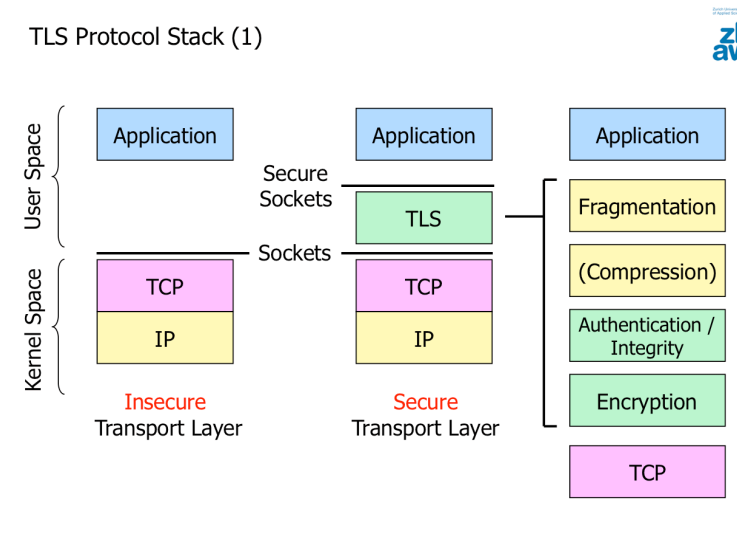
**IS ZA E2E Communication**

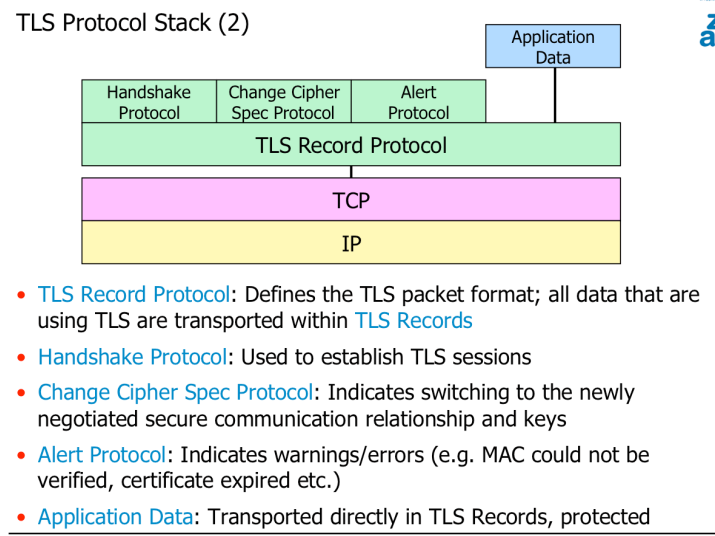
Secure communication protocols that protect the traffic flow between two end hosts or two applications on these end hosts

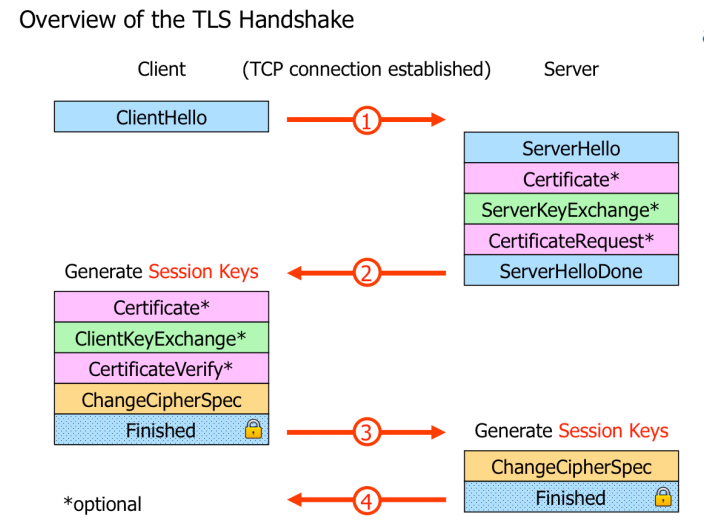
TLS => authenticated, integrity-protected, confidential

Typical setting:

* Server is authenticated as part of TLS
* Client is not authenticated as part of TLS (it’s possible)
* If necessary
  + Client is authenticated afterwards as part of the application using username/password

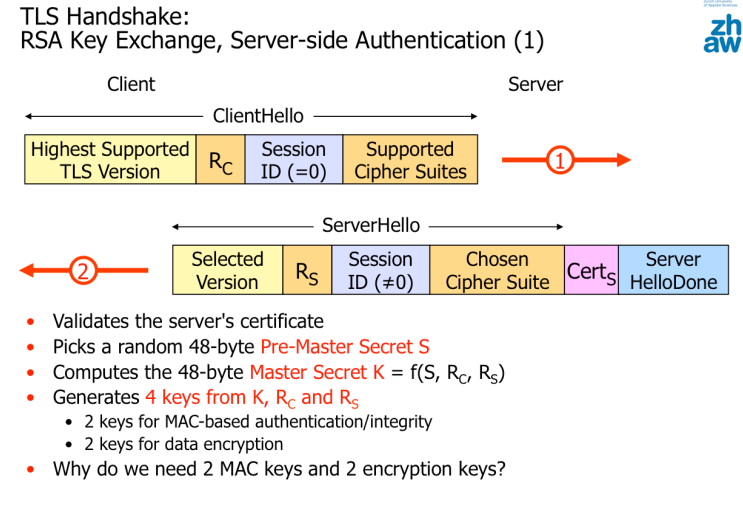






**TLS Handshake:**

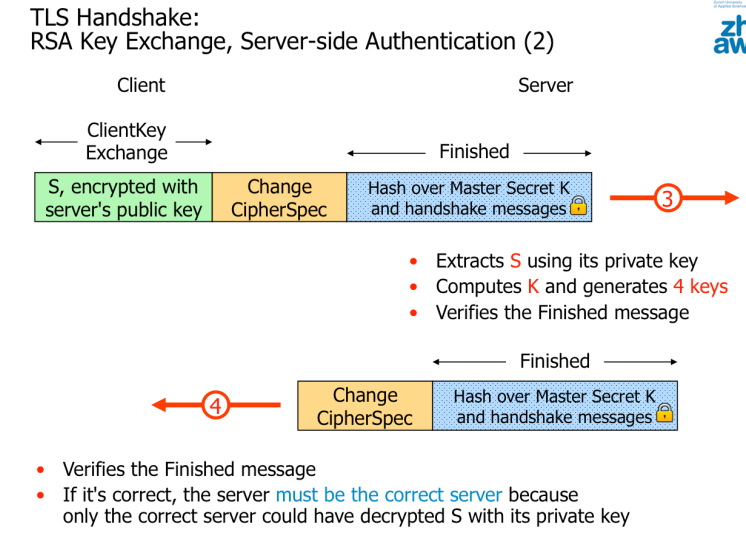
* Client and server agree on a protocol version
* “ “ “ select cryptographic algorithms
* Optionally authenticate each other
* Use public-key encryption techniques to generate shared secrets
* Client starts with **ClientHello**
* Server responds with **ServerHello**
* Attributes:
  + Protocol Version
  + Session ID
  + Cipher Suite
  + Compression Method
* Two random values generated and exchanged
  + ClientHello-Random Rc
  + ServerHello-Random Rs
* Optional:
  + Server sends certificate in Certificate message
  + If no certificate sent
    - Optional: ServerKeyExchange message may be sent instead
      * Containing the server part of a Diffie-Hellman DH secret
* If the server insists on a client side authentication
  + Optional: CertificateRequest message appended
* Server indicates the end of the server hello phayse by sending a **ServerHelloDone** message.
* Server => CertificateRequest message
  + Client must send either its certificate
  + Or a ‘no certificate’ alert
* Client has received a server certificate containing the server’s public RSA key
  + Client encrypts a randomly chosen premaster secret with it and send it to the server in a ClientKeyExchange message
  + Alternatively: the client can send its part of a DH key exchange
  + Each side can now form a shared master secret.
* Client emits a ChangeCipherSpec message announcing that the new parameters have been loaded
* Client sends Finished message => already encrypted with the new settings
  + Server does the same on its side

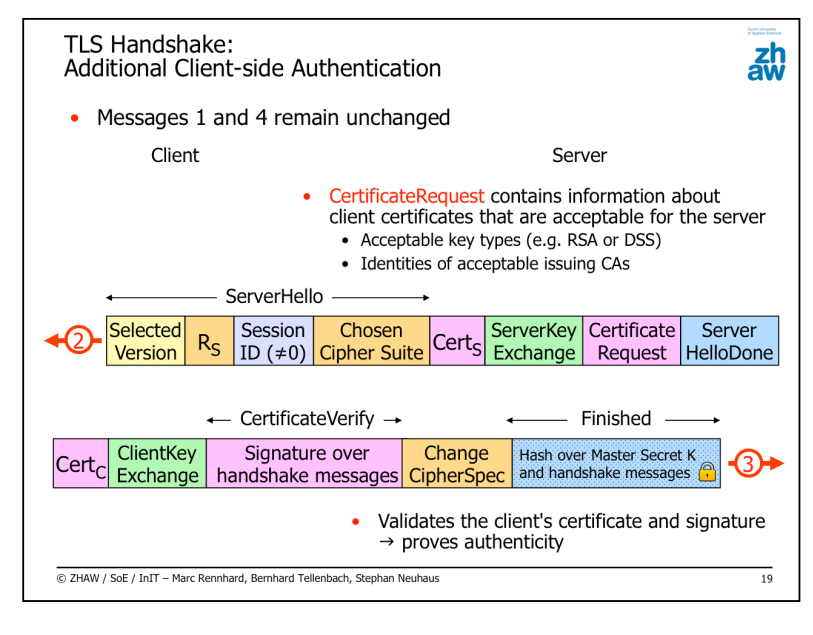


Perfect Forward Secrecy PFS

* If the attacker still canno decrypt the recorded communication, then we are talking about PFS
* RSA no PFS!
* DH => PFS!

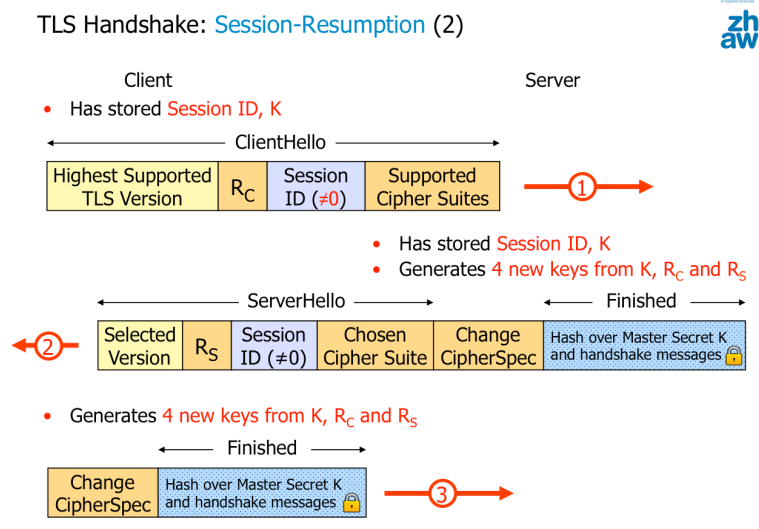
RSA is used more, because RSA means less computational workload for the server



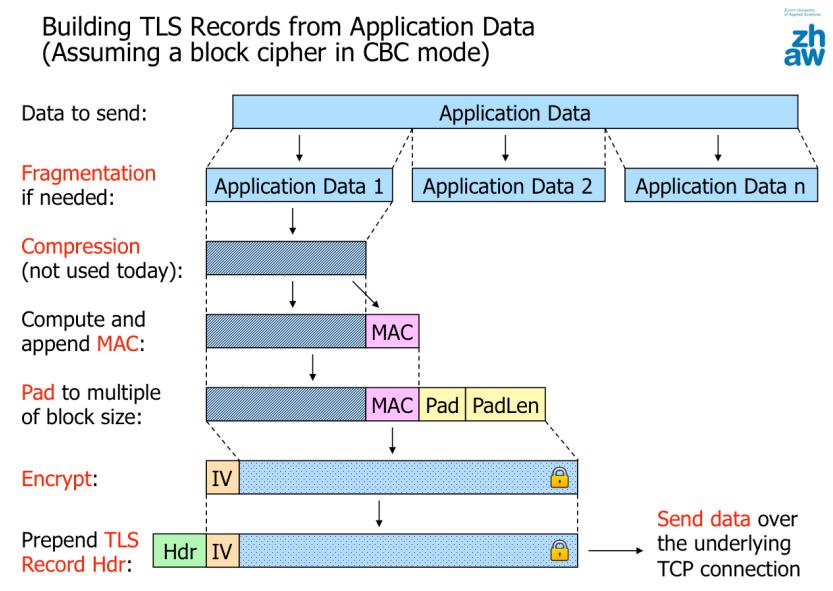
=>

TLS Session-Resumption

* TLS-Session is bound to the underlying TCP connection
* If the client opens a second TCP connection to the same server application, then a second TLS session must be established => significant workload => workload high and expensive due to the public key operations
* Session-Resumption = if TLS session exists => additional TLS sessions can be established **without** any public key computations



* Totally secure => if the server wouldn’t know the master secret K => it could not produce the Finished message
* Using fresh random values Rc and Rs guarantees the new keying material is also fresh

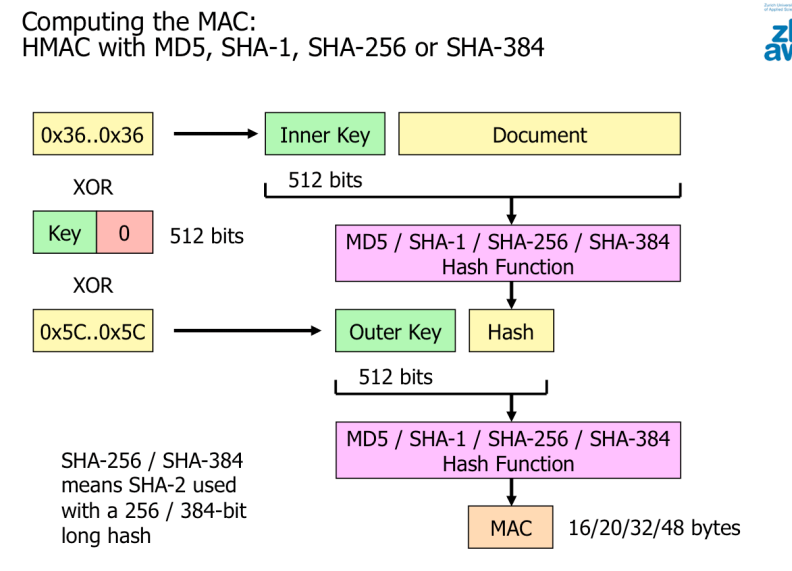


Padding:

* Added to force the length of the plaintext to be an integral multiple of the block cipher’s block length.
* Padding\_length field is one byte long and each byte in the padding data must be filled with the padding length value

Computing the MAC :

* HMAC (RFC 2104) is used to compute the MAC. The full output is used as the MAC.



Sequence Numbers

* MAC = HMAC(MAC\_key, sequence\_number || fragment)
* Are not transmitted, they are only used on the endpoints to compute and verify the MAC => possible because of TCP
* Used to make sure that all TLC records arrive at the destination and in the correct order:
  + Attack scenario: MITM => reorders/deletes/replays TLC records in a data stream
  + The recipient will notify this because verifying the MAC fails
  + This results in an alert that terminates the session

Cipher suites:

* Determines the set of cryptographic algorithms to be used for a TLS session
* TLS does not automatically mean secure!

TLS sessions are terminated when the underlying TCP connection is closed

* Potential problem: Truncation attack
* Solution: terminate a TLS session “in a clean way” into the TLS protocol
  + When an endpoint wants to terminate the TLS session, it first sends the other endpoint a specific alert message (close\_notify) to indicate that has completed sending the data; then it sends the TCP FIN or RST
  + The other endpoint should accept the data only if the close\_notify alert was received before TCP FIN or RST

Generating Key Material:

Computing the 48 Byte Master Secret

* PMS is either selected by the client (RSA key exchange) or computed during the DH key exchange
* Seed is used as input
  + Rc and Rs, which guarantees that even if the same PMS were used, a different Master Secret is generated.

RSA Pre-Master Secret

* 48 byte generated by client
  + Encrypted under the server’s public key and sent to the server
  + Server uses its private key to decrypt the PMS
  + Both parties then convert the pre master secret into the master secret

DH PMS:

* Negotiated key is used as PMS => converted into the master secret

IPsec

* Network layer secure end-to-end communications at the network layer
  + Secures the IP packet payload and parts of the IP header
  + Provides confidentiality, authentication and integrity
  + Protects all data above layer 3
  + IPSec must be installed in the kernel in the IPsec endpoints