

## Netzicherheit – Architekturen und Protokolle TLS/SSL und Policy



1. Einführung
2. TLS-Schlüsselaustausch
3. TLS-Datenaustausch
4. Zusammenfassung
5. Policy



## Netzicherheit – Architekturen und Protokolle TLS/SSL und Policy



1. Einführung
2. TLS-Schlüsselaustausch
3. TLS-Datenaustausch
4. Zusammenfassung
5. Policy



- **Sicherung von Transportverbindungen**
  - TCP, UDP, ... Anwendungen benötigen Sicherheit
    - ▶ Online-Banking, Online-Shopping, ...
    - ▶ Authentifizierung durch Passwörter
  - am häufigsten benutztes Sicherheitsprotokoll für
    - ▶ HTTP (→ HTTPS), FTP (→ FTPS), RMI, SMTP, LDAP, ...
- **SSL/TLS ermöglicht**
  - Ende-zu-Ende-Sicherheit
    - ▶ kann Email-Verkehr mit SSL komplett gesichert werden?
  - Gewährleistung der *Integrität*, *Vertraulichkeit* von Daten
  - *Authentifizierung* durch Zertifikate
  - Schutz vor Wiederholungsangriffen

2

1994

- Netscape: HTTP muss gesichert werden
- Entwicklung des *Secure Socket Layer* Protokolls (SSL)
- V1 Closed Source ohne externes Review, Schwachstellen
- V2: Diskussion mit externen Spezialisten, erstes Produkt

• 1995

- erstes Treffen der IETF
- V3: vollständige Neuentwicklung des Protokolls

• 1999

- TLS v1.0: Entwicklung von *Transport Layer Security* (TLS) in der IETF auf Basis von SSLv3

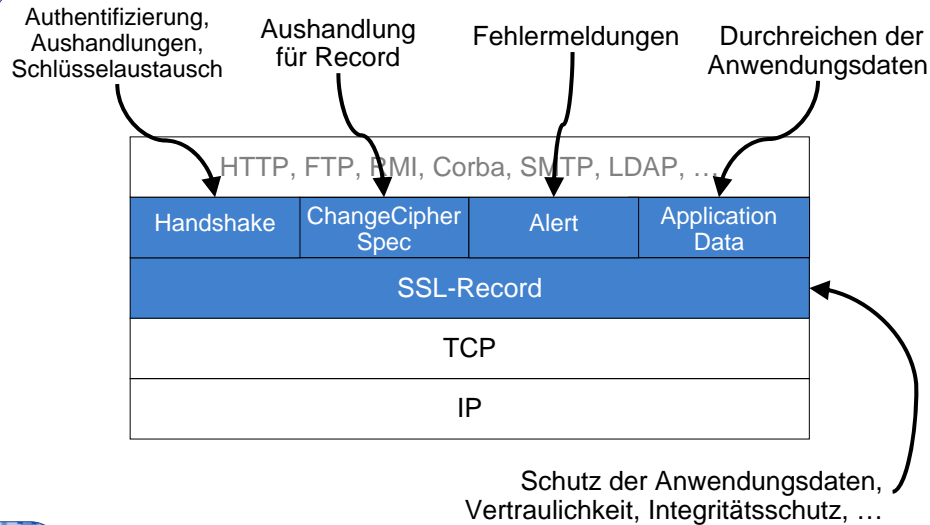
• 2006

- TLS v1.1 RFC 4346

• 2008

- *TLS v1.2* RFC 5246

3



4

- TCP als zuverlässiger Dienst
- Vollständigkeit
- Fehlerfreiheit
- Duplikatsfreiheit
- Übertragungswiederholung nach Timeout oder NAK

```

OpenSSL_add_all_algorithms();
SSL_load_error_strings();
SSL_METHOD* method =
    SSLv2_client_method();
SSL_CTX* ctx =
    SSL_CTX_new(method);

sock = socket(AF_INET,
    SOCK_STREAM, IPPROTO_TCP);

connect(sock, (struct sockaddr*)
    &server_addr,
    sizeof(server_addr));

SSL* ssl = SSL_new(ctx);
SSL_set_fd(ssl, sock);
SSL_connect(ssl); //SSL handshake

SSL_read(); //SSL-gesichert
SSL_write(); //SSL-gesichert

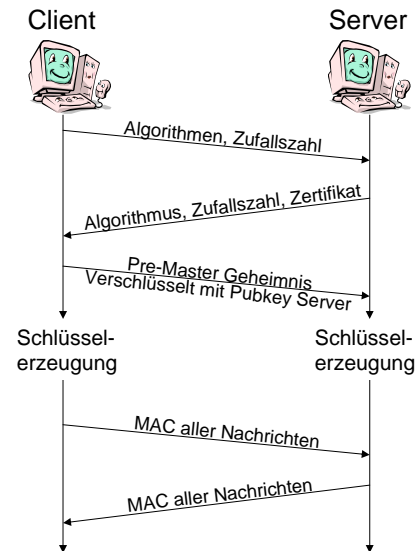
SSL_free(ssl);
close();
    
```

5

### • Handshake

- Aushandlung von Parametern (1,2)
- Authentifizierung des Servers (2)
- Schlüsselaustausch (1,2,3)
- Sicherung des Handshake, Erkennen von Änderungen durch Angreifer (3,4)

(Details später)



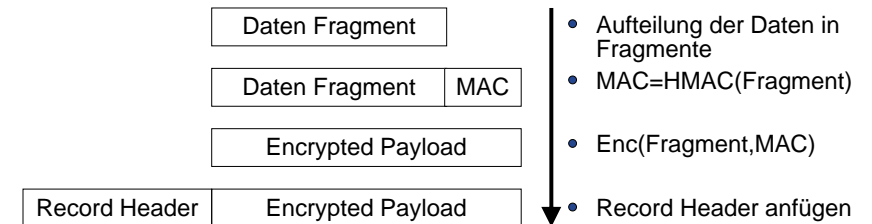
6

### • Handshake

- authentifiziert Server, erstellt Schlüsselmaterial  
→ aber: möchten **gesichert kommunizieren**

### • Record Protokoll

- Aufteilung des Datenstrom in Fragmente
- jedes Fragment einzeln schützen und übertragen



7

## Netzicherheit – Architekturen und Protokolle TLS/SSL und Policy



1. Einführung
2. TLS-Schlüsselaustausch
3. TLS-Datenaustausch
4. Zusammenfassung
5. Policy



## TLS Cipher-Suites

- Definition von *Cipher-Suites*
  - vordefinierte Kombinationen von Sicherungsalgorithmen
  - Präfix: **TLS\_**
  - zwei Sektionen
    - ▶ Algorithmen für Schlüsselaustausch
    - ▶ Algorithmen für Datenaustausch
    - ▶ getrennt durch Symbol: **\_WITH\_**
  - kennt eine Implementierung eine Cipher Suite nicht, so wird diese ignoriert
- Sonderfall: **TLS\_NULL\_WITH\_NULL\_NULL**
  - kein Schutz
  - Zustand bei Initialisierung
  - kein Schlüsselmaterial vorhanden

9



•Der Sonderfall „TLS\_NULL\_WITH\_NULL\_NULL“ wird benötigt, um TLS zu initialisieren. TLS startet als zuerst ohne Schutz, um Schlüssel auszutauschen

•Zusatzinformationen:

- In TLS v1.1 sind mehr als 50 Cipher-Suites definiert.
- CipherSuites für Kerberos existieren schon seit RFC2712, Okt 1999, <http://www.rfc-editor.org/rfc/rfc2712.txt>.

## • Beispiel

**TLS\_DH\_RSA\_WITH\_AES\_256\_CBC\_MD5**

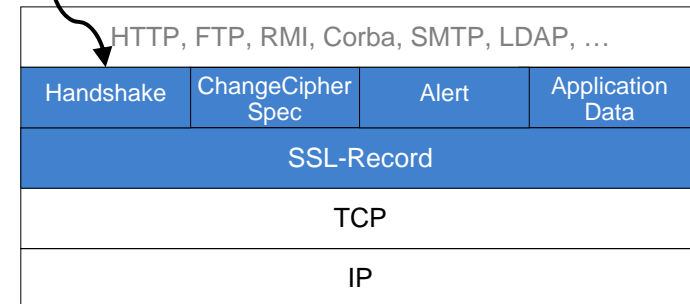
Präfix      Schlüsselaustausch      Trenner      Verschlüsselung, Integritätsschutz  
 → *Handshake-Protokoll*      → *Record-Protokoll*

## • Erklärung

- **TLS**: Präfix zum Erkennen des Protokolls
- **DH**: Schlüsselaustausch durch Diffie-Hellman
- **RSA**: Authentifizierung durch RSA
- **WITH**: Trenner für die Struktur
- **AES\_256\_CBC**: Verschlüsselung der Daten mit AES 256 CBC
- **MD5**: Integritätsschutz mittels HMAC-MD5

10

Authentifizierung,  
Aushandlungen,  
Schlüsselaustausch



11

- TCP als zuverlässiger Dienst
- Vollständigkeit
- Fehlerfreiheit
- Duplikatsfreiheit
- Übertragungswiederholung nach Timeout oder NAK

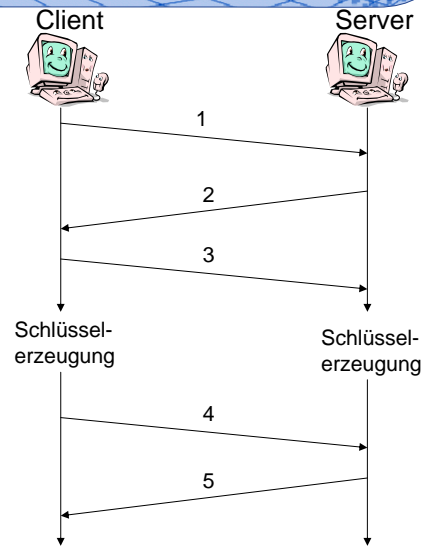


### Algorithmen-Aushandlung

1. Liste von Cipher-Suites  
Zufallszahl
2. Gewählte Cipher-Suite  
Zertifikat, Zufallszahl
3. PreMaster-Secret mit Public Key  
verschlüsselt

### Erzeugung des Master-Secret und der Schlüssel aus dem PreMaster-Secret

4. Message Authentication Code  
(MAC) aller Nachrichten
5. Message Authentication Code  
(MAC) aller Nachrichten



12

#### • „Phase 1“:

- Algorithmenaushandlung in Nachricht 1 und 2
- Schlüsselaustausch in Nachricht (2 und) 3

#### • „Phase 2“:

- Authentifizierung der „Phase 1“
- Die in 4) und 5) versendeten MACs sichern, dass vorher keine Nachricht manipuliert wurde. Das verhindert z.B. Downgrade-Angriffe.

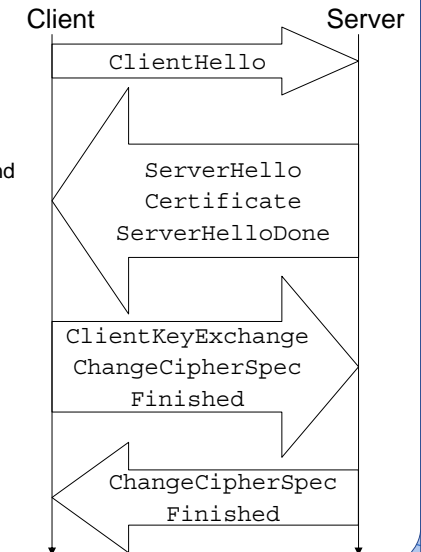
#### • Hinweis

- Diese Darstellung ist vereinfacht. In TLS enthält Nachricht 3 auch Nachricht 4.
- In der Regel wird nur der Server per Zertifikat authentifiziert und die Client-Authentifizierung dann später innerhalb des geschützten Anwendungsprotokoll (HTTP, SMTP, ...) durchgeführt.

### • Schlüsselaustausch *ohne* Diffie-Hellmann

**TLS\_RSA\_WITH\_**

- **ClientHello**: Liste angebotener Cipher-Suites und eine Zufallszahl
- **ServerHello**: Gewählte Cipher-Suites und eine Zufallszahl
- **Certificate**: Server-Zertifikat
- **ServerHelloDone**: Anzeige, dass alle Nachrichten übertragen wurden
- **ClientKeyExchange**: RSA-verschlüsseltes **zufällig gewähltes PreMaster-Secret**
- **ChangeCipherSpec**: Anzeige, dass alle weiteren Nachrichten gesichert werden
- **Finished**: Alle Nachrichten wurden übertragen, MAC über alle Nachrichten



13

#### • Kontrollfrage: Wird in diesem Ablauf PFS erreicht? Wird DH genutzt?

#### • Zusatzinformation

- Interop TLS und Kerberos
- ClientKeyExchange-Nachricht enthält Ticket und PreMaster-Secret  
PreMaster-Secret verschlüsselt mit dem Kerberos-Sitzungsschlüssel  
Server entschlüsselt das Ticket und erhält so den Sitzungsschlüssel

## • Schlüsselaustausch *mit* Diffie-Hellmann

### TLS\_DH\_RSA\_WITH\_

- **ServerKeyExchange** DH-Wert des Servers
- **ClientKeyExchange** DH-Wert des Client
- DH-Werte mittels RSA signiert

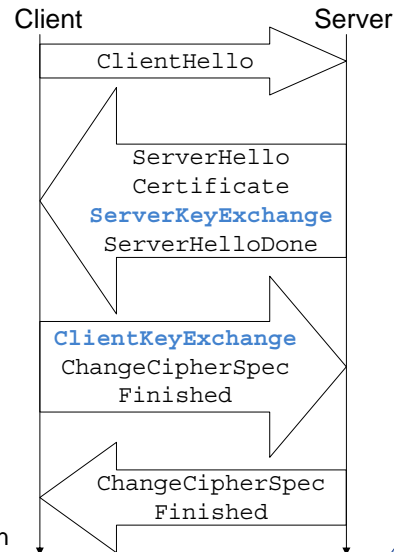
## • TLS unterstützt verschiedene DH-Varianten

- normales (DH), ephemeral (DHE, ähnl. IKE), anonym (DH\_anon, ohne Authentifizierung)

## • Key Derivation: Master-Secret

- Pre-Master-Secret *generiert von Client* oder *DH-Austausch*
- $master\_secret = PRF(pre\_master\_secret, "master\_secret", ClientHello.rand + ServerHello.rand);$
- **PRF** ist aus HMAC konstruiert

→ Client und Server generieren Master-Secret selbst aus Pre-Master-Secret und Zufallszahlen



14

## •Sonderfälle:

- DH: TLS sieht auch vor, dass der DH-Wert des Servers auch im Zertifikat enthalten sein kann. Dann kann die ServerKeyExchange-Nachricht entfallen.
- z.B. TLS\_DH\_RSA\_WITH\_ ...
- DH\_anon: anonymes DH kann verwendet werden, wenn keine Zertifikate vorhanden sind oder sein können. Einsatzszenarien eher theoretisch, allerdings im Privacy-Umfeld denkbar (siehe Kapitel Datenschutz und Privatsphäre)
- z.B. TLS\_DH\_anon\_WITH\_ ...

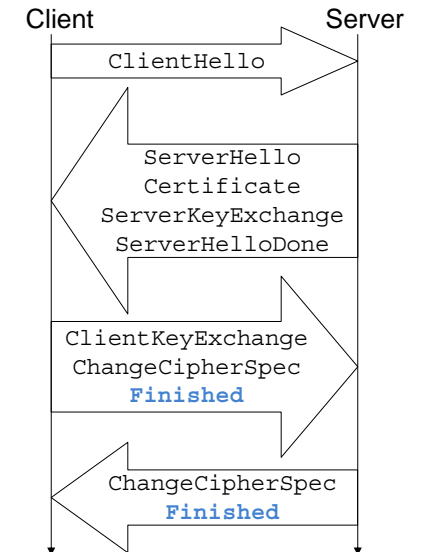
## • Downgrade-Angriff

- Angreifer löscht Cipher-Suites
- Ziel: Client und Server einigen sich auf ein schwaches Verfahren
- Bsp: NULL-Verfahren oder DES

## • Lösung

- Signierter MAC über alle ausgetauschten Nachrichten
- in der **Finished**-Nachricht

→ Überprüfung ob gesendete Nachrichten bei Empfänger unverändert ankamen

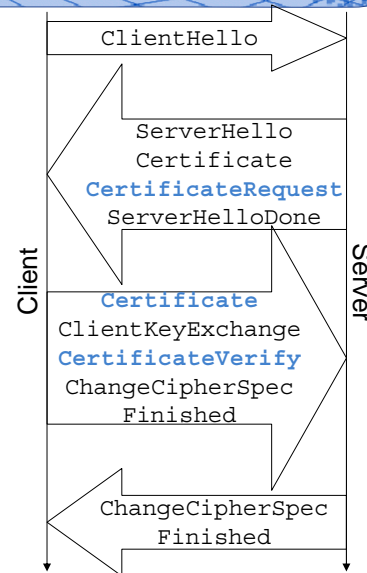


15



## • Client-Authentifizierung

- **CertificateRequest**: Anforderung vom Server, dass Client ein X.509-Zertifikat zur Identifizierung verwenden muss
  - ▶ kann nicht vom Client angeboten werden
  - ▶ Beschränkung des Dienstes auf bestimmte Nutzer
- **Certificate**: enthält das Zertifikat
  - ▶ Analog zur Zertifikat-Nachricht des Servers
- **CertificateVerify** ist ein signierter Hash über die bisherigen Nachrichten
  - ▶ Nachweis, dass der Client auch den dazugehörigen privaten Schlüssel besitzt



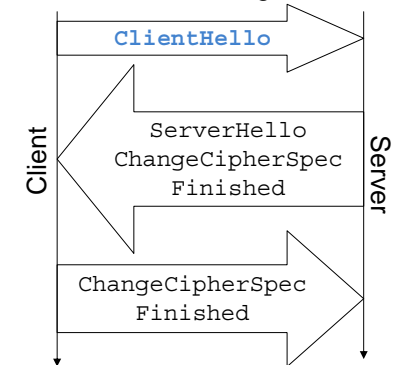
16

Der Server kann die Client-Authentifizierung erzwingen mit „CertificateRequest“.

## Vermeidung rechenintensiver Schlüsselaushandlung

- Erster Verbindungsaufbau wie gezeigt
  - Server wählt und sendet **Sitzungs-ID** in ServerHello
  - Speichert Master-Secret mit Sitzungs-ID
- **Wiederaufnahme** einer Verbindung
  - Client verwendet **Sitzungs-ID** in ClientHello
  - Server zeigt Einverständnis durch Übertragung der gleichen Sitzungs-ID in ServerHello
  - Überspringen der restlichen Aushandlung
- Diskussion
  - wann benötigt man das Verfahren?
  - Server hält Zustand, andere Möglichkeit?

### Zweiter Verbindungsaufbau:



17

## •Motivation

- Ein Schlüsselaustausch kann teuer sein. Die Aushandlung dauert mehrere Round-Trip-Times (RTTs), was bei großer Latenz ein Problem darstellen kann. Weiterhin sind Verfahren wie DH, RSA und Zufallszahlengenerierung u.U. für ein Gerät mit schwacher CPU durchaus zeitaufwändig. Für Anwendungen wie VoIP-Telefone kann dies ein Problem darstellen, da durch Wechseln von IP-Adressen u.U. gesicherte Verbindungen wieder neu aufgebaut werden müssen.

## •Zusatzinformation:

- Eine zustandslose Variante wurde erst kürzlich in RFC4507 standardisiert.
- RFC3407: „Transport Layer Security (TLS) Session Resumption without Server-Side State“, Mai 2006, <http://www.rfc-editor.org/rfc/rfc4507.txt>
- Der Client zeigt beim ClientHello an, dass er SessionTickets unterstützt.
- Der Server liefert ein verschlüsseltes und authentifiziertes SessionTicket an den Client am Ende des erfolgreichen Schlüsselaustausches.
- Wenn der Client später eine Sitzung wieder aufbauen will, muss er nur das SessionTicket in der ersten Nachricht vorlegen. Der Server kann dieses SessionTicket annehmen oder eine normale Aushandlung anstossen.

## Neuaushandlung des Schlüsselmaterials

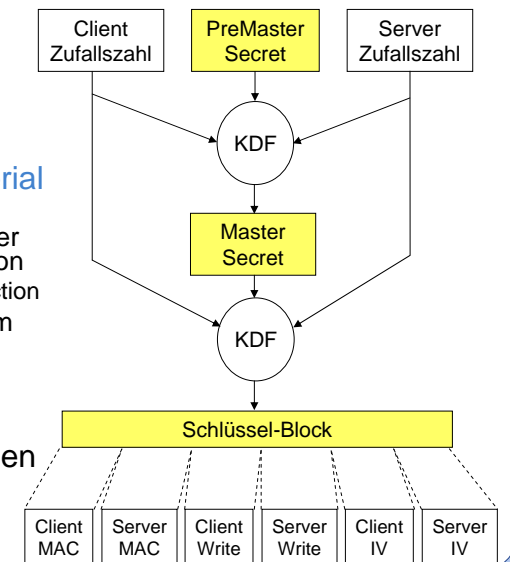
- Jederzeit von Client oder Server aus möglich
  - Aushandlung unter Schutz der bestehenden Verbindung
  - Neuaushandlung des PreMaster-Secrets
- Ablauf
  - Triggern des Rekeying
    - ▶ Server sendet eine HelloRequest-Nachricht
      - Anforderung an Client Rekeying zu beginnen
    - ▶ Client sendet eine ClientHello-Nachricht
  - Neuer Handshake wird durchgeführt
- Neuaushandlung nicht erwünscht
  - Ignorieren der Nachricht oder
  - Alert-Nachricht

18

## •Wiederholung:

- Verschiedene Gründe für Rekeying existieren. Beispiele:
  - Die Menge an bereits geschützten Bytes/Paketen hat ein Limit erreicht. (Konfliktwahrscheinlichkeit für IV, siehe auch Geburtstagsparadox)
  - Das Alter der Schlüssel hat ein Limit erreicht.

- Bisher: Handshake
  - Pre-Master Secret
- Generierung des Master-Secret für Records-Protokoll
- Erzeugung Schlüsselmaterial
  - Verwendung einer Pseudo-Zufalls-Funktion (PRF) in der Schlüsselerzeugungsfunktion
    - ▶ KDF: Key Derivation Function
  - PreMaster Secret ist geheim
  - beide Zufallszahlen sind öffentlich bekannt
- Länge des Key Block ist Summe der Schlüssellängen
  - Zerschneiden des Key-Blocks in Schlüssel



19

- „write key“ bezeichnet die Schlüssel zum Verschlüsseln (encryption keys)
- Kontrollfrage: Was passiert falls Klient und Server jedes Mal gleiche DH-Parameter wählen?
- Kontrollfrage: Was passiert bei Sitzungswiederaufnahme?
- Kontrollfrage: Warum ist es sinnvoll, dass der Server für die Sitzungswiederaufnahme das Master Secret speichert?

## Netzicherheit – Architekturen und Protokolle TLS/SSL und Policy

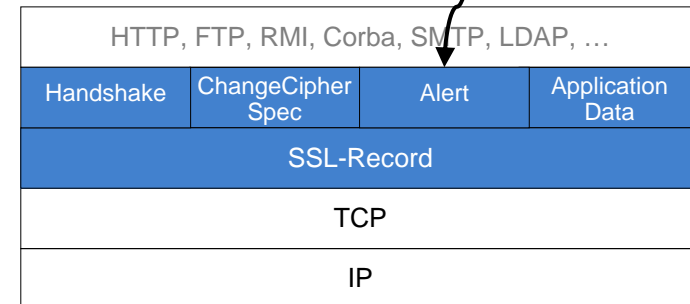


1. Einführung
2. TLS-Schlüsselaustausch
3. TLS-Datenaustausch
4. Zusammenfassung
5. Policy



## SSL Protokollstruktur

Fehlermeldungen



21

Netzicherheit – Architekturen und Protokolle

TLS/SSL und Policy

Institut für Telematik [www.tm.kit.edu](http://www.tm.kit.edu)  
Karlsruher Institut für Technologie (KIT)

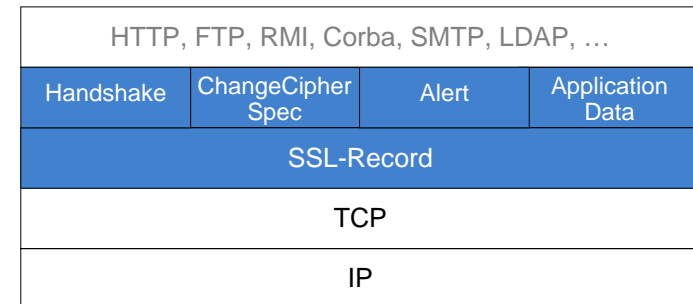


- TCP als zuverlässiger Dienst
- Vollständigkeit
- Fehlerfreiheit
- Duplikatsfreiheit
- Übertragungswiederholung nach Timeout oder NAK

## Alert-Protokoll

- zwei Arten von Fehlern
  - **Fataler Fehler**: sofortiger Verbindungsabbruch
  - **Hinweis**: **CloseNotify** oder **NoRenegotiation**
    - ▶ später mehr zu **CloseNotify**
- Fehler des Zertifikats
  - Signatur-Algorithmus nicht unterstützt
  - Signatur falsch
  - Zertifikat zurückgezogen
  - CA nicht vertrauenswürdig
- Fehler beim Datenempfang
  - MAC des Schlüsselaustauschs falsch
  - entschlüsseltes Paket nicht authentisch
  - überlanges Paket

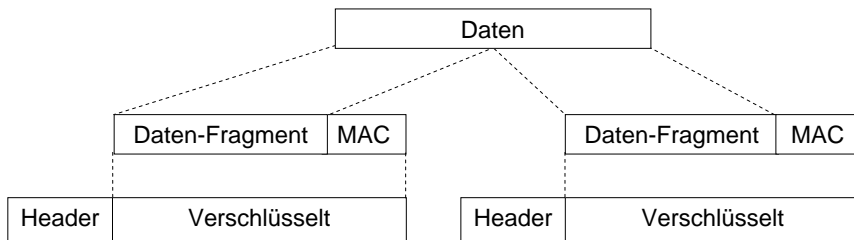
22



Schutz der Anwendungsdaten,  
Vertraulichkeit, Integritätsschutz, ...

23

- TCP als zuverlässiger Dienst
- Vollständigkeit
- Fehlerfreiheit
- Duplikatsfreiheit
- Übertragungswiederholung nach Timeout oder NAK



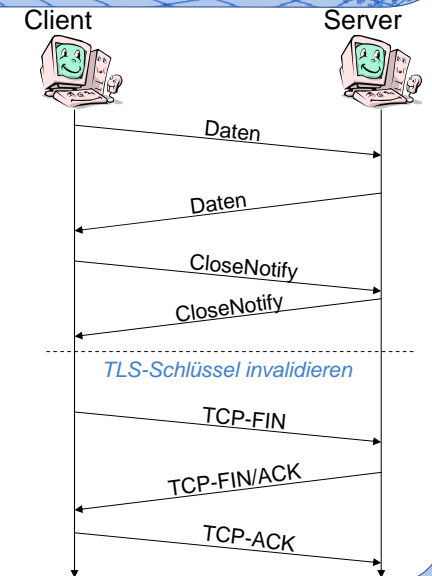
## Funktionsweise des Record Layers

- Zerlegen der Daten in **Fragmente**
- Berechnen des **MAC** und **Anhängen** an das Fragment
- **Verschlüsseln** von (Fragment + MAC)
  - ▶ Null-Verschlüsselung während Handshake
- Anfügen des **Record Headers**
  - ▶ Type, Version, Länge

24

Verteilung von Datenstrom auf TLS/TCP/IP-Pakete

- Einleitung durch **CloseNotify**
  - Alert-Nachricht des Typs Hinweis
  - Schlüsselmaterial/Kontext löschen
- Schutz vor Verkürzungsangriffen
  - Truncation-Angriffen
  - bis SSLv2 keine Nachricht für SSL-Session Ende
    - ▶ Beenden der Session bei Ende der TCP-Verbindung
    - ▶ Angreifer kann TCP-FIN an Alice schicken. Für Bob ist SSL-Session noch offen → je nach Implementierung Fehlverhalten möglich
- **Fehlerfälle**
  - Empfang eines TCP-FIN ohne vorheriges **CloseNotify**
    - ▶ Lässt auf möglichen Angriff schließen
    - ▶ Sitzung darf nicht wieder aufgenommen werden, da TLS nicht korrekt abgebaut
  - Auslassen der zweiten **CloseNotify**-Nachricht
    - ▶ End-Timer in der Anwendung nötig
    - ▶ Direktes close() durch Peer



25

## Netzicherheit – Architekturen und Protokolle TLS/SSL und Policy



1. Einführung
2. TLS-Schlüsselaustausch
3. TLS-Datenaustausch
4. Zusammenfassung
5. Policy



## TLS Zusammenfassung

- Transport Layer Security
  - TLS erzeugt einen **sicheren Kanal**
  - mehrere Verbindungen in einem TLS-Kanal
  - Server-Authentifizierung
    - ▶ Client-Authentifizierung durch Anwendung
    - ▶ z.B. Name/Passwort
  - Wiederaufnahme einer Sitzung

TLS für TCP entwickelt, welche Änderungen für UDP notwendig?



## Netzicherheit – Architekturen und Protokolle TLS/SSL und Policy



1. Einführung
2. TLS-Schlüsselaustausch
3. TLS-Datenaustausch
4. Zusammenfassung
5. Policy



## Übersicht

### Schlüsselaustausch-Protokoll



Kanal zur Schlüsselaushandlung

### Schlüsselaustausch-Protokoll



Gesicherte Kommunikation

- wie beschreibt man den „gesicherten Kanal“?
- wie sieht die Schnittstelle zur Anwendung aus?
  - explizite Schnittstelle für Anwendung → TLS
  - implizite Schnittstelle für Anwendung → IPsec



- **Socket-Schnittstelle**
  - Transport Layer Security (TLS)
    - ▶ UDP Version DTLS
  - nutzen einer speziellen API im Programm
    - ▶ vollständig im User-Space
- **Netzwerk-Protokollstapel**
  - IPsec
  - transparent für die Anwendung
    - ▶ keine Änderungen an der Anwendung notwendig
  - Änderung am Betriebssystem
    - ▶ Schutz durch Administration, nicht durch Anwendung/Benutzer
  - Ort der Schutzfunktionalität frei wählbar
    - ▶ Endsystem: komplexe Konfiguration
    - ▶ Gateway: zentraler Punkt, aber kein Schutz im internen Netz

30

- in etwa: TLS = SSL
- Vorteile Socket-Schnittstelle:
  - Es ist klar, welche Verbindung gemeint ist
- Vorteile Netzwerk-Protokollstapel
  - Wesentlich mächtiger
  - Aggregation möglich
- (Open)Solaris bietet eine Socket-Schnittstelle für IPsec. Hiermit kann dann die Anwendung Schutz durch IPsec anfordern.

- **Welche** Pakete sollen **wie** geschützt werden?
- Selektoren
  - **ausgehende Pakete**: Verbindungskontext
  - **eingehende Pakete**: Verbindungskontext
- Klassifikation durch Anwendungsport
  - Nutzer kontrolliert explizit durch Portwahl
  - HTTP auf Port 80 → HTTPS auf Port 443
  - Alternative: StartTLS durch Anwendung (smtp, http, ...)
    - ▶ nur ein Port notwendig, Anwendungen entscheiden ob gesichert kommuniziert wird durch Umschalten auf TLS

31

- StartTLS
  - Server gibt an, dass er StartTLS unterstützt
  - Client signalisiert, dass er StartTLS nutzen möchte
  - TLS-Aushandlung und –Nutzung
- Vorteil:
  - Nur ein Port für verschlüsselte und unverschlüsselte Verbindung
  - Flexiblere Entscheidung, ob Schutz notwendig (z.B. abhängig vom Inhalt, Adressen, ...)

- Welche Pakete sollen wie geschützt werden?
- Selektoren für **ausgehende Pakete**
  - Quell- und Zieladresse (IPv4 oder IPv6)
  - Schicht-4-Protokoll (TCP oder UDP), Quell- und Zielports
  - Name (nur auf Endsystem)
    - ▶ Voll qualifizierter DNS-Benutzername (z.B.: [administrator@tm.uka.de](mailto:administrator@tm.uka.de))
    - ▶ X.500 name
- Selektoren für **eingehende Pakete**
  - Zieladresse (IPv4 oder IPv6)
  - IPsec-Protokoll
  - SPI (Security Parameter Index)
- Klassifikation durch Regelwerk der **Security Policy Database (SPD)**
  - Festlegung der Verarbeitungsstrategien

32

## •Security Policy Database (SPD):

Jedes eingehende und ausgehende Paket wird mit Hilfe der SPD betrachtet  
Regelwerk aus Selektoren und Verarbeitungs-Strategien  
Erster übereinstimmender Eintrag legt die Verarbeitungsstrategie fest  
SPD muss linear geordnet sein  
Verwerfen des Pakets, wenn keine gültige Regel in der SPD

## •Drei unterschiedliche Verarbeitungsstrategien

Drop: Sofortiges Verwerfen des Pakets  
Bypass: Weiterleitung des Pakets ohne IPsec-Schutzfunktion  
IPsec: Anwenden der IPsec-Schutzfunktionalität auf das Paket

## •IPsec-Paketverarbeitung

Bestimmung der zu verwendenden Sicherheitsbeziehung  
Aushandlung nicht-existenter Sicherheitsbeziehungen  
Misserfolg: Verwerfen des Pakets  
Nach erfolgreicher Aushandlung: IPsec-Verarbeitung

## Beispiel: SPD auf einem Security Gateway

Regel	Src-Addr	Dest-Addr	Src-Port	Dest-Port	Protocol	Action
1	SG1	SG2	500	500	UDP	Bypass
2	SG1	SG2	*	*	*	IPsec – SA 1
3	Host 1	*	*	*	*	IPsec – SA 2
4	Netz 1	Netz 2	*	*	*	IPsec – SA 3

- Regel 1: IKE-Verkehr wird nicht durch IPsec geschützt
- Regel 2: Sonstiger Verkehr zwischen den beiden SGs wird durch SA 1 geschützt
- Regel 3: Verkehr von Host 1 wird durch SA 2 geschützt
- Regel 4: Sämtlicher Verkehr von Netz 1 (außer Host 1) ins Netz 2 wird durch SA 3 geschützt



33

## • IPsec

- ☺ mächtiges, flexibles Verfahren
- ☺ Administrator kann in Policy eingreifen
- ☺ Aggregation mehrerer Verbindungen möglich
- ☺ schützt transparent alle Schicht-4-Protokolle
- ☹ hohe Komplexität

## • TLS

- ☺ einfaches Verfahren
- ☺ Anwendungsnahe
- ☺ nutzt Kontext von TCP
- ☹ Aggregation Verbindungen unterschiedl. Systeme nicht möglich
- ☹ TLS schützt nur Anwendungsdaten, nicht Schicht 3/4
- ☹ Administrator kann nicht in Policy eingreifen, von Anwendung def.

34

Die Aggregation verschiedener Verbindungen kann mittels TLS nur durch zusätzlichen Aufwand geschehen. Hierfür muss i.d.R. dann zusätzliche Software installiert werden, welche nicht unerheblichen Aufwand verursacht (siehe auch SSL-VPNs).

## • Fazit

- Policy definiert „was“ „wie“ geschützt wird
- Umsetzung und Ort hängen vom Protokoll ab

## • Sicherheitstechniken

- IPsec
  - ▶ komplex aber mächtig
  - ▶ schwierig für den Anwender
- TLS
  - ▶ einfach aber weniger mächtig
  - ▶ für den Anwender einfacher

35

- **SSL and TLS – Designing and Building Secure Systems**, E. Rescorla; Addison-Wesley, 2001
  - Buch vom Autor des TLS-Standards
- **Private Communication in a Public World**, C. Kaufmann, R. Perlman, M. Speciner; Network Security, Prentice Hall; 2003
  - Allgemeines Buch, hier findet man sehr viel
- **The Transport Layer Security (TLS) Protocol Version 1.2**, T. Dierks, E. Rescorla, RFC 5246, August 2008
  - Aktueller TLS Standard