Lecture Introduction into Cyber Security

Transport Layer Security (TLS) (Part 2)

Asya Mitseva, M.Sc.
Prof. Dr.-Ing. Andriy Panchenko

Chair of IT Security
Brandenburg University of Technology Cottbus-Senftenberg

17 January 2019

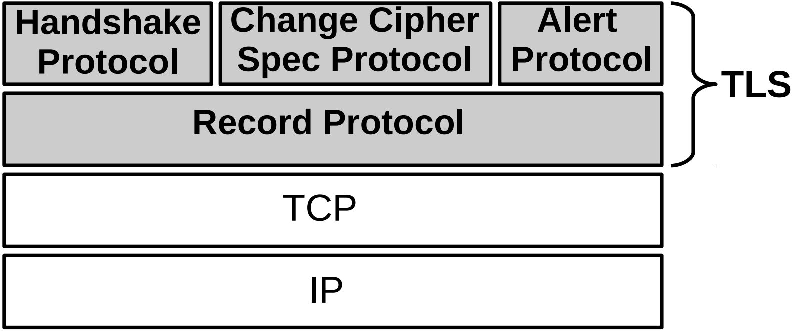
Brandenburgische
Technische Universität
Cottbus - Senftenberg

UNIVERSITÉ DU
LUXEMBOURG

Erasmus+

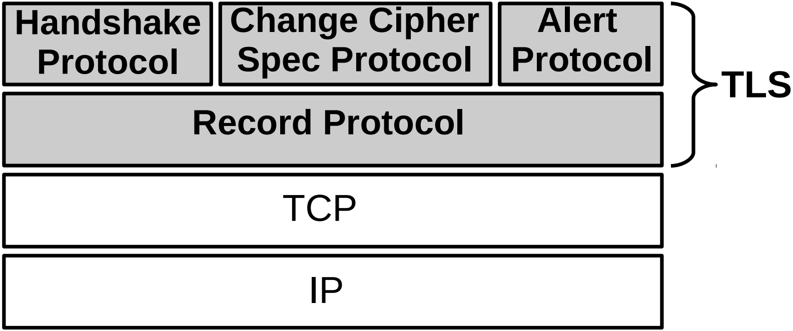
Recap: TLS Architecture (1/2)

- **Handshake Protocol**
  - Assure authentication of both communication parties
  - Negotiate encryption and MAC algorithms
  - Negotiate shared keys used to protect application data
- **Change Cipher Spec Protocol**
  - Activate the negotiated cipher suite



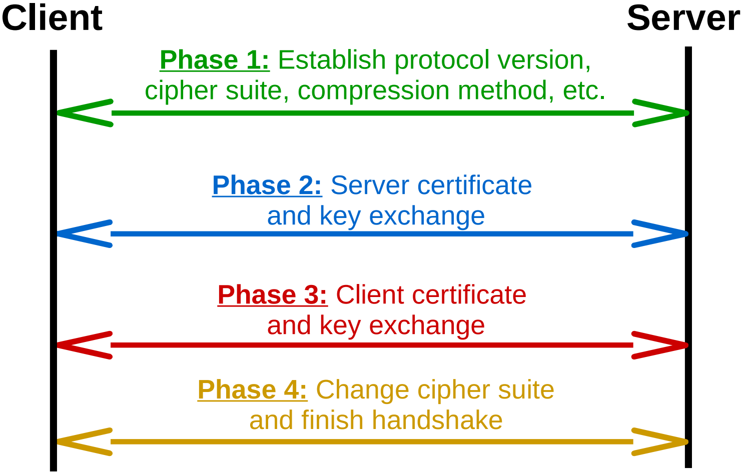
Recap: TLS Architecture (2/2)

- **Alert Protocol**
  - Used to exchange TLS-related alerts between communicating parties
- **Record Protocol**
  - Compute MAC on application data
  - Encrypt application data
  - Use keys based on master secret negotiated by the *Handshake Protocol*



Recap: Handshake Protocol

- Consist of *four phases*



Phase 1: Establish protocol version, cipher suite, compression method, etc.

Phase 2: Server certificate and key exchange

Phase 3: Client certificate and key exchange

Phase 4: Change cipher suite and finish handshake

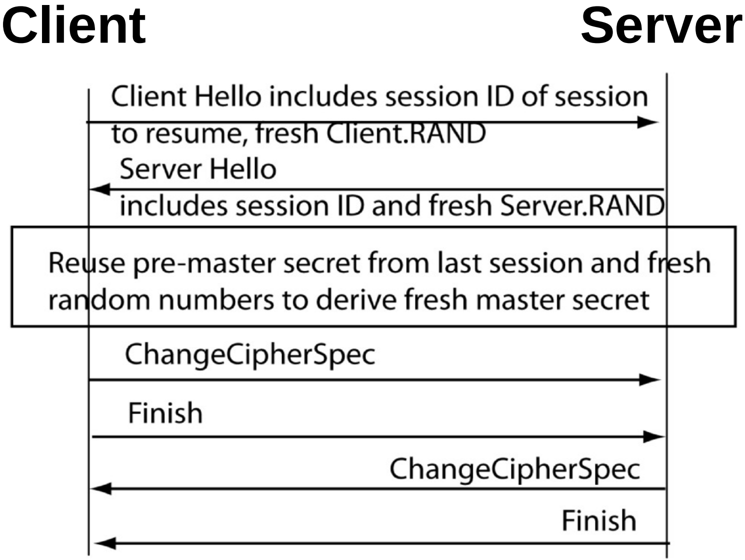
Mutual vs. Server-side-only Authentication

- **Sever-side-only authentication can be reached by**
  - RSA key exchange with server-only authentication
  - Ephemeral Diffie-Hellman on server side and anonymous Diffie-Hellman on client side
  - Fixed DH on server-side and anonymous DH on client side
- **Mutual authentication is reached when**
  - Client and server use Ephemeral Diffie-Hellman
  - Client and server use fixed Diffie-Hellman
  - Server uses Ephemeral Diffie-Hellman and client uses fixed Diffie-Hellman
- **Which alternative is used is determined by the server**
  - If the server requests certificate from the client, mutual authentication is used

Session Resumption: Overview

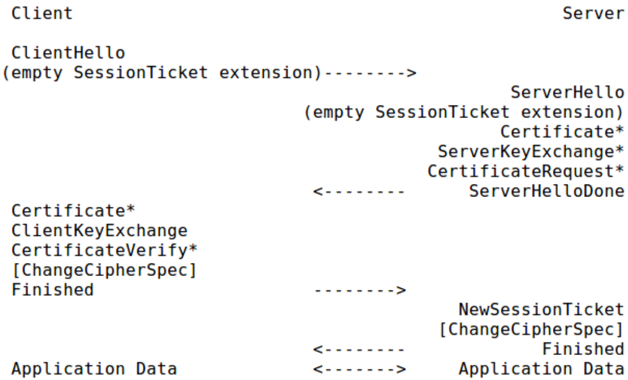
- **TLS session setup has substantial overhead**
- **Randomness generation by client and server is required**
- **Transmission of certificates by server (and client)**
- **Derivation of master secret and derived keys by client and server**
- **Problems**
  - Significant performance penalty (mainly on server)
  - Server vulnerable to clogging attacks
- **Servers can resume sessions**
  - If client makes many connections to same server
  - Server and client can re-use `pre_master_secret` from previous connections

Session Resumption: Handshake


Client
Server
Client Hello includes session ID of session to resume, fresh Client.RAND
Server Hello
includes session ID and fresh Server.RAND
Reuse pre-master secret from last session and fresh random numbers to derive fresh master secret
ChangeCipherSpec
Finish
ChangeCipherSpec
Finish

Session Resumption based on SessionTickets

- Servers do not need to keep track of session IDs
- Server sends encrypted session-state data to the client, i.e., *ticket*
- Client caches the ticket along with the master secret
- In case of session resumption, client presents ticket back to the server



```
      Client                                                Server

      ClientHello
     (empty SessionTicket extension)-------->
                                                       ServerHello
                                   (empty SessionTicket extension)
                                                      Certificate*
                                                ServerKeyExchange*
                                               CertificateRequest*
                                    <--------      ServerHelloDone
      Certificate*
      ClientKeyExchange
      CertificateVerify*
      [ChangeCipherSpec]
      Finished                      -------->
                                                  NewSessionTicket
                                                [ChangeCipherSpec]
                                    <--------             Finished
      Application Data              <------->     Application Data
```

TLS Heartbeat Protocol

- *Heartbeat:* Periodic signal generated by hardware or software to indicate normal operation
- **Heartbeat Protocol**
  - Run ot top of TLS Record Protocol
  - Consist of two message types: `heartbeat_request` and `heartbeat_response`
  - Use of the protocol negotiated in Phase 1 of the handshake
  - Heartbeat response contains exact copy of heartbeat request payload
  - *Purpose of the protocol*
    - Assure sender that the recipient is still alive
    - Generate activity across connection during idle periods
    - Avoid closure by firewalls which do not tollerate idle connections

Datagram Transport Layer Security (DTLS)

- Provide security for applications running upon *UDP*
- Include additional mechanisms to deal with
  - *Packet reordering*
    - Inclusion of explicit sequence number in DTLS record
    - If next record is not the expected one, it is queued for future handling
  - *Packet loss*
    - Make use of simple retransmission timer
    - Records are retransmitted when the timer expires

Changes in TLSv1.3

- **Remove support for number of options and functions**
  - Compression
  - Ciphers that do not offer authenticated encryption
  - Static RSA and Diffie-Hellman exchange
  - Change Cipher Spec Protocol
- **Use of Diffie-Hellman or Elliptic Curve Diffie-Hellman**
  - Does not permit RSA any more
- **Reduce number of packets sent during handshake**
  - Client sends its crypto parameters for key exchange before cipher suite has been negotiated
  - Server calculates the master secret before sending its first response

TLS/SSL: Conclusion

- **Protocol suite providing**
  - Integrity and encryption of application data
  - Authentication of identities of both communicating parties
- **TLS is standardized version of Secure Socket Layer (SSL)**
- **Executes four-phase handshake to negotiate keys**
  - RSA algorithm
  - Fixed Diffie-Hellman algorithm
  - Ephemeral Diffie-Hellman algorithm
  - Anonymous Diffie-Hellman algorithm
- **Support session resumptions**
- **But: Validation done by application, not by TLS/SSL**