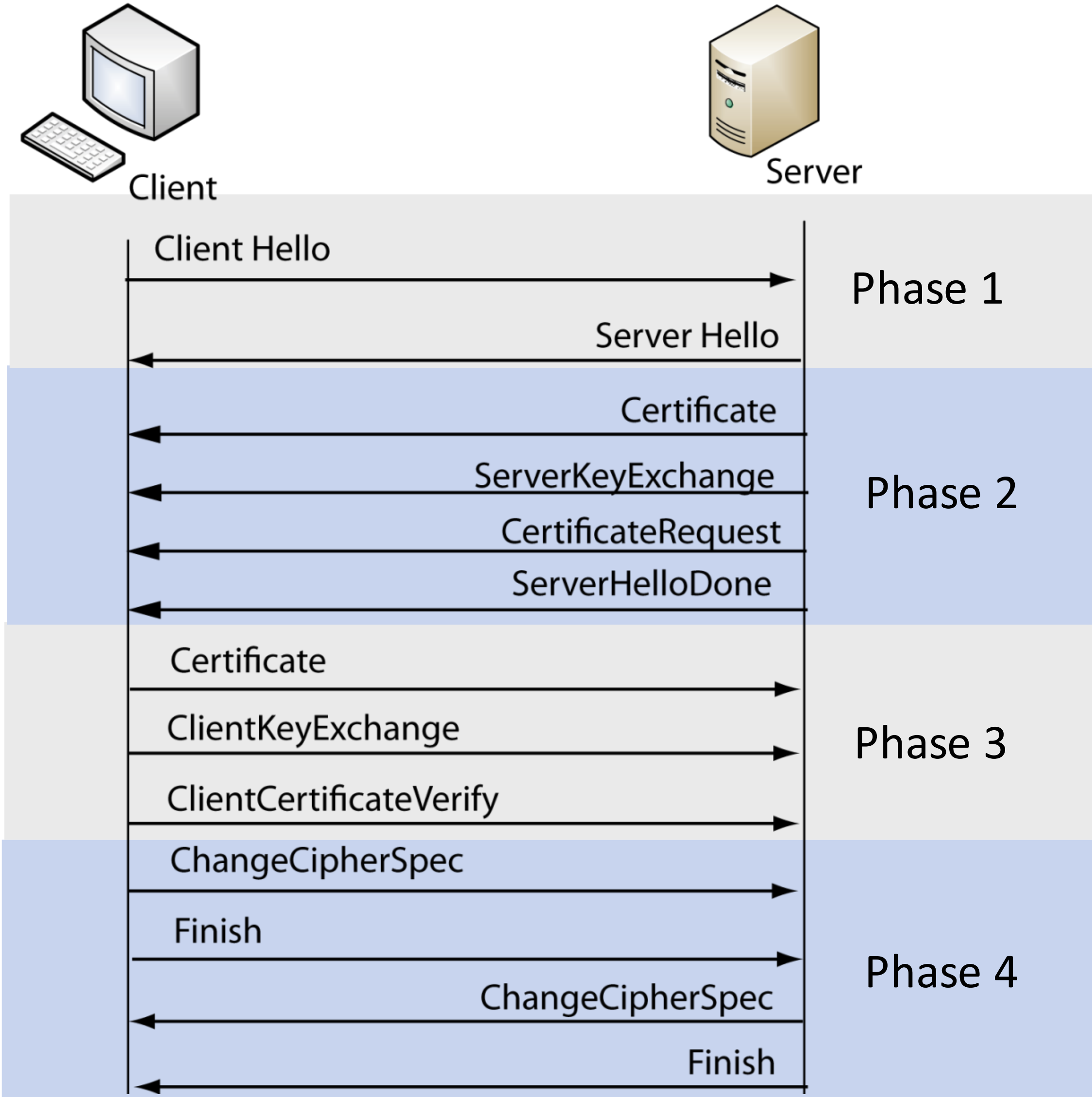
- Client / server generate ephemeral (= fresh) private DH values
- Client / server signs its ephemeral public DH value with its private signature key
  - Using either RSA or DSS signatures
- Client and server additionally exchange certificates on their signature keys (not shown above)

Key Exchange Methods: Fixed DH

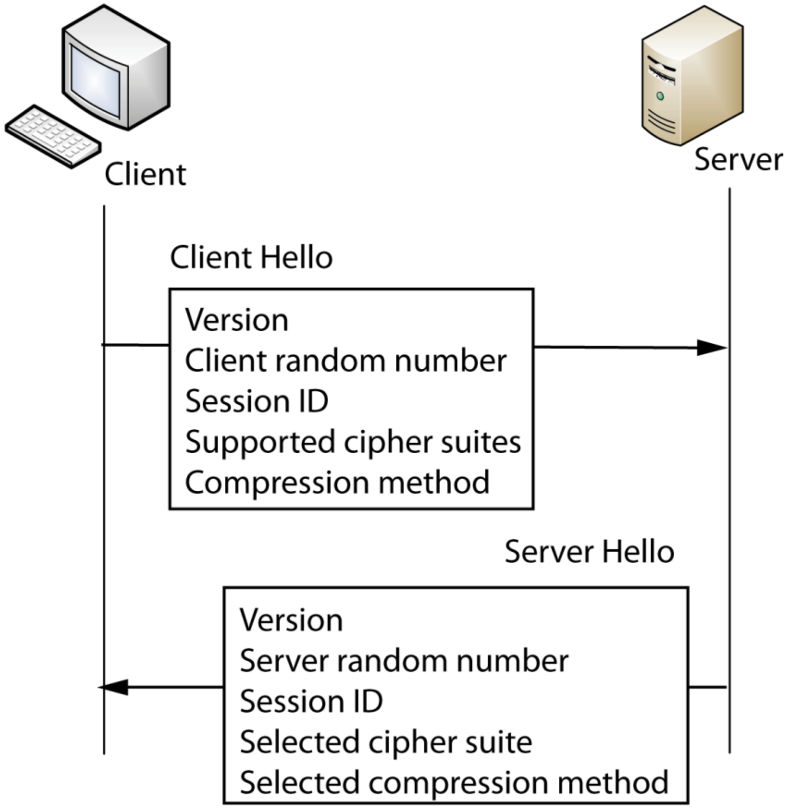


- Client and server use fixed public DH values that are signed by a certification authority
- Both compute the pre-master secret from the public DH value of the other party and their own private DH value

Handshake Overview



Handshake: Phase 1

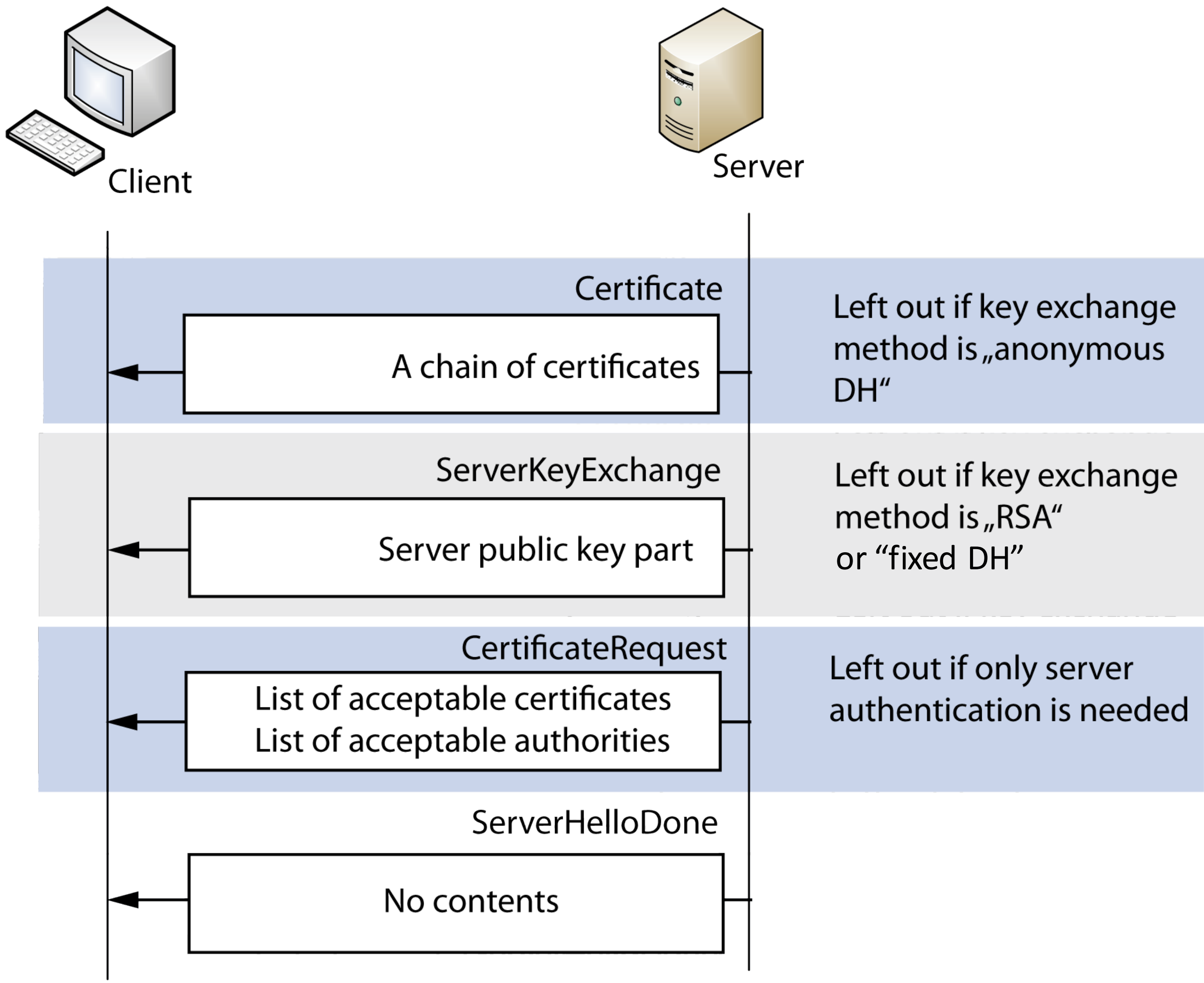


- Client and Server exchange "Hello" messages
- Client indicates supported cipher suites and compression methods
- Server selects cipher suite and compression method
- Client includes Session ID if resumable session exists
- Server assigns new session ID if new session is to be created
- Client and server exchange random numbers

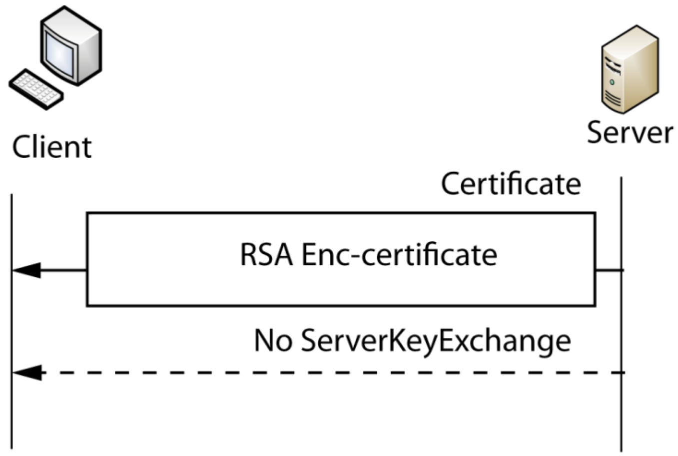
Cipher Suite Selection

- Client indicates supported cipher suite in the Client Hello
  - Includes
    - Supported key exchange (RSA, DHE, Fixed DH, DH-Anon)
    - Supported encryption algorithms
    - Supported MAC algorithms
  - Client orders the cipher suites of its choice according to its preferences
    - Most preferred selection first
- Server chooses a cipher suite and indicates its choice in the Server Hello
  - If no choice is acceptable for the server, it indicates this in an alert message

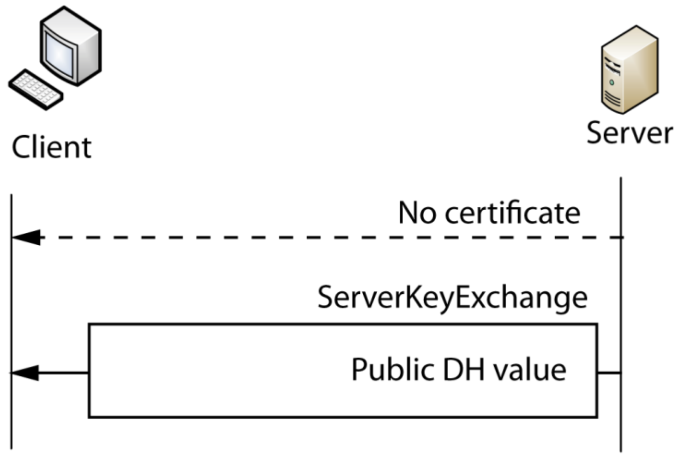
Handshake: Phase 2



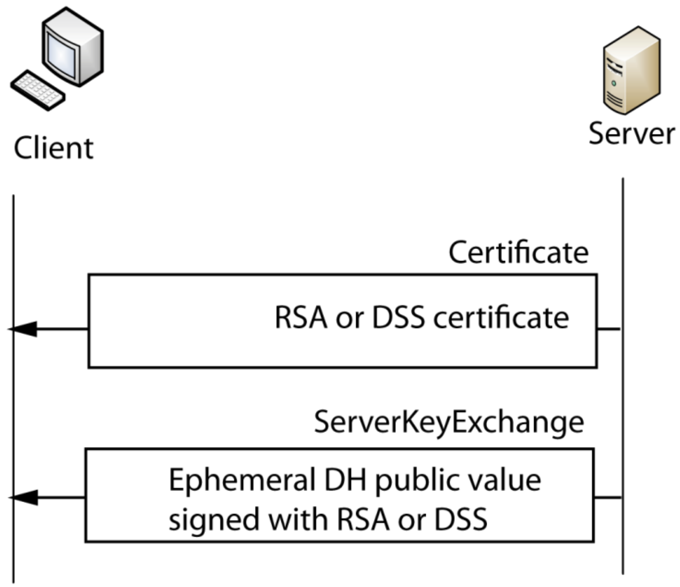
Different Key Exchange Types Phase 2



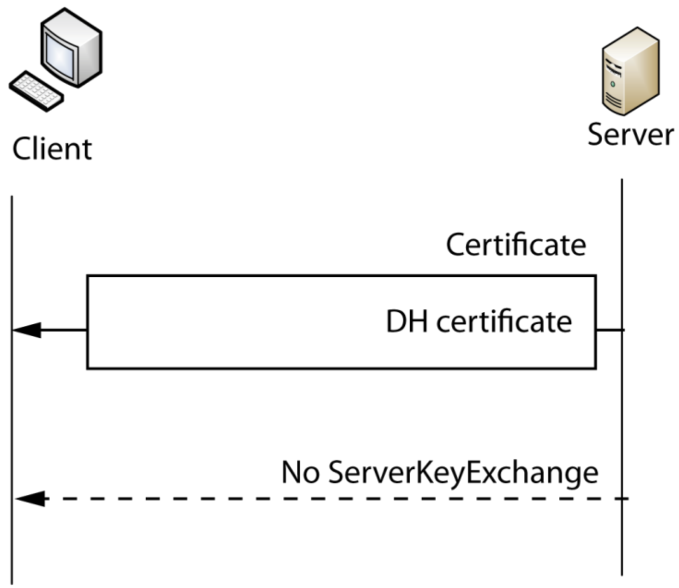
a) RSA



b) Anonymous DH

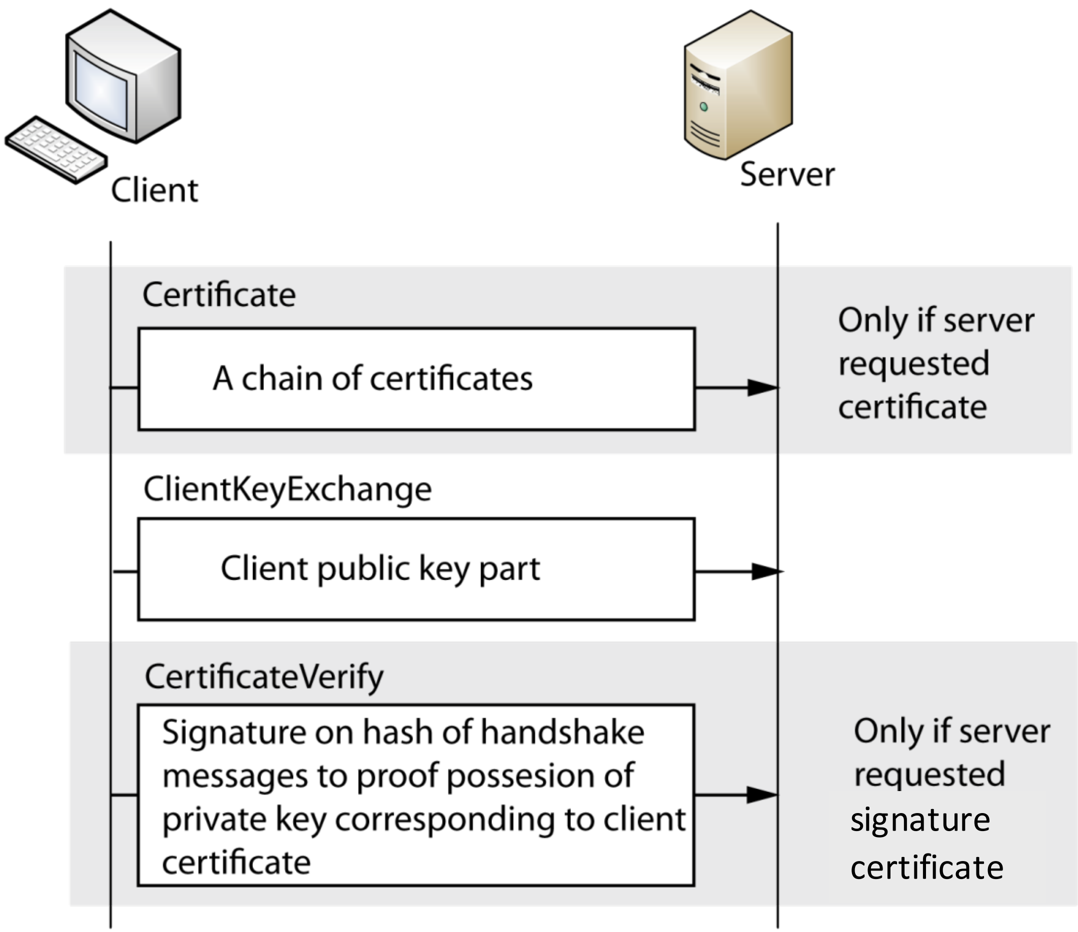


c) Ephemeral DH



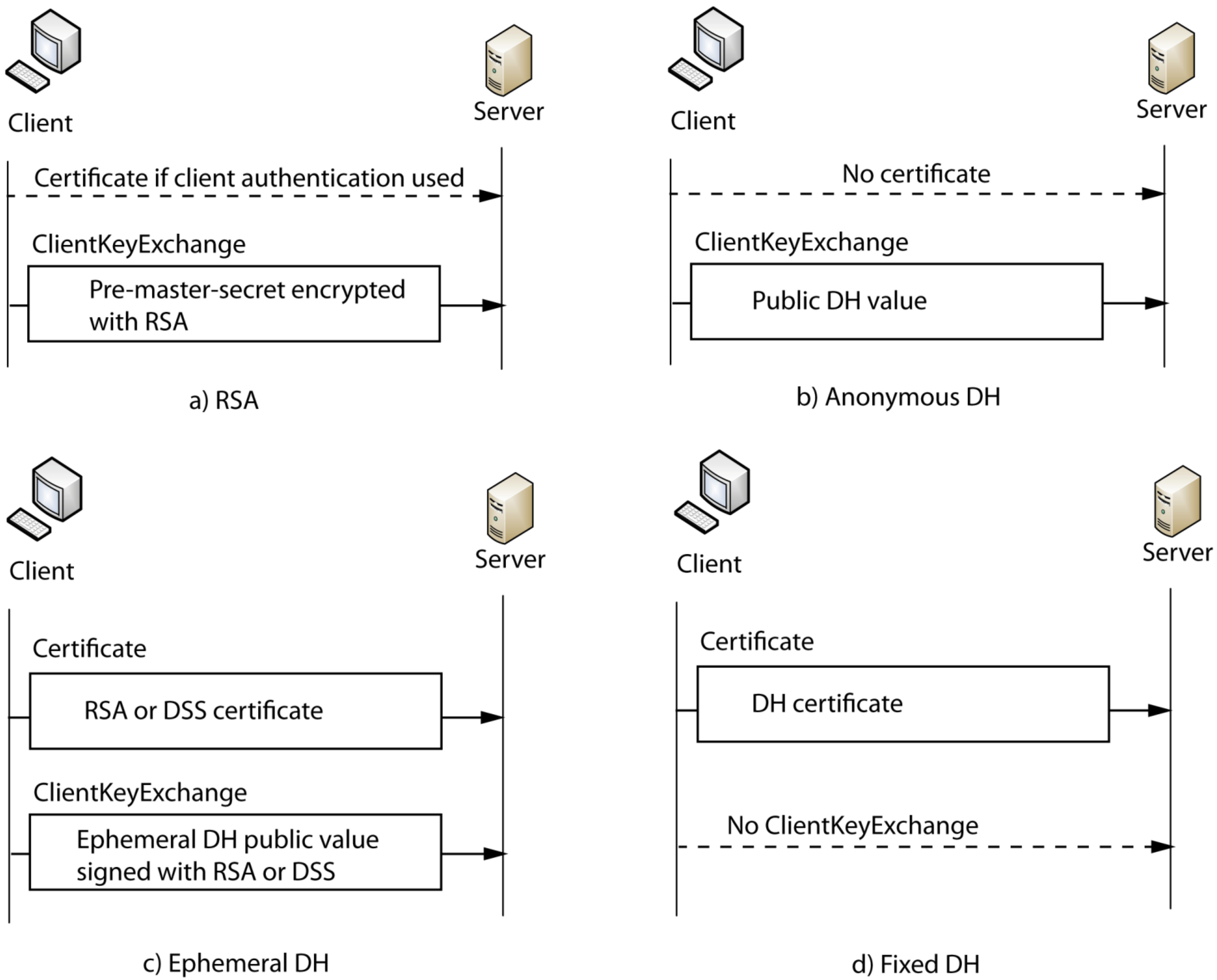
d) Fixed DH

Handshake: Phase 3



- If the server requested a certificate from the client the client sends its certificate
- The client sends the ClientKeyExchange message
  - Differs for the four key establishment methods
- The Client with its private key signs a hash of all handshake messages exchanged

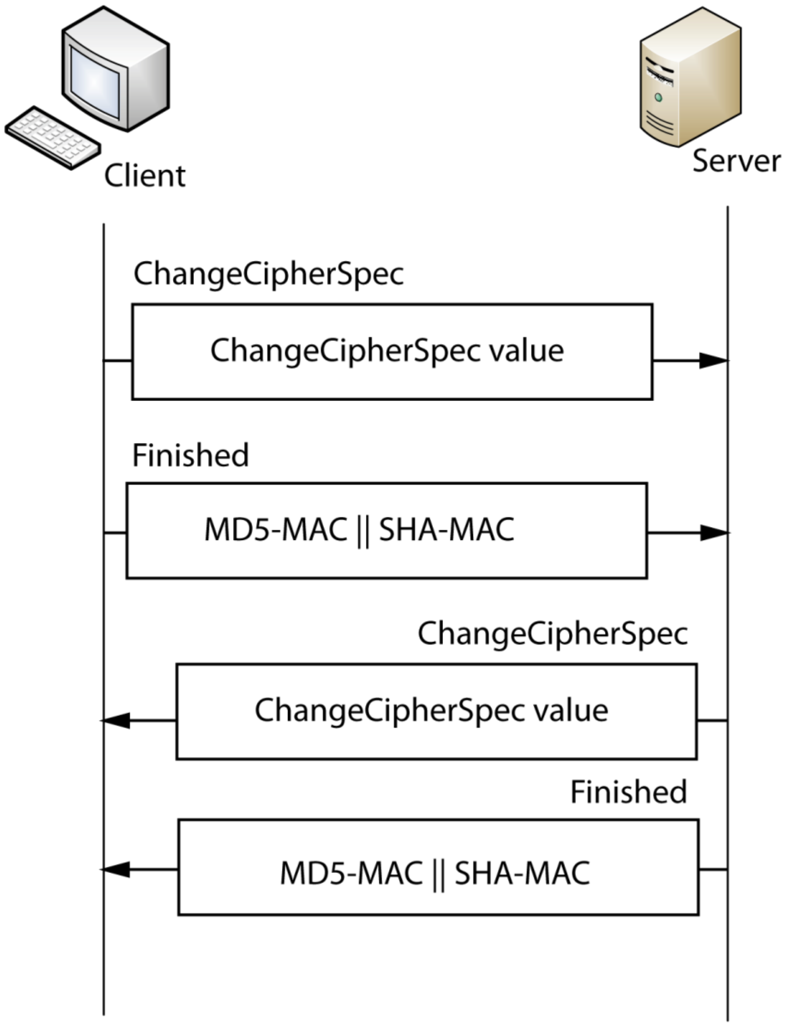
Different Key Exchange Types Phase 3



Mutual vs. Server-side-only Authentication

- Sever-side-only authentication can be reached by
  - Using the RSA key exchange
  - Or using DHE on server-side and anonymous DH on client-side
  - Or using fixed DH on server-side and anonymous DH on client side
- Mutual authentication is reached if
  - Client and server both use DHE
  - Or Client and server both use fixed DH
  - Or one uses DHE and the other uses fixed DH
- Which alternative is used is determined by the server
  - If the server requests a certificate from the client, mutual authentication is used

Handshake: Phase 4



- The ChangeCipherSpec message indicates that the client / server activates the agreed upon cipher suite and keys
- The Finished messages indicated that client and server have indeed established the same master secret
- PRF(master secret, „finished“, MD5(handshake_messages) || SHA-1(handshake_messages))
- In TLS 1.0 PRF is again based on an expansion of HMAC-MD5 and HMAC-SHA-1

From Pre-Master Secret to Master Secret

- The Master secret is computed as follows
  - Master Secret = PRF (pre-master secret, "master secret", ClientHello.random || ServerHello.random)
  - Where PRF is a pseudo random function
    - Since v. 1.2 this PRF can be negotiated as part of the cipher suite
    - Before v.1.2 the PRF was based on an XOR of and expansion of HMAC-MD5 and HMAC-SHA1
  - Length of parameters:
    - Pre-master secret: length varies with key exchange method
    - Master Secret: 48 byte
    - ClientHello.random, ServerHello.random: 32 byte

ChangeCipherSpec Protocol

- Specifies the ChangeCipherSpec message used as part of the handshake protocol
- Activates the cipher suite on both sides
  - i.e. changes status of cipher suite from pending to active
  - E.g. on server side after receiving ChangeCipherSpec message from client:

w	r
aa	
bb	
xx	
x	
i	

Pending

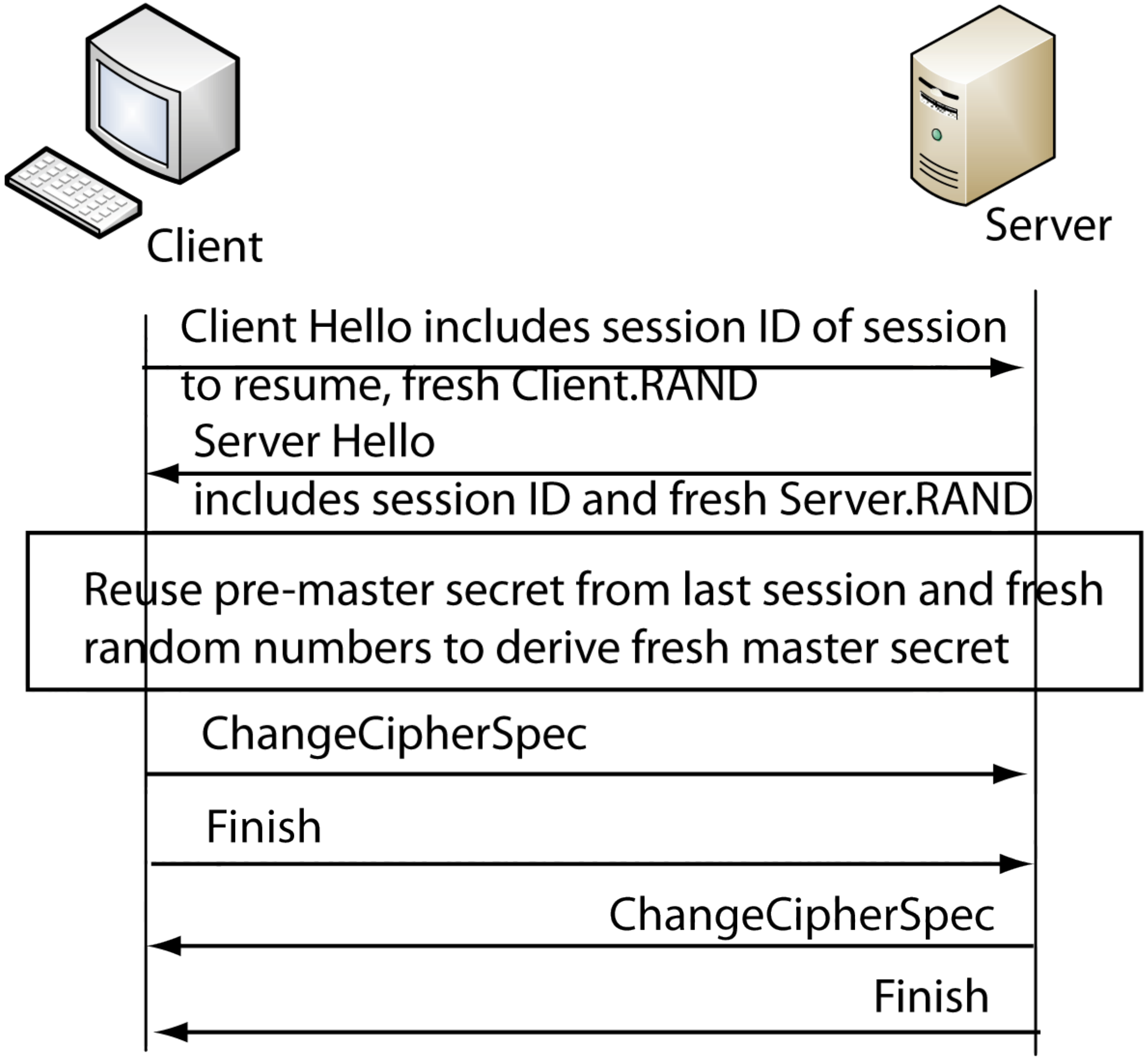
w	r
	aa
	bb
	yy
	y
	j

Active

Session Resumption

- SSL session setup has substantial overhead
- Randomness generation by client and server
- Transmission of certificates by server (and client)
- Derivation of master secret and derived keys by client and server
- Problems:
  - Significant performance penalty (mainly on server)
  - Server vulnerable to clogging (DOS) attacks
- Session resumption:
  - If client makes many connections to same server…
  - Server, client can re-use Pre-Master-secret from last connection
- How? By identifying a session using *session ID*

Handshake on Session Resumption



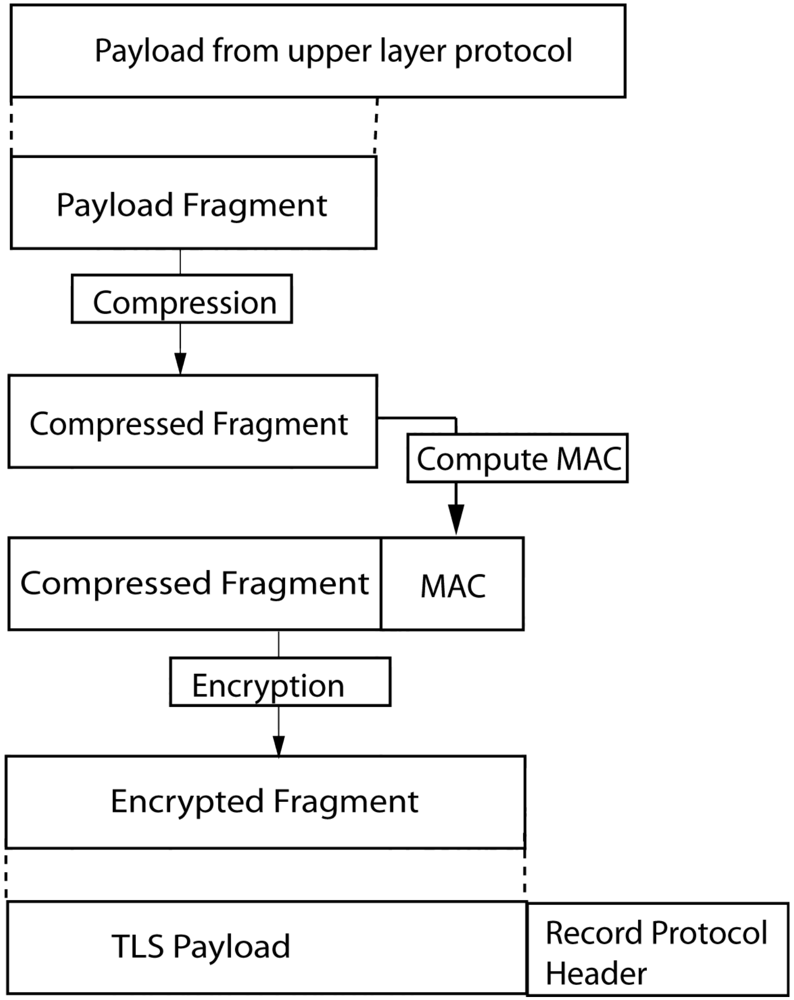
Alerts Defined for TLS in Alert Protocol

Value	Description	Meaning
0	CloseNotify	*Sender will not send any more messages*
10	UnexpectedMessage	*An inappropriate message was received*
20	BadRecordMAC	*An incorrect MAC received*
30	DecompressionFailure	*Unable to decompress appropriately*
40	HandshakeFailure	*Sender unable to finalize handshake*
41	NoCertificate	*Client has no certificate to send*
42	BadCertificate	*Received certificate corrupted*
43	UnsupportedCertificate	*Type of received certificate not supported*
44	CertificateRevoked	*Signer has revoked certificate*
45	CertificateExpired	*Certificate expired*
46	CertificateUnknown	*Certificate unknown*
47	IllegalParameter	*Out-of-range or inconsistent field*

Record Protocol

- Operates on a TLS connection state
- TLS connection state specifies
  - Compression, encryption, MAC algorithm
  - Parameters for these algorithms for both directions
    - MAC secrets
    - Encryption keys
    - IVs
- Parameters for pending states are set by the record layer
- Parameters needed to set the state parameters are set in the handshake protocol (session state)

Operation of the Record Layer



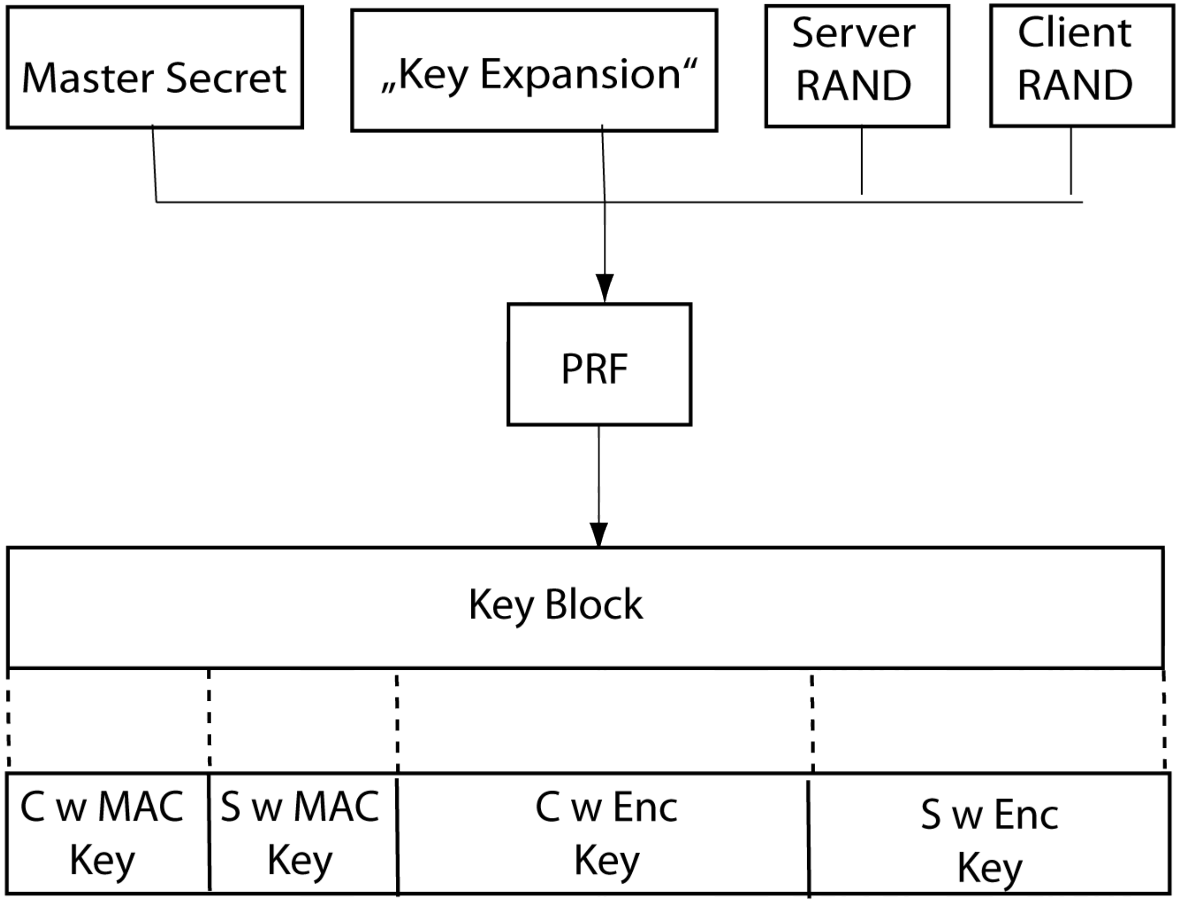
- The record layer protocol operates on payload of upper layer protocols
- The TLS Payload consists of compressed, integrity protected and encrypted fragments of the payload
- The record protocol header consists of a protocol field, the version number and a length field

Cipher Suites: Examples

Cipher Suite	Key Exchange	Encryption	MAC
TLS_NULL_WITH_NULL_NULL	NULL	NULL	NULL
TLS_RSA_WITH_NULL_MD5	RSA	NULL	MD5
TLS_RSA_EXPORT_WITH_RC4_40_MD5	RSA	RC4 (40bit)	MD5
TLS_RSA_WITH_RC4_128_SHA	RSA	RC4 (128bit)	SHA
TLS_RSA_WITH_IDEA_CBC_SHA	RSA	IDEA_CBC	SHA
TLS_DH_DSS_WITH_DES_CBC_SHA	DH_DSS	DES_CBC	SHA
TLS_DHE_RSA_WITH_DES_CBC_SHA	DHE_RSA	DES_CBC	SHA

- The different versions of TLS define different cipher suites
  - TLS v. 1.0: 28 different ones
  - TLS v. 1.1: 19 different ones (RC2 no longer supported)
  - TLS v. 1.2: 37 different ones (DES, IDEA phased out, AES (128 and 256) supported)
- All of the MAC computations are based on HMAC

Key Generation



- The key generation in TLS 1.0 was based on a PRF based on an XOR of an expansion to the size of key block of HMAC-MD5 HMAC-SHA-1
- The size of the key block depends on the cipher suite chosen

Changes in TLS 1.1

- Addition of new alert messages to prevent misuse of already existing alert messages
- IANA registries are defined for protocol parameters
- Premature closes no longer cause a session to be non-resumable
- Additional informational notes were added in the appendix to discuss how TLS protects against various attacks such as
  - Rollback attacks to lower versions
  - Bidding down attacks on the cipher suites

Changes in TLS 1.2

- PRF is made negotiable
  - Instead of using a fixed PRF (MD5-SHA-1 combination)
    - For generating the master secret from the pre-master secret and the encryption and integrity keys from the master secret
    - PRF is also used for the hash calculation in the Finished messages
- Substantial cleanup to the client's and server's ability to specify which hash and signature algorithms they will accept
- Some new cipher suites are defined
- TLS Extensions definition and AES Cipher Suites were merged from external documents into the base document

Changes in TLS 1.2 (cont'd)

- Alerts MUST now be sent in many cases
- After a certificate request, if no certificates are available, clients now MUST send an empty certificate list
- Added HMAC-SHA256 cipher suites
- Removed IDEA and DES cipher suites. They are now deprecated and will be documented in a separate document
- Added an implementation pitfalls section

Attacks against SSL/TLS

- Renegotiation Attacks
  - Attack possible due to missing cryptographic binding between renegotiation and original negotiation, Fixed in RFC 5746
- Bidding Down and Rollback Attacks
- BEAST Attack (2011): affects CBC-based ciphers in SSL 3.0 and TLS 1.0
  - Theoretical attack known since ‘95, due to the use of non random IVs, chosen plaintext attacks
- CRIME Attack (2012): theoretical attack known since 2004,
  - Exploits use of compression in TLS (ciphertext length leaks plaintext length leaks amount of compression, leaks tiny bit of plaintext)
- Lucky 13 Attack (2013): Theoretical attack known since 2002/03,
  - Affects CBC-based ciphers even in TLS 1.2
  - Exploits specifics of padding format in TLS
  - Makes use of information leaked from incorrect padding

Attacks against SSL/TLS continued

- RC4 Attack (2013): extends previous attack to RC4
  - https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/alFardan
- POODLE Attack (2014)
  - Padding Oracle Attack on deterministic padding used in the CBC-based cipher suites in SSLv3, permits plaintext recovery, in combination with downgrading attacks particularly serious
- Heartbleed Bug
  - Implementation bug in OpenSSL that misuses the heartbeat in TLS
  - Allows attacker to read up to 64 kByte RAM from server in one request

Secure Usage of SSL/TLS

- Designing Secure Applications using SSL API
- Validating Certificate (or certificates chain)
- Server Access Control (client authentication)
  - Using client certificates
  - Using username and password, etc.
- Client Access Control (server authentication)
- Site spoofing attacks on browsers

Designing Secure Applications

- Several SSL toolkits available
  - E.g. OpenSSL
  - All of them have slightly different APIs
- Initialization tasks:
  - Load CA's certificates (at clients; servers: only if using client auth)
  - Load keys and certificates
  - Seed random number generator
  - Load allowed cipher suites
  - Most toolkits allow adding new (more secure?) cipher suites
- Connection API calls
  - Very similar to standard TCP (Sockets) API
  - But returns server (and optionally client) certificate
  - Need to validate certificate

Validating Certificates

- Validation done by application, not by TLS/SSL!!
- Validation shall include
  - Verify that root CA is trusted
    - Check that CA is contained in predefined list of `trusted CAs` in application
      - E.g CAs included as 'trusted' in web browsers
  - Verify signature on certificate (signature on each certificate in the certificate chain)
  - Check validity/expiration dates
  - Check identities, constraints, key usage...
    - In particular: check if identifier in application server certificate matches the identifier of the desired application server!!!
  - Check for revocations
    - SSL does not carry CRLs
    - Application must collect CRLs by itself

Access Control

- Client access control (after server authentication):
  - Is this the server the client wanted to connect to?
  - Is this the *kind of server* the client had in mind?
  - Done by client application (e.g. browser) and (hopefully) client manually
- Server access control (after client authentication)
  - Is this an authorized client/customer?
  - What are his permissions?
  - Done by application server

Server Authentication/Client Access Control

- It is critical to authenticate (identify) the server
  - To protect secrets sent to server by the user (password, PINs, TANs, etc…)
  - To ensure validity of information from the server
- TLS/SSL authenticates server using server certificate
- Certificate contains identifier of server and public key of server
- SSL handshake confirms the server has matching private key
- Certificate is signed by a Certificate Authority (CA)
- Browser (or other application) knows how to validate CA's signature
- But users must
  - Specify whom they depend on to validate identifiers i.e. which CAs they trust and on what basis
  - Check the identifier (name) of the server they want to connect to
  - Check if server was indeed successfully authenticated

Specify Trusted CAs

- Remember the list of pre-installed trusted certificates in browsers!
- Very unclear how to determine trusted CAs
  - Certification Practice Statements of CAs (if available) indicate how CAs check identities before issuing certificates
  - E.g. Certification Practice Statement of Deutsche Telekom: http://pki.telesec.de/service/DT_ROOT_CA_2/cps.pdf
  - But: how much does that say?:
    - “Grundvoraussetzung für einen Neuauftrag ist ein bestehendes Vertragsverhältnis. Dieses Vertragsverhältnis wird durch T-Systems Vertriebseinheiten unter Zuhilfenahme juristischer Abteilungen generiert. Damit ist die ausreichende Authentifizierung des externen Kunden gewährleistet”
- Big hassle even for security-educated users!
- Reality: Most users never change the settings for the pre-installed CAs in their browsers

Check Server Identifier

- Server identifier is its Domain (DNS) Name
  - e.g. www.citibank.com
- Presented and input in the browser’s address bar
- But input is often indirect e.g. from search engine
- Users rarely notice changed address bar
- Sites often change address for different reasons
- Most users are not even aware of DNS structure
  - Cf. citibank.account.com to account.citibank.com
- Also: spoofing attacks present fake address bar (more later in chapter on phishing)
- Reality: users rarely detect incorrect identifier even if presented in address bar

Check if Server Authenticated

- Browsers and other applications typically indicate if server authentication was successful
  - E.g. by key/lock icons
  - by optional message box (rarely enabled)
  - by menu options (rarely used)
- Many users often don’t validate key/lock icons
- Small icon, requires awareness & inspection
- Some Web-spoofing attacks emulate key/lock icons, even menu
- Reality: users rarely notice if server is (not) authenticated

Client Authentication / Server Access Control

- Using client certificates…
  - High level of security (crypto done by SSL)
  - Requires issuing (buying?) certificates for each client
  - Browsers prompt user to select certificate (hassle)
  - If based on identity, requires database of clients in server
- Using Username-Password authentication
  - Browser sends password as argument of a form
  - Possibly filled by browser (`wallet` function)
  - Relies on SSL security (encryption+integrity+server authentication)

References and Further Reading

- Stallings Chapter 17
- Forouzan Chapter 17
- RFC 2246 TLS Version 1.0 (January 1999)
- RFC 4346 TLS Version 1.1 (April 2006)
- RFC 5246 TLS Version 1.2 (August 2008)

- Overview on recent pratical attacks against the different TLS versions
  - https://www.ietf.org/proceedings/89/slides/slides-89-irtfopen-1.pdf