

# SSL/TLS

Stephan Eckner  
stephan at eckner dot org

9. Juli 2003

# Überblick

- Verschlüsselung und Netzverkehr
  - Verschlüsselung auf Anwendungsebene
  - Verschlüsselung auf Linkebene
  - Verschlüsselung auf IP- Ebene
  - Veranschaulichung anhand des TCP/IP Layer Modells
- SSL Übersicht
  - Geschichte des SSL-Protokolls
  - Überblick über das SSL-Protokoll
  - Designziele des SSL-Protokolls
- Crash-Kurs Kryptographie
  - Kryptosysteme
  - Symmetrische Algorithmen
  - Message Authentication Codes und Hash Funktionen
  - Asymmetrische Algorithmen

- Diffie-Hellman Algorithmus
  - RSA Algorithmus
  - Digitale Signatur mit dem RSA Algorithmus
  - Eigenschaften Asymmetrischer Algorithmen
  - Hybrid-Protokolle
  - Zertifikate und Public Key Infrastrukturen
  - Probleme von Public Key Infrastrukturen
- 
- SSL Record Protokoll
    - Aufgaben des SSL-Record Protokolls
    - Aufbau des SSL-Record Protokolls
    - Aufbau eines SSL-Record Paketes
- 
- SSL Handshake Protokoll
    - Aufgaben des Handshake Protokolls
    - Aufbau des Handshake Protokolls
    - Phase I – Client Hello
    - Phase I – Server Hello
    - Phase II – Server Authentication und Key Exchange
    - Phase III – Client Authentication und Key Exchange

- Phase IV – Finish
  - Kreierung der Keys
- SSL Alert Codes
- SSL in Action
- Links und Literatur
- Diskussion

# Verschlüsselung auf Anwendungsebene

- anwendungsspezifisch, insbesondere *nicht transparent*
- Vertrauensbeziehung zwischen User-Level Programmen
- Beispiele
  - TLS/SSL
  - SSH
  - PGP, S/MIME

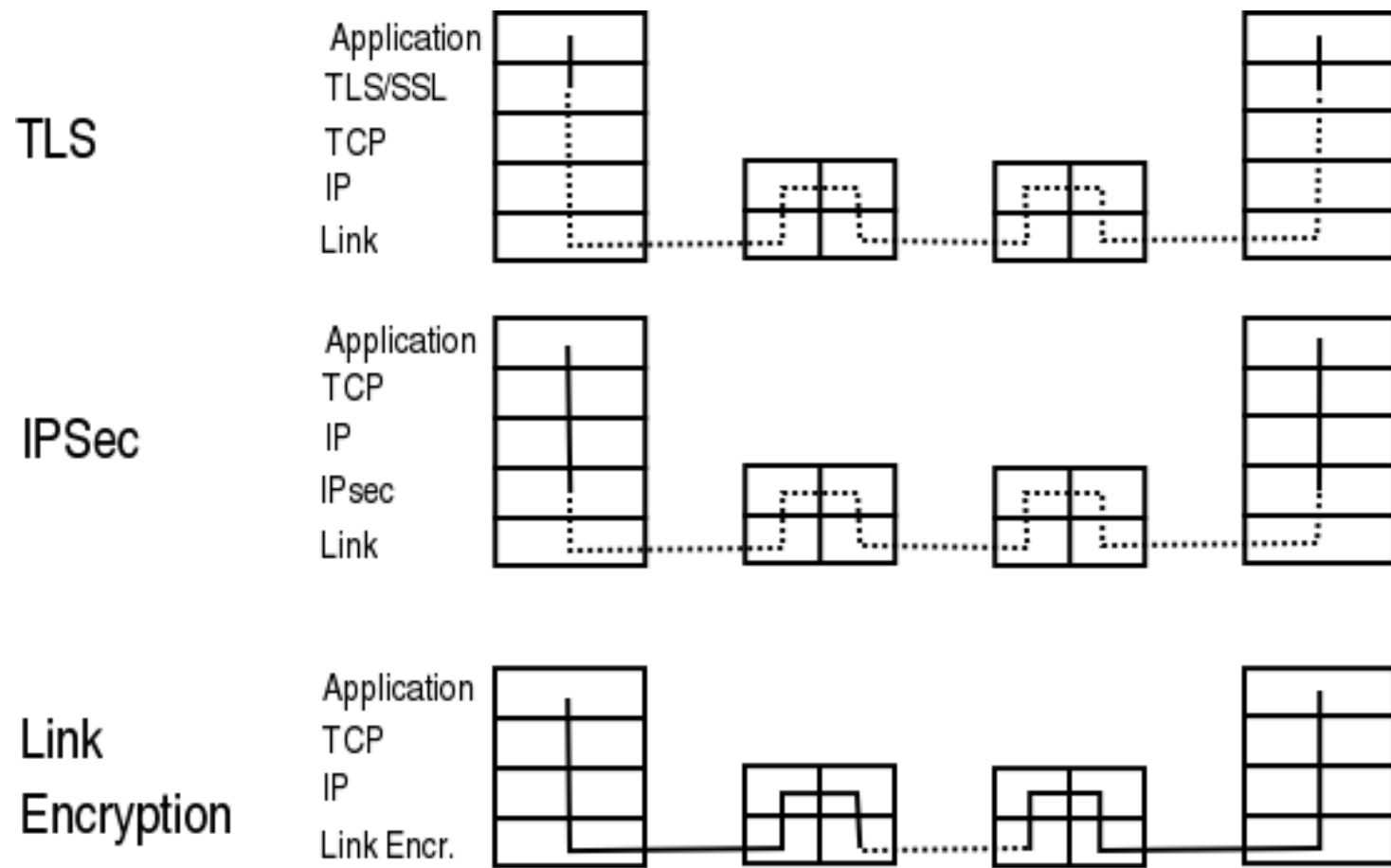
# Verschlüsselung auf Linkebene

- transparent
- kann nicht geroutet werden
- Beispiele
  - ISDN-Krypto-Router
  - PPTP

# Verschlüsselung auf IP-Ebene

- transparent
- kann geroutet werden
- koexistiert mit unverschlüsseltem IP-Traffic
- Beispiel
  - IPSec

# Veranschaulichung anhand des TCP/IP Layer-Modells



# Überblick

- Verschlüsselung und Netzverkehr
- **SSL Übersicht**
- Crash-Kurs Kryptographie
- SSL Record Protokoll
- SSL Handshake Protokoll
- SSL Alert Codes
- SSL in Action
- Links und Literatur
- Diskussion

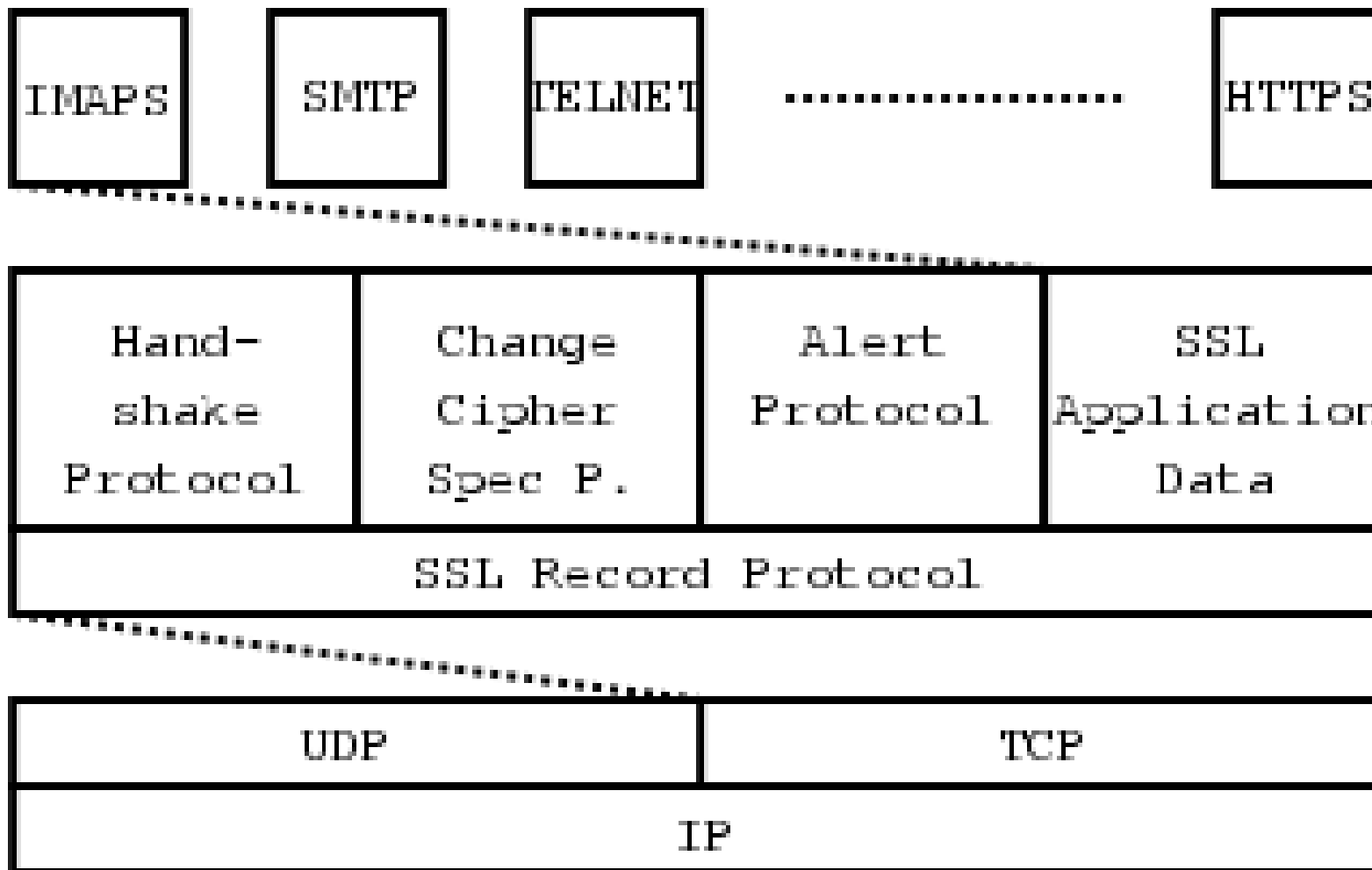
# Geschichte des SSL-Protokolls

- SSL Anfang der 90er Jahre von Netscape entwickelt
- SSL Version 1.0
  - nur Netscape-intern verwendet
  - diverse Sicherheitsprobleme
- SSL Version 2.0
  - enthalten in Navigator 1.0 - 2.x
  - einige subtile Sicherheitsprobleme
- SSL Version 3.0
  - released März 1996
  - enthalten in Navigator 3.0 - 4.x
  - einige sehr subtile Sicherheitsprobleme

# Geschichte des SSL-Protokolls

- SSL Version 3.1/TLS Version 1.0
  - released als RFC 2246, Januar 1999
  - enthalten in Mozilla, Navigator 6.x
  - kann nach öffentlicher Diskussion als Teil des RFC-Release-Verfahrens und mehrerer öffentlicher Kryptoanalysen (s. Abschnitt *Links und Literatur*) als sehr sicher eingestuft werden

# Überblick über das SSL-Protokolle



## Designziele des SSL-Protokolls

Zitat RFC 2246: *"The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery."*

- Verschlüsselung der Verbindung mit symmetrischer Kryptographie *"DES, RC4, etc."*.
- Generierung der Session-Keys durch ein separates Protokoll (TLS Handshake Protocol)
- Sichere (gegen Mitlauschen immune) Generierung der Session-Keys.
- Integritätsüberprüfung mittels *"keyed MAC"* und *"secure hash functions SHA, MD5, etc."*.
- Identitätsüberprüfung mittels asymmetrischer Kryptographie *"e.g. RSA, DSS, etc."*.

- Immunität gegen aktive Angreifer.

# Überblick

- Verschlüsselung und Netzverkehr
- SSL Übersicht
- **Crash-Kurs Kryptographie**
- SSL Record Protokoll
- SSL Handshake Protokoll
- SSL Alert Codes
- SSL in Action
- Links und Literatur
- Diskussion

# Kryptosystem

*κρυπτος* – geheim, *γραφειν* – schreiben  
*Kryptographie* – Entwicklung von Codes

Ein *Kryptosystem* besteht aus vier Komponenten:

- Eine Menge von Klartexten  $K$
- Eine Menge von Geheimtexten  $G$
- Einem Paar von Funktionen  
 $f : K \longrightarrow G$  (Verschlüsselungsfunktion) und  
 $g : G \longrightarrow K$  (Entschlüsselungsfunktion)  
mit  $g(f(k)) = k$  für alle  $k \in K$

# Beispiele

- Rot13

$$K = \{a, b, \dots, z\}$$

$$G = \{a, b, \dots, z\}$$

$f(\text{Buchstabe}) = 13$ . darauffolgender Buchstabe im Alphabet

$$(a \longrightarrow m, b \longrightarrow n, \dots)$$

$g(\text{Buchstabe}) = 13$ . vorangehender Buchstabe im Alphabet

$$(m \longrightarrow a, n \longrightarrow b, \dots)$$

- XOR

- einfache Buchstabenpermutation

# Algorithmen und Schlüssel

- Problem: Skalierbarkeit
- Will ich mit  $n$  verschiedenen Personen verschlüsselt kommunizieren, so benötige ich auch  $n$  verschiedene Verschlüsselungsfunktionen
- Grosse Menge an gemeinsamen Geheimnissen
- Idee: Auftrennung der Verschlüsselungsfunktion in
  - einen möglichst großen öffentlich bekannten Teil, genannt *Algorithmus* (komplexe Beschreibung der Verschlüsselungsmethode) und
  - einen möglichst kleinen geheimzuhaltenden Teil, genannt *Schlüssel*, so dass
  - Algorithmus und Schlüssel zusammen eine Verschlüsselungsfunktion ergibt

# Kryptosystem, zweiter Versuch

Ein *Kryptosystem* besteht aus *fünf* Komponenten:

- Eine Menge von Klartexten  $K$
- Eine Menge von Geheimtexten  $G$
- Eine Menge von Schlüsseln  $S$
- Einem Paar von Funktionen  
 $f : S \times K \longrightarrow G$  (Verschlüsselungsalgorithmus) und  
 $g : S \times G \longrightarrow K$  (Entschlüsselungsalgorithmus),  
so dass für alle  $s \in S$  gilt  
 $g(s, f(s, k)) = k$  für alle  $k \in K$

# Beispiel

- *Rot* ist Algorithmus,  $0 < n < 26$  ist Schlüssel:  
 $f(n, \text{Buchstabe}) = n$ ter darauffolgender Buchstabe im Alphabet  
 $g(n, \text{Buchstabe}) = n$ ter vorangehender Buchstabe im Alphabet
- Vorteil: ein (bekannter) Algorithmus und  $k$  (geheime) Schlüssel für  $k$  Personen anstelle von  $k$  (geheimen, verschiedenen) Verschlüsselungsfunktionen für  $k$  Personen
- enorme Reduzierung der geheimzuhaltenden Informationen

# Symmetrische Algorithmen

- Algorithmen heissen *symmetrisch*, falls für jeden Schlüssel  $k$  die Entschlüsselungsfunktion  $g_k$  gleich der Verschlüsselungsfunktion  $f_k$  ist.
- Beispiele
  - Rot13 (nicht jedoch Rot14 !)
  - XOR
  - DES
  - Rijndael (aka AES)
  - RC2
  - RC4
  - Blowfish
  - IDEA
  - ...

# Eigenschaften Symmetrischer Algorithmen

## Symmetrische Algorithmen

- gewährleisten Vertraulichkeit zwischen Personen, die über ein gemeinsames Geheimnis verfügen
- authentifizieren Personen, die über ein gemeinsames Geheimnis verfügen
- bieten *keine* Möglichkeit, ein gemeinsames Geheimnis über ungesicherte Kanäle zu erzeugen/auszutauschen
- gewährleisten *nicht* per se die Integrität der übermittelten Daten
- skalieren schlecht:  $n$  Personen benötigen  $\frac{2(n-1)}{2}$  Schlüssel, um sich untereinander zu authentifizieren bzw. vertraulich miteinander zu kommunizieren

# Message Authentication Codes (MAC)

- Verschlüsselung von Daten gewährleistet nicht die Integrität, sondern nur die Vertraulichkeit
- Idee: Erzeuge einen *Fingerprint* der Daten:
  - Sei  $f$  ein (symmetrischer) Algorithmus und  $k$  ein Schlüssel
  - Teile den Klartext in  $N$  Blöcke  $E_1, \dots, E_N$
  - Verschlüssele den ersten Block des Klartextes mit  $f_k$ :  
 $O_1 = f_k(E_1)$
  - XORe  $O_1 \oplus E_2$
  - Bilde  $O_2 = f_k(O_1 \oplus E_2)$  usw.
  - Bilde schließlich  $O_N == f_k(O_{N-1} \oplus E_N)$
- obiges Verfahren bezeichnet man auch als *FIPS 113* bzw. *ANSI X9.17*

# Hash Funktionen

- Problem von MACs: nicht sehr performant
- Idee: Benutze eine schnelle Funktion (Hashfunktion) zum 'Zusammenstauchen' der Daten und verschlüssele dann nur das 'Zusammengestauchte' (*keyed Hash*)
  - eine Hashfunktion ist eine Funktion, die beliebig viel Input akzeptiert und daraus einen Output (Hash) fester Länge erzeugt
  - Eigenschaft: Es muss sehr schwierig sein, zwei Datensätze zu finden, die den selben Hash erzeugen
- Beispiele
  - MD5
  - SHA-1
  - RIPEMD-160

# HMAC

- HMAC: MAC mit Hash Funktionen, auch *keyed Hash* genannt
- Idee: Mixe Schlüssel mit Klartext und wende Hash-Funktion darauf an
- Vorteile:
  - Genauso schnell wie Hash-Funktion
  - Funktioniert mit beliebiger Hash-Funktion
- Üblich: HMAC-MD5, HMAC-SHA-1, HMAC-RIPEMD-160
- Genaueres: siehe RFC 2104

# Asymmetrische Algorithmen

- Problem herkömmlicher Kryptographie mit Symmetrischen Algorithmen:
  - Alles basiert auf der a priori Existenz eines gemeinsamen geheimen Schlüssels
  - kein Verfahren zum Austausch/Erzeugung dieses geheimen Schlüssels
- Bis 1976: Diffie/Hellman *"New Directions in Cryptography"*
  - Verfahren zur Erzeugung eines gemeinsamen geheimen Schlüssels über unsichere Kanäle
- 1978: Rivest/Shamir/Adleman *RSA-Algorithm*
  - Verfahren zur Ver- und Entschlüsselung

# Diffie-Hellman Algorithmus

- Trick: Sei  $x = c^a \bmod n$ , dann kann aus  $x$  und  $c$  nicht  $a$  errechnet werden (*Problem der Berechnung des diskreten Logarithmus*)
- Algorithmus:
  - $A$  und  $B$  einigen sich auf Modulus  $p$  und Basis  $c$ , wobei
  - $p$  Prim ist und
  - $c$  so gewählt, dass  $c^n \neq 1 \bmod p$  für  $0 < n < p - 1$  (d.h.  $\bar{c}$  generiert die abelsche Gruppe  $(\mathbb{Z}_p \setminus \{0\}, \cdot)$ )
  - $A$  wählt Exponent  $a$  und berechnet  $x = c^a \bmod p$
  - $B$  wählt Exponent  $b$  und berechnet  $y = c^b \bmod p$
  - $A$  und  $B$  übermitteln  $x$  und  $y$  (aus denen ein Angreifer nicht  $a$  oder  $b$  berechnen kann, s.o.)
  - $A$  berechnet  $y^a = (c^b)^a \bmod p = e$
  - $B$  berechnet  $x^b = (c^a)^b \bmod p = e$
  - $e$  ist das gemeinsame Geheimnis von  $A$  und  $B$

# RSA Algorithmus

- Trick1: Sei  $x = c^a \bmod n$ , dann kann aus  $x$  und  $c$  nicht  $a$  errechnet werden (*Problem der Berechnung des diskreten Logarithmus*)
- Trick2: Sei  $n = pq$ ,  $p, q$  Prim, dann gilt für alle  $k$  und  $0 < m < n - 1$ :  
 $m^{k(p-1)(q-1)+1} = m \bmod n$  (Spezialfall des Satzes von Euler)
- Algorithmus:
  - $A$  findet Primzahlen  $p$  und  $q$  und berechnet  $l = (p - 1)(q - 1) + 1$
  - $A$  Zerlegt  $l$  in  $l = ed$
  - $A$  veröffentlicht  $n$  und  $e$  (Public Key)
  - $A$  behält  $d$  für sich (Private Key)
  - $B$  verschlüsselt einen Text  $m$ ,  $0 < m < n - 1$ , indem sie  $c = m^e \bmod n$  berechnet
  - $A$  entschlüsselt  $c$ , indem sie  
 $c^d \bmod n = m^{ed} \bmod n = m \bmod n$   
berechnet

# Digitale Signatur mit dem RSA Algorithmus

- Der Symmetrie in der obigen Berechnung wegen kann  $A$  natürlich auch einen Text  $m'$  mit seinem privaten Schlüssel  $d$  verschlüsseln:

$$c' = m'^d \bmod n$$

- Das Ergebnis kann jeder mit dem öffentlichen Schlüssel  $e$  wieder entschlüsseln:

$$c'^e \bmod n = m'^{de} \bmod n = m' \bmod n$$

- Diesen Vorgang nennt man auch Digitale Signatur.
- Um Platz zu sparen, staucht man die zu signierenden Daten vorher mit einem der erwähnten Hash-Algorithmen (MD5, SHA1, RIPEMD-160) zusammen.

# Eigenschaften Asymmetrischer Algorithmen

- Schlüsseltausch über unsichere Kanäle möglich (Diffie-Hellman)
- Skalierbares Keymanagement (RSA-Algorithmus): für eine Authentisierung/vertrauliche Kommunikation von  $n$  Personen untereinander benötigt man  $n$  Keys, nicht  $\frac{2(n-1)}{2}$
- Problem: Asymmetrische Algorithmen benötigen um mehrere Größenordnungen mehr Ressourcen
  - Lösung: Hybrid-Protokolle
- Einziges verbleibendes Problem:  $A$  und  $B$  können zwar Schlüssel tauschen und vertraulich miteinander kommunizieren, sich aber nicht gegenseitig identifizieren (woher weiss  $A$ , ob  $B$  die ist, die sie zu sein vorgibt?)
  - Lösung(sversuch): Public Key Infrastruktur

# Hybrid Protokolle

- Kombinieren die Vorteile von Symmetrischen und Asymmetrischen Algorithmen
- Zur Authentisierung und zum Schlüsseltausch werden Asymmetrische Verfahren verwendet
- Die eigentlichen Nutzdaten werden aber (mit dem frisch erzeugten Session-Key) symmetrisch verschlüsselt
- Alle im Internet gängigen Krypto-Protokolle sind Hybrid-Protokolle
  - SSL/TLS
  - S/MIME
  - OpenPGP
  - IPsec

# Zertifikate und Public Key Infrastrukturen

- Problem: Identifizierung unbekannter Personen (Firmen, Webservern ...)
- Lösung: Überprüfung der Identität aller durch eine zentrale Instanz – *Trust Center, Certificate Authority (CA)*
- CA stellt *Zertifikate* aus, bestehend aus drei Teilen:
  - Identifikations-String (Name der Person/Firma ...)
  - einem dazu gehörenden Public-Key
  - einer digitalen Signatur der obigen Daten mit dem private key der CA
- mit ihrer Unterschrift bindet die CA den Identifikations-String an den zugehörigen public key
  - Ein Client benötigt natürlich den public key der CA um die Korrektheit der Unterschrift prüfen zu können
  - Diese CA-Root keys sind in allen Browsern 'ab-Werk' enthalten
- Der Client kann also, ohne direkten Kontakt zur CA aufnehmen zu müssen, die Identität des Gegenüber verifizieren

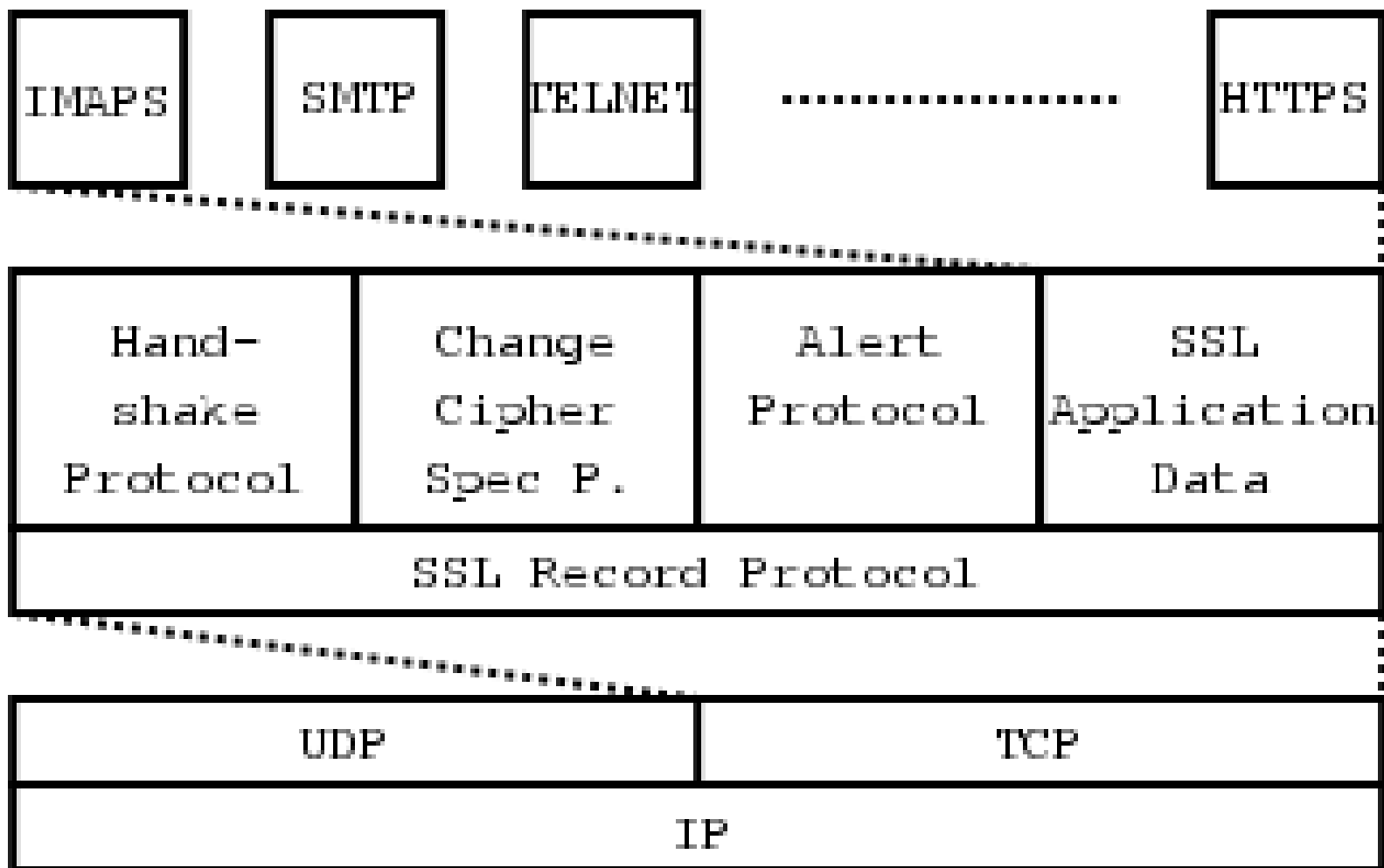
# Probleme von Public Key Infrastrukturen

- Alle wichtigen Daten liegen bei der CA. Ist die CA kompromittiert, ist die Sicherheit der gesamten PKI dahin
- Der Client ist von der Sorgfältigkeit der CA bei der Erteilung der Zertifikate abhängig
- CAs wollen auch Geld verdienen, also machen sie es den Kunden leicht, Zertifikate zu erhalten ('Funktionierende E-Mail Adresse genügt' usw.)
- Was passiert, wenn ein böser Hacker mein Webserver-Zertifikat klaut?
  - das Zertifikat wird zurückgezogen
  - das merkt der Client natürlich nur, wenn er explizit bei der CA nachfragt
  - das tut (im Moment) kein Client

# Überblick

- Verschlüsselung und Netzverkehr
- SSL Übersicht
- Crash-Kurs Kryptographie
- **SSL Record Protokoll**
- SSL Handshake Protokoll
- SSL Alert Codes
- SSL in Action
- Links und Literatur
- Diskussion

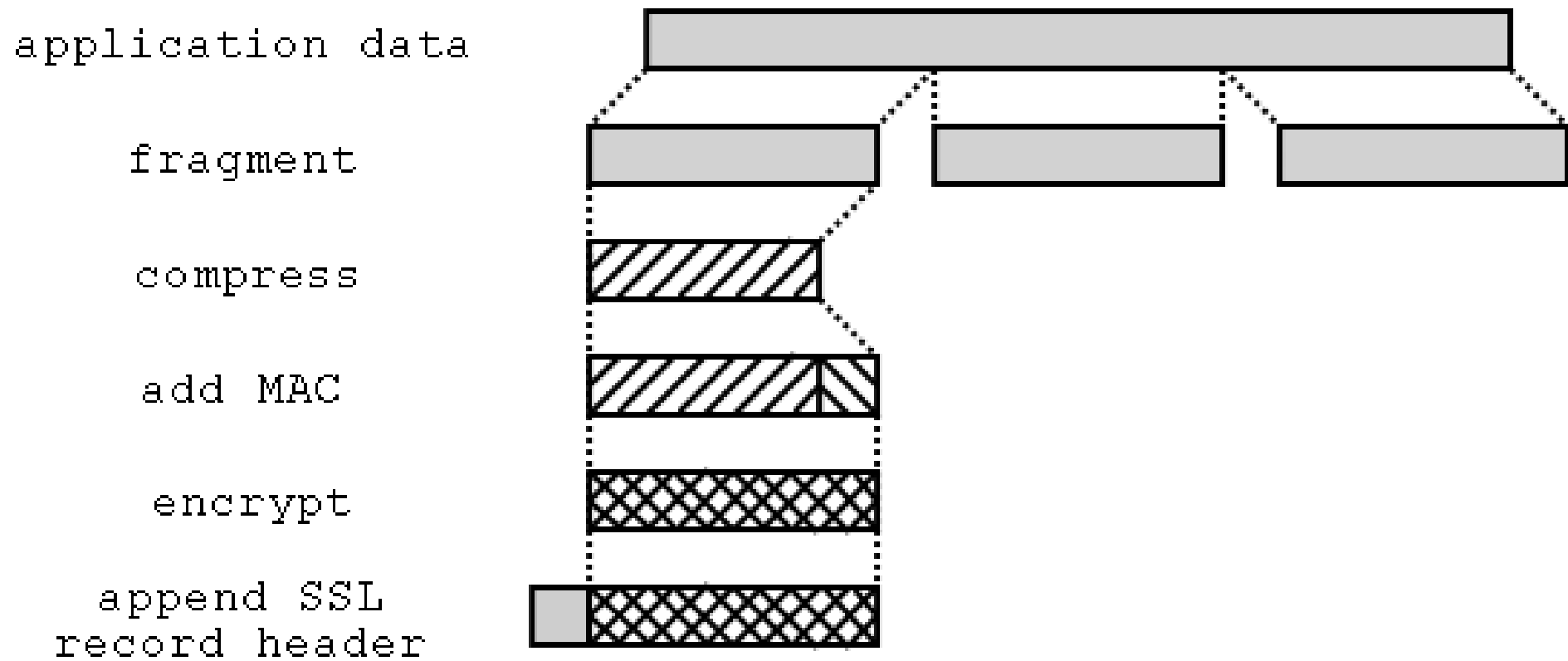
# SSL-Stack im Zusammenhang



# Aufgaben des SSL-Record Protokolls

- Fragmentierung der Applikationsdaten
- Komprimierung der Applikationsdaten (momentan nicht implementiert)
- Sicherstellung der Integrität durch 'keyed Hash'
- Ver-/Entschlüsselung
- sämtliches Key-Material (Verschlüsselung, Initialisierungs Vektor (IV) und MAC-Schlüssel) wird im Rahmen des Handshake-Protokoll erzeugt

# Aufbau des SSL-Record Protokolls



## Aufbau eines SSL-Record Paketes

Content Type	Major Version	Minor Version	Compr. Length
Plaintext ( encrypted)			
MAC ( encrypted)			

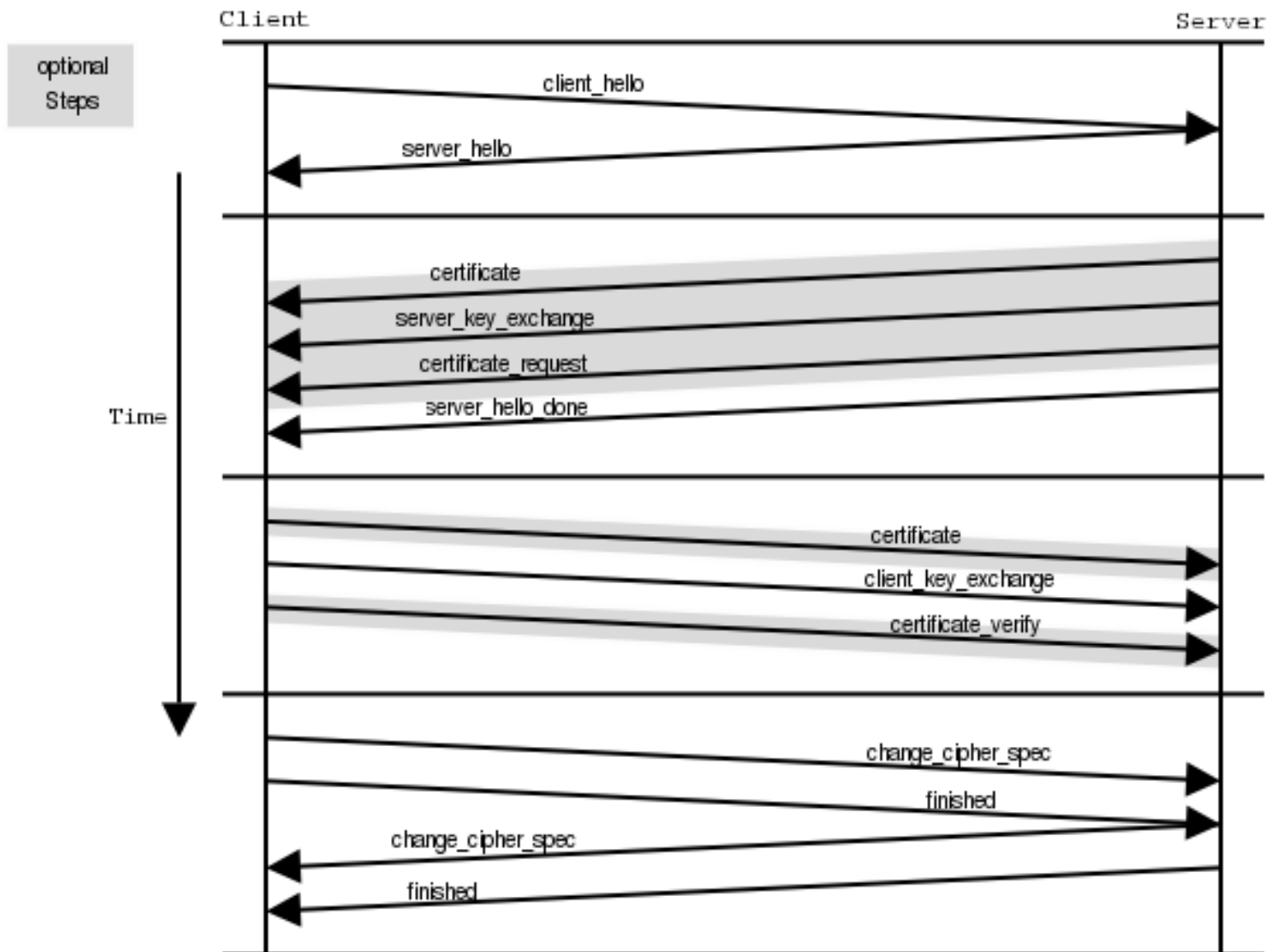
# Überblick

- Verschlüsselung und Netzverkehr
- SSL Übersicht
- Crash-Kurs Kryptographie
- SSL Record Protokoll
- **SSL Handshake Protokoll**
- SSL Alert Codes
- SSL in Action
- Links und Literatur
- Diskussion

# Aufgaben des SSL-Handshake Protokolls

- Aushandlung der zu verwendenden SSL-Protokoll-Version
- Aushandlung der zu verwendenden Krypto-Algorithmen (*cipher spec*)
- Austausch und Verifizierung der Zertifikate (*mutual authentication*)
- Aushandlung des *Master Secret* (*Key Exchange*)

# Aufbau des SSL-Handshake Protokolls



## Phase I – Client Hello

- die *Client Hello* Nachricht enthält:
- höchste vom Client verstandene SSL-Version
- Client-Random, generiert aus 32-bit UNIX Timestamp und 28-byte PRNG
- session ID (leer, falls keine etablierte SSL-Verbindung existiert)
- Liste von *cipher suites*, die der Client unterstützt

## Phase I – Server Hello

- die *Server Hello* Nachricht enthält:
- vom Client erhaltene und höchste vom Server verstandene SSL-Versionsnummer
- Server-Random, generiert aus 32-bit UNIX Timestamp und 28-byte PRNG
- vom Client erhaltene session ID (falls nicht leer)
- Auswahl einer *cipher suite* aus der vom Client präsentierten Liste
- ist die Client session ID nicht leer, so sucht der Server in seinem Cache nach den Daten der darüber referenzierten Verbindung
- Sobald der Client seine session ID empfängt, geht er direkt zur *change cipher spec* Meldung über

## Phase II – Server Authentication und Key Exchange

- Sind die Verbindungsdaten nicht gecached und müssen also neu ausgehandelt werden, so sendet der Server sein Zertifikat
- abhängig von dem Zertifikat wird entweder RSA oder DH für den Key-Exchange benutzt
- ein *server key exchange* wird nur dann versendet, wenn das Server Zertifikat weder einen RSA Public Key noch DH Parameter enthält

## Phase III – Client Authentication und Key Exchange

- Besitzt der Client ein Zertifikat, so schickt er es dem Server
- Enthält das Server Zertifikat einen RSA Public Key des Servers, so verschlüsselt der Client einen *pre-master-key* mit dem im Zertifikat enthaltenen Public Key des Servers und sendet diesen in der *client key exchange* an den Server.
- Enthält das Server Zertifikat DH Parameter, so benutzen Client und Server diese für das DH-Key-Agreement. Der so vereinbarte Schlüssel dient beiden als *pre-master-key*.
- Besitzt der Client ein Zertifikat, so schickt er danach ein explizites *certificate verify* an den Server

## Phase IV – Finish

- Hat der Client erfolgreich seinen Key generiert, so teilt er das dem Server durch Versenden einer *change cipher spec* Nachricht mit.
- Strenggenommen ist das nicht mehr Teil des Handshake Protokolls, sondern des *change cipher spec* Protokolls.
- Direkt danach sendet der Client ein *finished*. Diese Nachricht ist bereits mit den neu ausgehandelten Parametern/Keys verschlüsselt.
- Der Server antwortet ebenfalls mit *change cipher spec* und *finished*.

## Kreierung der Keys

- RSA: der Client erzeugt das *pre master secret* und übergibt es verschlüsselt an den Client
- DH: beide vereinbaren via DH Key-Agreement ihr *pre master secret*
- Das *master secret* wird als gemischter MD5/SHA1 Hash aus der Konkatination von *pre master secret*, *Client.Hello random* und *Server.Hello random* berechnet
- Die MAC-Keys, Server und Client Keys sowie die Server und Client Initialisierungs Vektoren werden als gemischter MD5/SHA1 Hash aus der Konkatination von *master secret*, *Client.Hello random* und *Server.Hello random* berechnet

# Überblick

- Verschlüsselung und Netzverkehr
- SSL Übersicht
- Crash-Kurs Kryptographie
- SSL Record Protokoll
- SSL Handshake Protokoll
- **SSL Alert Codes**
- SSL in Action
- Links und Literatur
- Diskussion

# SSL Alert Codes

- decryption\_failed
- record\_overflow
- unknow\_ca
- access\_denied
- decode\_error
- export\_restricion
- protocol\_version
- insufficient\_security
- internal\_error
- decrypt\_error
- user\_canceled
- no\_renegotiation

# Überblick

- Verschlüsselung und Netzverkehr
- SSL Übersicht
- Crash-Kurs Kryptographie
- SSL Record Protokoll
- SSL Handshake Protokoll
- SSL Alert Codes
- **SSL in Action**
- Links und Literatur
- Diskussion

# SSL in action

- Spaß am Gerät ...

# Überblick

- Verschlüsselung und Netzverkehr
- SSL Übersicht
- Crash-Kurs Kryptographie
- SSL Record Protokoll
- SSL Handshake Protokoll
- SSL Alert Codes
- SSL in Action
- **Links und Literatur**
- Diskussion

## Links und Literatur – RFC's

- RFC 2246 – The TLS Protocol Version 1.0
- RFC 2104 – HMAC: Keyed-Hashing for Message Authentication
- RFC 1319 – The MD2 Message Digest Algorithm
- RFC 1321 – The MD5 Message Digest Algorithm
- RFC 2268 – A Description of the RC2(r) Encryption Algorithm
- RFC 2459 – Internet Public Key Infrastructure: Part I: X.509 Certificate and CRL Profile

## Links und Literatur – Bücher und Papers

- „The Code Book” *Simon Singh*, Anchor Books, 2000
- „Cryptography and Network Security ” *William Stallings*, Prentice Hall, Aug. 1999, sec. Ed.
- „Security Technologies for the World Wide Web” *Rolf Oppliger*, Artech House, 2000
- „Factorization and Primality Testing” *David M. Bressoud*, Springer UTM, 1988
- „Applied Cryptography” *Bruce Schneier*, John Wiley and Sons, 1996
- „The Ten Risks of PKI” *B. Schneier/C. Ellison*  
<http://www.counterpane.com/pki-risks.html>
- „Analysis of the SSL 3.0 Protocol” *D. Wagner, B. Schneier*  
<http://www.counterpane.com/ssl.html>

## Links und Literatur – sonstige Links

- „CERT Advisory CA-2001-04 Unauthentic 'Microsoft Corporation' Certificates”

<http://www.cert.org/advisories/CA-2001-04.html>

# Überblick

- Verschlüsselung und Netzverkehr
- SSL Übersicht
- Crash-Kurs Kryptographie
- SSL Record Protokoll
- SSL Handshake Protokoll
- SSL Alert Codes
- SSL in Action
- Links und Literatur
- **Diskussion**