Themen zur Computersicherheit

Transport Layer Security (TLS)

PD Dr. Reinhard Bündgen
buendgen@de.ibm.com

TLS/SSL

- Transport Layer Security (TLS)
  - sicheres Transport Protokoll
  - Nachfolger von Secure Socket Layer (SSL)
  - neuste Version: 1.3
    - RFC 8446
  - am meisten verbreitet: Version1.2
    - RFC 5264
  - Protokolle, die TLS/SSL benutzen:
    - HTTPS, POP3, SMTP, NNTP, SIP, IMAP, IRC, LDAP, FTP, TN3270, OpenVPN
  - Open Source Implementierungen:
    - OpenSSL, GnuTLS

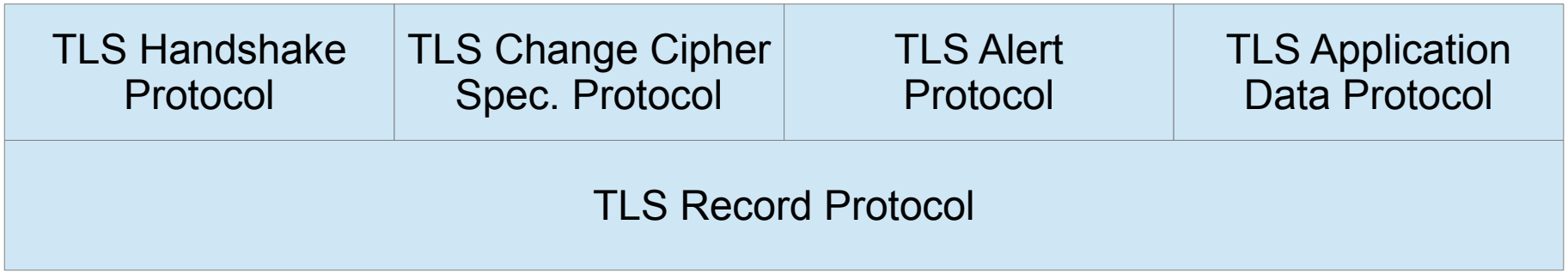
Zweck von TLS/SSL

- Aushandeln von Verschlüsselungs- und Authentisierungsmethoden und Schlüsseln (handshake)
- basierend auf dem Verhandlungsergebnis Übertragen von verschlüsselten und authentisierten Daten.

TLS 1.2 Protokoll

Protokollaufbau

- TLS Protokoll läuft über TCP
- DTLS Protokoll läuft über UDP



Sicherheitsparameter einer TLS Verbindung

Parameter

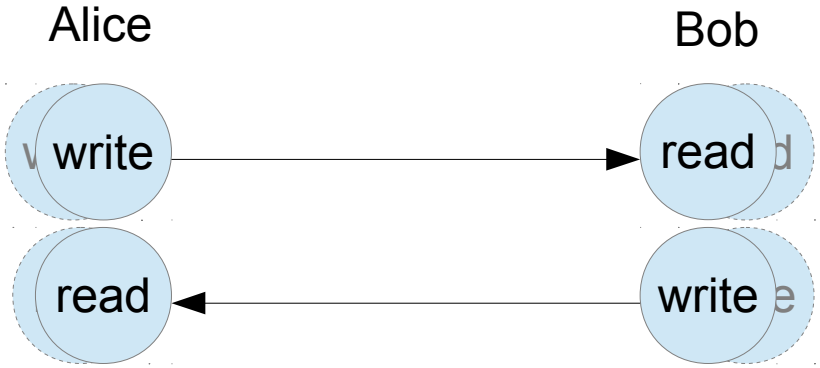
- Verbindungsendpunkt: Client oder Server
- PRF Algorithmus
- Massendatenverschlüsselungs-algorithmus
- MAC Algorithmus
- Komprimierungsalgorithmus
- master secret (48 Bytes)
- client random (32 Bytes)
- server random (32 Bytes)

aus den Sicherheitspara-metern werden folgende Werte berechnet:

- client write MAC key
- server write MAC key
- client write encryption key
- server write encryption key
- client write IV
- server write IV

Zustände für das TLS Record Protokoll

- Verbindungszustand
  - Sicherheitsparameter
  - Komprimierungszustand
  - Verschlüsselungszustand
  - MAC-key
  - Sequenz-Nummer
- 4 Zustände
  - pending read
  - pending write
  - current read
  - current write



- initiale „current“ Zustände
  - keine Komprimierung
  - keine Verschlüsselung
  - keine MACs
- initiale „pending“ Zustände:
  - ungültig
- das handshake Protokoll kann „pending“ Zustände setzen
- Change Cipher Spec Protokoll
  - ersetzt current Zustand durch pending Zustand falls Pending Zustand gültig
  - ersetzt pending Zustand durch ungültigen Zustand

Aufgaben des TLS Record Protokolls

1) Fragmentieren
   - Nachrichtenstücke ≤ 2^{14} Bytes Klartext
2) Komprimieren der Fragmente
   - Identitätsfunktion zulässig
3) für Strom- und Blockchiffren
   a) Berechnen einer MAC für komprimiertes Fragment
   b) Verschlüsseln des komprimierten Fragments und der MAC (+ Padding)
4) für AEAD Chiffren
   - berechnen von AEAD Resultat (Geheimtext + Tag) für komprimiertes Fragment

Das TLS-Handshake-Protokoll

- verhandelt Sitzungseigenschaften
  - Sitzungsidentifikator
  - Partner Zertifikat
  - Komprimierungsmethode
  - Chiffrenspezifikation
    - PRF
    - Massendatenverschlüsselungsalgorithmus
    - MAC-Algorithmus
  - master secret
  - „is resumable“ Flag

Handshake-Ablauf

Client		Server
ClientHello	-------->	
		ServerHello
		Certificate*
		ServerKeyExchange*
		CertificateRequest*
	<-------	ServerHelloDone
Certificate*		
ClientKeyExchange		
CertificateVerify*		
[ChangeChipherSpec]		
Finished	------->	
		[ChangeCipherSpec]
	<-------	Finished
Application Data	<------->	Application Data

*) optional

Inhalt der Hello Nachrichten

Client Hello

- Client Version
- Client Random
  - 4 Byte Zeit, 28 Byte random
- Sitzungsidentifikator
  - falls nicht leer, möchte Client Sitzung mit gegebener ID fortsetzen
- Liste unterstützter Cipher Suites
- Liste unterstützter Komprimierungsmethoden
- optional: Erweiterungen
  - signature extension: unterstützte Signatur/Hash Verfahren

Server Hello

- Server Version
- Server Random
  - 4 Byte Zeit, 28 Byte random
- Sitzungsidentifikator
  - Server entscheidet, ob er bestehende Sitzung fortsetzen „will“
- ausgewählte Cipher Suite
  - aus der Liste des Clients
- Komprimierungsmethode
  - aus der Liste des Clients
- optional: Erweiterungen

Weitere Komponenten des Handshakes

- ServerKeyExchangeMessage (optional)
  - enthält zusätzliches Schlüsselmaterial, z.B. DH Parameter
- ClientKeyExchange
  - premaster secret
  - RSA: verschlüsselt mit öffentlichem Schlüssel des Servers
    - Client Version + 46 Byte Random
    - PKCS #1 (v 1.5) padding (encoding)
  - DH: Client-Teil als öffentlicher Client DH Schlüssel
- Berechnung des Master Secrets
  - PRF(premaster secret, „master secret“, ClientHello.random + ServerHello.random);

Cipher Suites

Zwei Typen

- TLS_<key exchange>_WITH_ <symmetric cipher>_<HASH>
  - signature = key exchange
- TLS_<key exchange>_<signature>_WITH_<symmetric cipher>_<HASH>
- HASH → HASH-HMAC

Beispiele

- TLS_NULL_WITH_NULL_NULL
- TLS_RSA_WITH_NULL_SHA256
- TLS_RSA_WITH_RC4_128_SHA
- TLS_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_DH_anon_WITH_RC4_128_MD5
- TLS_DH_DSS_WITH_AES_128_CBC_SHA
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA
- TLS_DHE_DSS_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSS_WITH_AES_128_GCM_SHA256

Was unterstützt Ihr Browser? https://cc.dcsec.uni-hannover.de

TLS 1.3

seit August 2018

Änderungen zu TLS 1.2 (Auswahl)

- nur noch AEAD Chiffren
  - GCM, CCM, chacha20-Poly1305
- Schüsselaustausch: DHE, ECDHE oder PSK
- kein Change Cipher Spec Protokol
- HKDF-extract statt PRF
- keine MD5, SHA1, SHA224 mit Signaturen
- RTT-0 Option

Basic full TLS 1.3 handshake

Client		Server	
Client Hello			key exchange
+ key share * + signature algorithms* + key share exchange modes* + pre shared key*	--->		
		Server hello	
		+ key share* + pre key share*	
		{Encrypted Extensions} {CertificateRequest*}	server parameters
		{Certificate*} {CertificateVerify*}	authentication
		{Finished}	
	<---	[Application Data*]	
{Certificate*} {CertificateVerify*}			
{Finished}	--->		
[Application Data]	<-->	[Application Data|	

+ : Erweiterung, *: Optional, {...}: geschützt mit handshake_traffic_secret,
[...] geschützt mit application_traffic_secret_N,

TLS 1.3

- key share = (EC)DH public key
  - fehlerhafter key share: HelloRetryRequest von Server
- PSK = pre-shared key (identifier)
  - genutzt bei Session-Fortführung

TLS 1.3 RTT-0 handshake

Client		**Server**
Client Hello		
+ early data + key share * + key share exchange modes + pre shared key (Application Data*)	--->	
		Server hello
		+ pre shared key + key share*
		{Encrypted Extensions}
		+ early data*
		{Finished}
	<---	[Application Data*]
(EndOfEarlyData)		
{Finished}	--->	
[Appliction Data]	<-->	[Application Data]

- keine forward secrecy
- nicht sicher gegen re-play Attacken

+ : Erweiterung, *: Optional, [...], {...}: geschützte Daten
(...) geschützt mit client_early_traffic_secret

TLS 1.3 Schlüsselableitungen

Funktionen

- Derive-Secret (DS)
  - *Geheimnis*
  - Konstante
  - Nachricht
- HMAC based extract key derivation function . RFC5869 (HKDF)
  - *Salt*
  - *Geheimnis*

0, PSK -> HKDF: Early Secret

- DS: binder_key
- DS: client_early_traffic_secret
- DS: early_exporter_master_secret

DS , (EC)DH: HKDF = Handshake Secret

- DS: client_handshake_traffic_secret
- DS: server_handshake_traffic_secret

DS, 0 -> HKDF = Master Secret

- DS: client_application_traffic_secret_0
- DS: server_application_traffic_secret_0
- DS: exporter_master_secret
- DS: resumption_master_secret