

SSL – Secure Socket Layer Algorithmen und Anwendung

Präsentation vom 03.06.2002

Stefan Pfab

Überblick

- **Motivation**
- **SSL-Architektur**
- **Verbindungsaufbau**
- **Zertifikate, Certification Authorities**
- **Angriffsmöglichkeiten**
- **SSL aus Entwicklersicht**

Motivation

- **Wie lässt sich die Vertrauenswürdigkeit der Gegenstelle gewährleisten?**
- **Wisst ihr alle genau, durch welche Rechner eure TCP-Pakete gehen?**

Routenverfolgung zu `www.consors.de` [62.128.11.222] über maximal 30 Abschnitte:

```
1 192.168.0.99
2 nbg2-d4-2.mcbone.net [62.104.219.39]
3 G3-0.nbg2-gsr.mcbone.net [62.104.219.5]
4 L0.ffm4-gsr.mcbone.net [62.104.191.128]
5 de-cix.regio.net [80.81.192.94]
6 bbone-gw.ffm0.regio.net [192.135.7.5]
7 noris-gw.cust.regio.net [192.135.7.9]
8 wwwffm.consors.de [62.128.11.222]
```

Ziele für SSL:

- **Verschlüsselung des Datenverkehrs**
 - Kein Mitlesen möglich
- **Authentifizierung der Gegenstelle**
 - Keine Spoofings möglich
- **Verifikation der Daten**
 - Keine Änderung an übermittelten Daten durch Dritte während der Übertragung

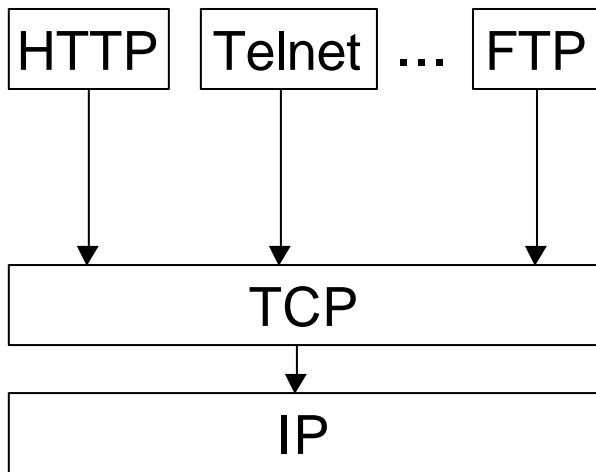
Entwicklung von SSL

- **SSL V1.0 (1993)**
 - Entwicklung von Netscape
- **SSL V2.0 (1994)**
 - Mit Netscape Navigator veröffentlicht
 - Nur 40 Bit maximale Schlüssellänge
- **SSL V3.0 (1995)**
 - Bugfixes und aktuellere Kryptographie
 - Aufhebung der Begrenzung der Schlüssellänge

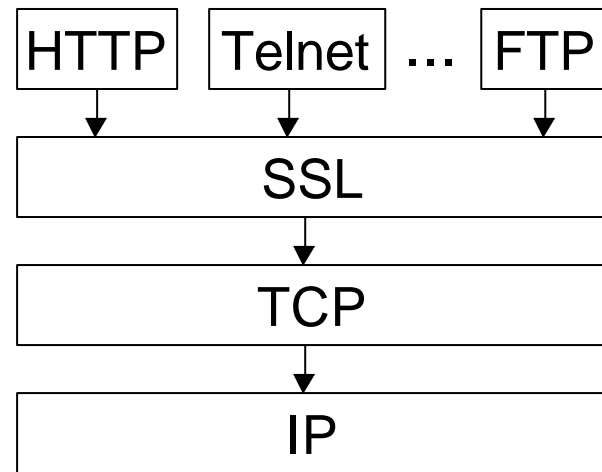
Grundlagen (1)

- **ISO/OSI-Layer mit SSL**

Standard ISO



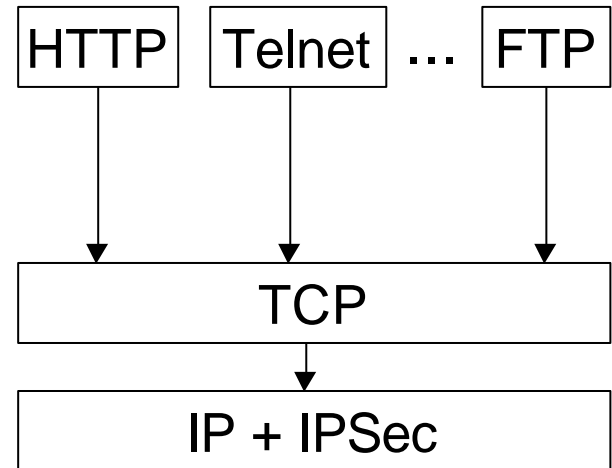
Standard ISO mit SSL



Grundlagen (2)

■ SSL vs. IPSec

- IPSec ist im Protokollstack des Betriebssystems verankert
- Bedingt keine Änderung an Programmen; bei SSL z.B. Verwendung von `ssl_write()` statt `write()` nötig



Wiederholung: Kryptographie

- **Symmetrische Verfahren**
 - Performant
 - Aber: Problem des Schlüsselaustausches
- **Asymmetrische Verfahren**
 - Bieten hohe Sicherheit, da kein gesicherter Schlüsseltausch nötig
 - Höherer Rechenzeitbedarf für Ver- und Entschlüsselung als symmetrische Verfahren
- **SSL kombiniert die Vorteile beider Verfahren**

Verbindungsaufbau

Client

Server

Client_Hello



- Verfügbare sym. Verschlüsselungen
- Client-Zufallszahl

Verbindungsaufbau

Client

Server

Client_Hello

- Verfügbare sym. Verschlüsselungen
- Client-Zufallszahl

- Verschlüsselung
- Server-Zufallszahl

Server_Hello

Verbindungsaufbau

Client

Server

Client_Hello

- Verfügbare sym. Verschlüsselungen
- Client-Zufallszahl

- Verschlüsselung
- Server-Zufallszahl

Server_Hello

- Server-Zertifikat mit Public Key

Server_Certifikate

Verbindungsaufbau

Client

Server

Client_Hello

- Verfügbare sym. Verschlüsselungen
- Client-Zufallszahl

- Verschlüsselung
- Server-Zufallszahl

Server_Hello

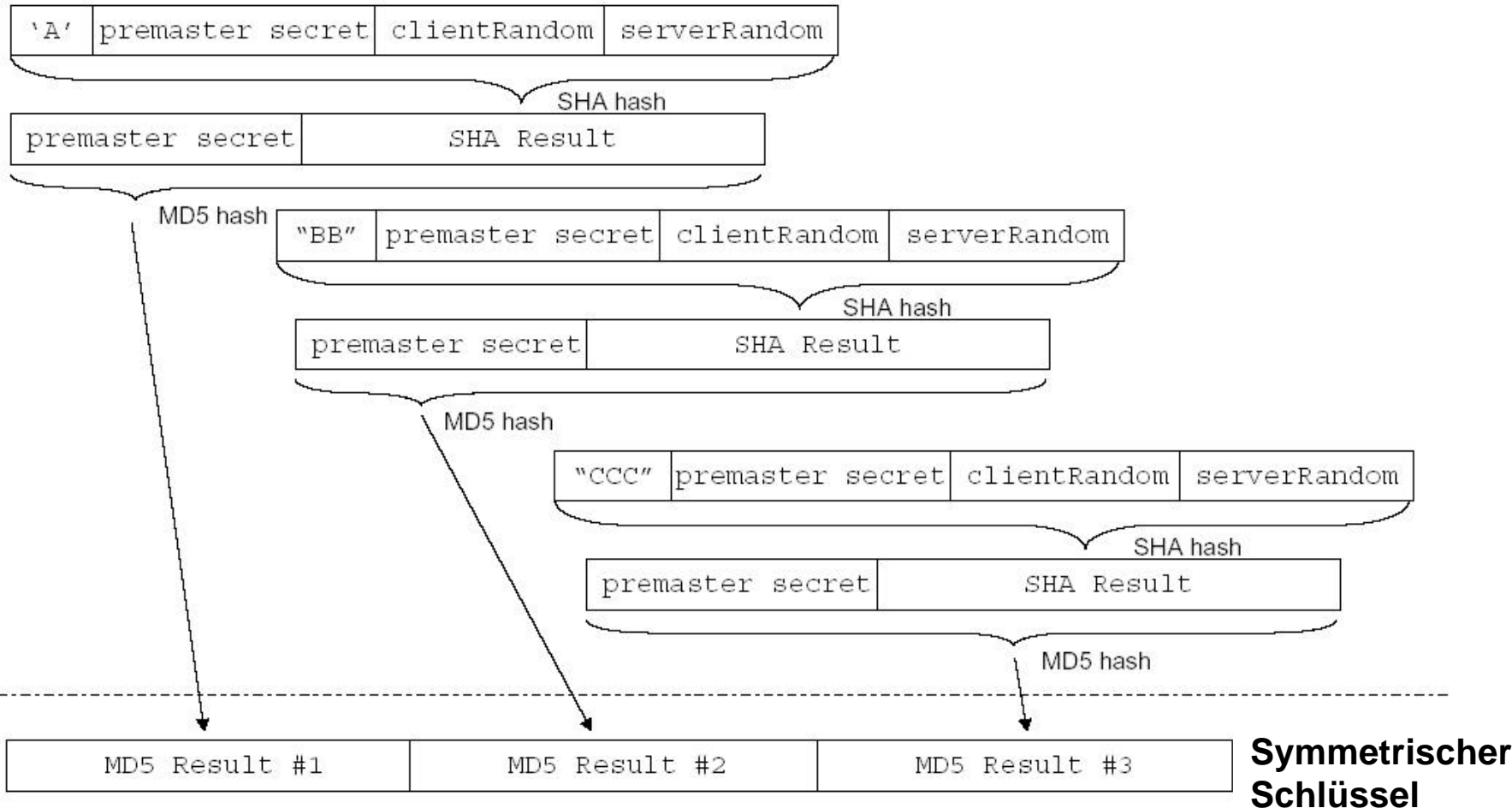
- Server-Zertifikat mit Public Key

Server_Certifikate

ClientKeyExchange

Symmetrischer Schlüssel , verschlüsselt mit Public Key

Einschub: Schlüsselerzeugung



Verbindungsaufbau

Client

Server

ClientKeyExchange



Symmetrischer Schlüssel, verschlüsselt mit Public Key

Change_Cipher



Beginn der symmetrischen Verschlüsselung



Verify_Key

Verify_Key



Sicherer Kanal



Sicherer Kanal

Zertifikate nach X.509-Standard

■ **Funktion**

- Sicherstellung der Authentizität der Gegenstelle durch Zertifikate
- Austausch der öffentlichen Schlüssel

■ **Austellung**

- durch Certification Authorities (CA)
- Selbst generierte Zertifikate mittels SSL-Paketen (z.B. openssl)

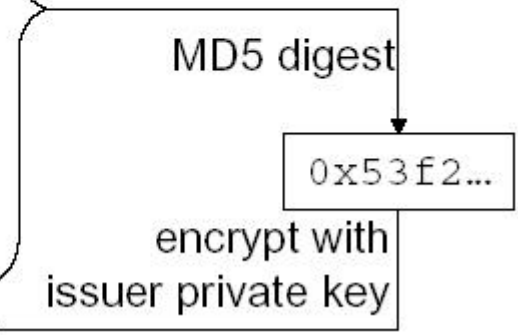
Aufbau eines Zertifikats

- **Beinhaltet folgende Felder**

- Aussteller
- Inhaber
- Gültigkeitsdauer
- Zertifizierungspfad
- Public Key
- Verwendete Algorithmen
- Mit dem PrivateKey verschlüsselter Hashwert über das Zertifikat

Beispiel-Zertifikat

Issuer	VeriSign, Inc.
Subject	Amazon, Inc. www.amazon.com
Validity Period	01-01-2000 00:00.00 UTC 01-01-2001 23:59.59 UTC
Public Key	RSA, 0x434d9384582...
	⋮
Algorithm	MD5 + RSA
Signature	0x3a53cb25445...



Certification Authorities

- **Sichere Identifizierung erfordert dritte Instanz, der beide Kommunikationspartner vertrauen**
- **Bei X.509-Zertifikaten: Certification Authority (CA)**
- **Erstellung eines eindeutigen Zertifikats:**
 1. Eine CA validiert die Identität des Zertifikatinhabers auf konventionellem Weg (Ausweis)
 2. Die CA erstellt das Zertifikat und schickt Zertifikat (per eMail) und privaten Schlüssel (per Post) an Antragsteller.

Certification Authorities (2)

- **Die CA besitzt damit auch den privaten Schlüssel**
- **Problem: hohe Verantwortung für CA (Schlüsselverlust!!)**
- **Deshalb umfangreiche Sicherheitsvorkehrungen bei CAs gegen Datenverlust**

- **Für private Nutzung: selbst generierte Zertifikate, bei denen CA und Inhaber dieselbe Person ist**

Angriffsszenarien (1)

- **Passive Traffic-Analyse**

- Länge des http-get()-Statements und der Antwort bekannt
- Scan gegen Webserver auf passende URLs und HTML-Seiten möglich

- **Beispiel: unmittelbar nach Verbindungsaufbau**

1. Client -> Server: 53 Byte übertragen
2. Server -> Client: 3742 Byte übertragen

Suche nach 53 Byte langen URLs, die 3742 Bytes Daten zurückliefern

Angriffsszenarien (2)

■ **Aktive Attacken**

- Man-in-the-middle Attacken
- Nur erfolgversprechend während des Verbindungsaufbaus, da dort die einzige unverschlüsselte Kommunikation stattfindet
- Angriffe auf verschlüsselte Daten nicht aussichtsreich (siehe Vorträge über symmetrische Verschlüsselungen)

Bewertung der Sicherheit

- **Nach heutigem Kenntnisstand sehr sicher**
- **Zwei Schwachpunkte**
 - Certification Authorities
 - Konzentration sehr vieler privater Schlüssel machen sie zu einem lohnenden Ziel
 - Benutzer
 - Unwissenheit
 - Schwache Passwörter

SSL aus Entwicklersicht

- **Beispiel: Verwendung von SSL in C auf Client-Seite**
 - Anlegen einer SSL Daten-Struktur (speichert wichtige Infos wie verwendeten Socket, PreMaster-Secret etc.)
`ssl_struct=(SSL *)SSL_new();`
 - wie üblich Socket anlegen, und `connecten(socket(), [bind()])`
`connect()`
 - Verbindung zwischen Unix-Socket und SSL herstellen
`SSL_set_fd(ssl_struct, socket);`
 - Verbindungsaufbau starten
`SSL_connect(ssl_struct);`
 - auf SSL-Socket schreiben oder lesen
`SSL_write(ssl_struct, "bye\n", 4);`

SSL aus Entwicklersicht

■ Beispiel: Verwendung von SSL in C auf Server-Seite

- `ssl_struct=(SSL *)SSL_new();`
- wie üblich Socket anlegen, auf Verbindungen warten etc.
(`socket()`, `bind()`, `listen()`, `accept()`)
- `SSL_set_fd(ssl_struct,socket);`
- Privaten Schlüssel auslesen
`SSL_use_RSAPrivateKey(ssl_struct,"server.rsa");`
- Zertifikat für aktuelle Verbindung bestimmen
`SSL_use_certificate(ssl_struct,"server.cert");`
- `SSL_connect(ssl_struct);`
- auf SSL-Socket schreiben oder lesen
`SSL_write(ssl_struct,"bye\n",4);`

Quellen

- **Eine Implementierung von SSL:**
 - <http://www.openssl.org/>
- **Die Entwickler von SSL:**
 - <http://developer.netscape.com>
- **Zwei sehr detaillierte Artikel über SSL**
 - <http://citeseer.nj.nec.com/252522.html>
 - <http://citeseer.nj.nec.com/mitchell98finitestate.html>
- **<http://www.verisign.com/products/site/secure/index.html>**
- **Eine Möglichkeit, Personen zu authentifizieren:**
 - <http://www.deutschepost.de/brief/js/index.html?/brief/produkte-services/postident/inhalt.html>
- **Programmers Reference für ssleay:**
 - <http://www2.psy.uq.edu.au/~ftp/Crypto/ssl.html>

Fragen?