

IT-Sicherheit

- Sicherheit vernetzter Systeme -

Kapitel 12: Netzsicherheit -
Schicht 4: Transport Layer
SSL / TLS

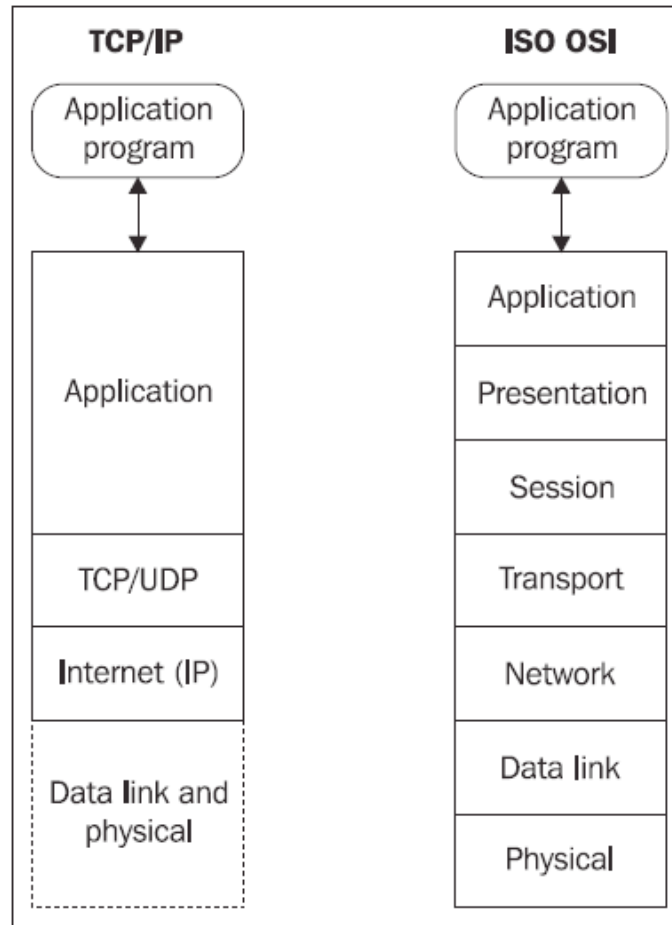
Inhalt

- Transport Layer: Funktionen
- Secure Socket Layer (SSL) & Transport Layer Security (TLS) -
Historie
- SSL / TLS Protokoll-Architektur
 - SSL / TLS Record Protocol
 - SSL / TLS Handshake Protocol
 - Schlüsselerzeugung
- SSL / TLS Anwendung

Transport Layer

- Nach OSI: Transportdienst zwischen Endsystemen; Ende-zu-Ende
 - *Zuverlässiger* Transport von
 - Nachrichten der *Endsysteme*
- In der Internet-Welt: Ende-zu-Ende-Verbindung zwischen Anwendungen
 - OSI-Schichten 5, 6 und 7 fallen in der Anwendungsschicht zusammen
 - Ports definieren die Prozesse (Dienste) der Anwendungsschicht
- Sicherungsprotokolle der Transportschicht
 - setzen auf TCP oder UDP auf
 - realisieren zum Teil die Funktionalität der Sitzungsschicht
 - liegen zwischen Transport Layer und Application Layer

Gegenüberstellung ISO/OSI- / Internet-Modell



Bildquelle: codeguru.com

Secure Socket Layer (SSL): Historie

- Ab 1994 ursprünglich entwickelt, um HTTP-Verkehr zu sichern (https); entwickelt von Netscape und ab SSL v2 in deren Browser integriert
- 1995 Internet Explorer mit PCT (Private Communication Technology)
- SSL v3: Protokollverbesserungen (aus PCT) und de-facto Standard
- Kann beliebige Anwendungen sichern (nicht nur HTTP)
- IETF entwickelt basierend auf SSL seit 1996 Transport Layer Security (TLS)
 - SSL gehört der Firma Netscape
 - TLS ist eine IETF-basierte, freie Spezifikation
 - TLS 1.0 und SSL 3.0 sind nahezu identisch
 - SSL und TLS werden häufig synonym gebraucht
 - Aktuell: TLS v1.2

SSL/TLS Einordnung

Anwendungsschicht				Anwendung
SSL Application Data Protocol	SSL Alert Protocol	SSL Change Cipher Spec Protocol	SSL Handshake Protocol	SSL
SSL Record Protocol				
Transportschicht				Netz
Netzwerkschicht				
Verbindungsschicht				

SSL / TLS Überblick

■ Authentisierung

- ❑ Vor der eigentlichen Kommunikation ist eine Authentisierung möglich
- ❑ Einseitig oder auch zweiseitig:
 - Nur Client prüft Server (z.B. HTTPS bei Online-Banking)
 - Nur Server prüft Client (eher unüblich)
 - Client und Server prüfen sich gegenseitig (z.B. Intranet-Zugang mit Client-Zertifikat)

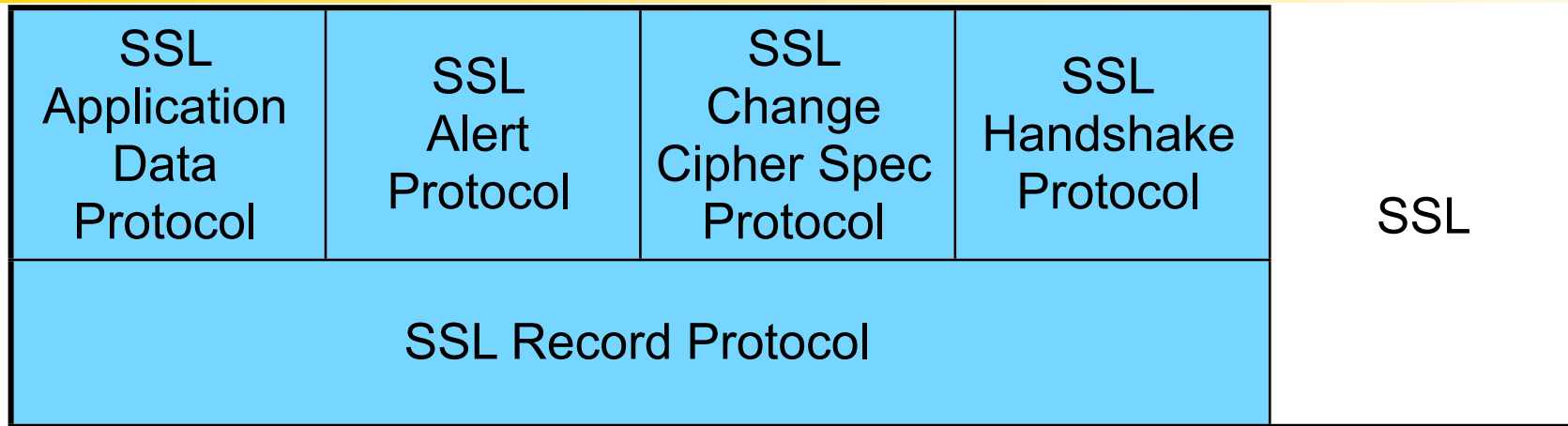
■ Vertraulichkeit der Nutzdaten

- ❑ Nur, falls während des Sitzungsaufbaus vereinbart
- ❑ Verschiedene (symmetrische!) Verschlüsselungsverfahren: RC2, RC4, DES, 3DES, DES40, IDEA, AES

■ Integrität der Nutzdaten

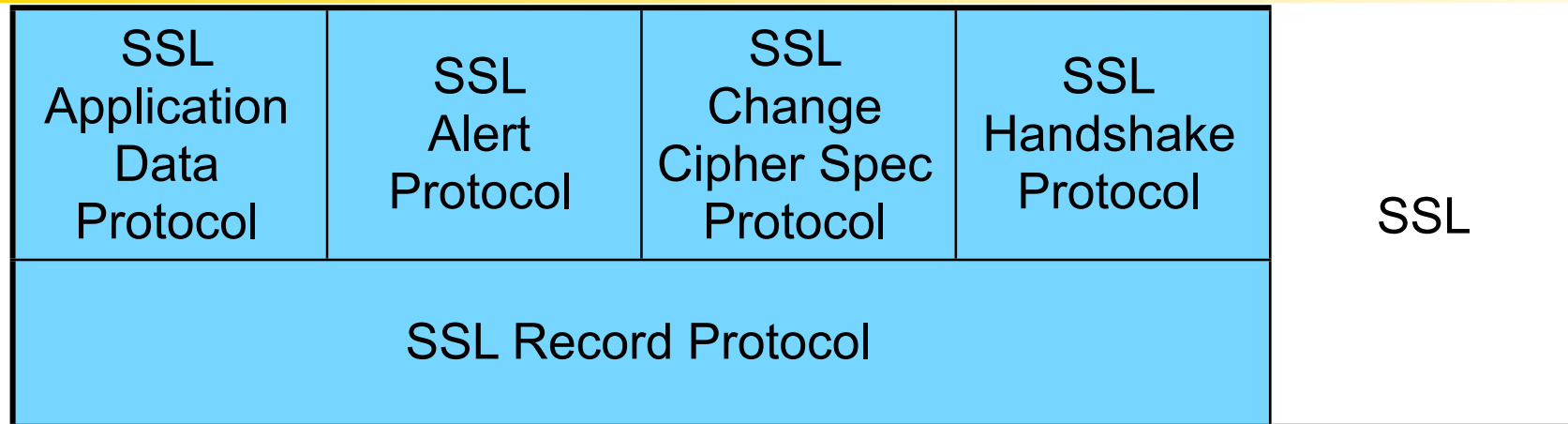
- ❑ Kryptographischer Hash-Wert, parametrisiert mit Schlüssel: HMAC
- ❑ Algorithmen: MD5, SHA

SSL/TLS Protokoll Architektur



- Application Data Protocol
 - Datenübermittlung zwischen Anwendung und SSL
 - Zugriff auf Record Protocol
- Alert Protocol:
 - Warn- und Fehlermeldungen
- Change Cipher Spec Protocol
 - Änderung der Krypto-Verfahren
 - Initialisierung und Einigung auf neu zu verwendende Verfahren

SSL/TLS Protokoll Architektur (Forts.)



■ Handshake Protocol:

- Authentisierung
- Schlüsselaustausch
- Vereinbarung der Parameter

■ Record Protocol

- Fragmentierung
- Kompression der Klartext-Daten (optional)
- Verschlüsselung (optional)
- Integritätssicherung (optional)

SSL/TLS Record Protocol

	7	15	23	31
Type	Major Version	Minor Version	Length	
Length	Data			

■ Type

- Change Cipher Spec (20)
- Alert (21)
- Handshake (22)
- Application Data (23)

■ Major und Minor Version (z.B. 3, 2 für TLS 1.1)

■ Length: Länge der Daten in Byte

SSL/TLS Record Protocol

7	15	23	31
Type	Major Version	Minor Version	Length
Length	Data		

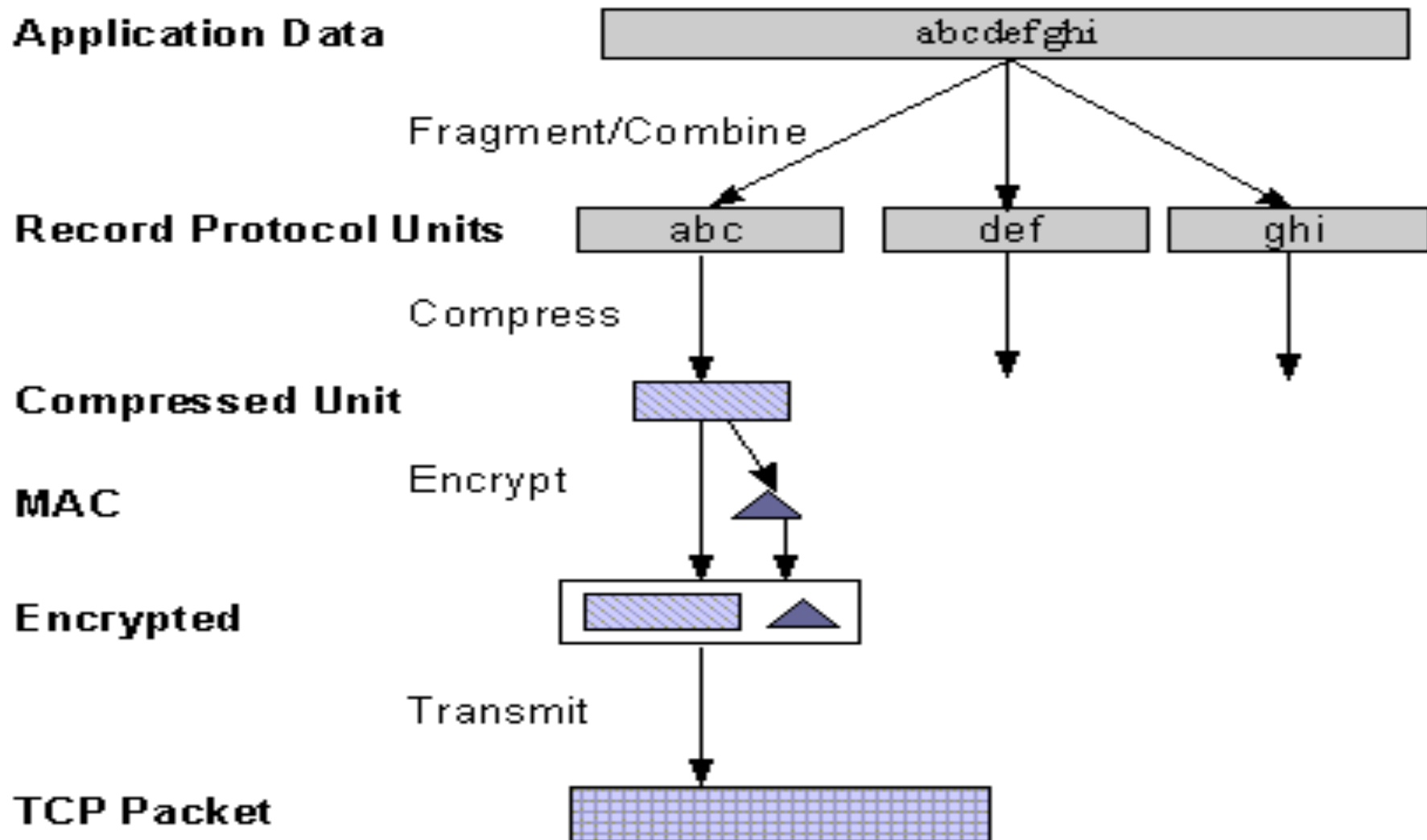
■ Sender

1. Fragmentierung der Nutzdaten in max. 2^{14} Bytes (16 kb)
2. Kompression der Daten (Default-Algorithmus *null*)
3. Integritätssicherung mittels HMAC
4. Verschlüsselung

■ Empfänger:

- Entschlüsselung; Integritäts-Check; Dekompression; Defragmentierung; Auslieferung an höhere Schicht

SSL/TLS Record Protocol - Ablauf



Bildquelle: Ralf S. Engelschall, Apache mod_ssl Dokumentation

SSL/TLS Handshake Protokoll

- Zweck: Authentisierung, Algorithmenauswahl, Schlüsselmaterial

Alice Client

Bob Server

ClientHello

enthält Liste der vom Client unterstützten Algorithmen

- Server wählt in Hello Nachricht Algorithmen
- Zertifikat zur Authentisierung
- Schlüsselmaterial (PreMaster Secret)
- Anforderung an den Client zur Authentisierung mittels Zertifikat

- ServerHello
- [ServerCertificate]
- ServerKeyExchange
- [CertificateRequest]
- ServerHelloDone

- [ClientCertificate]
- ClientKeyExchange
- [CertificateVerify]
- [ChangeCipherSpec]
- Finished

- [ChangeCipherSpec]
- Finished

SSL/TLS Handshake Protocol: Schlüsselerzeugung

- Schlüssel werden aus dem PreMasterSecret abgeleitet
- PreMasterSecret (variable Länge) wird erzeugt:
 - **RSA**: Zufallszahl; mit dem öffentlichen Schlüssel des Servers verschlüsselt vom Client übertragen
 - **Diffie-Hellman**: Übertragung der Diffie-Hellman Gruppe unverschlüsselt; falls nicht schon in Zertifikat enthalten; Erzeugung des PreMasterSecret über Diffie-Hellman Verfahren
- MasterSecret (immer 48 Byte) wird aus PremasterSec. erzeugt
 - $\text{MasterSecret} = \text{PRF}(\text{PreMasterSecret}, \text{„Master Secret“}, \text{ClientHello.random} + \text{ServerHello.random})$

SSL / TLS Schlüsselgenerierung

- KeyBlock = PRF (SecurityParameter.MasterSecret, „key expansion“, SecurityParameter.ServerRandom + SecurityParameter.ClientRandom)
- Der KeyBlock wird in folgende Teilblöcke zerlegt
 - client_write_MAC_secret [SecurityParameter.HashSize]
 - server_write_MAC_secret [SecurityParameter.HashSize]
 - client_write_key [SecurityParameter.KeyMaterialLength]
 - server_write_key [SecurityParameter.KeyMaterialLength]
- SSL erlaubt Schlüsselerzeugung auch ohne Authentisierung
 - In diesem Fall Man-in-the-Middle-Attack möglich und nicht erkennbar

SSL/TLS: Pseudo Random Function (PRF)

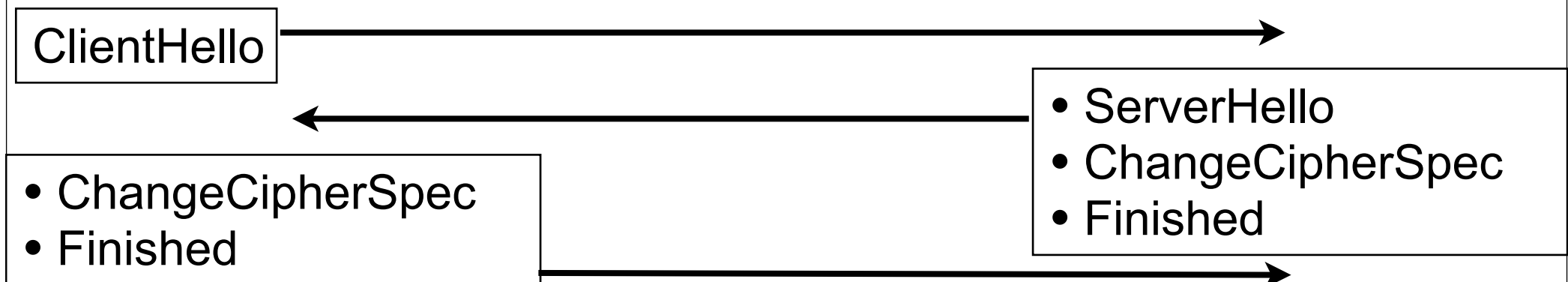
- Pseudo-Random Function (PRF); gebildet aus MD5 und SHA
- PRF soll Sicherheit bieten auch wenn MD5 oder SHA „gebrochen“ werden
- Expansionsfunktion $P_hash(secret, seed)$
 - Durch iterative Anwendung Schlüsselmaterial in beliebiger Länge
 - $P_hash(secret, seed) = \text{HMAC_hash}(secret, A(1) | seed) |$
 $\text{HMAC_hash}(secret, A(2) | seed) |$
 \dots
 $\text{HMAC_hash}(secret, A(n) | seed)$ mit
 - $A(0) = seed$
 $A(i) = \text{HMAC_hash}(secret, A(i-1));$
- $\text{PRF}(secret, label, seed) = P_MD5(S1, label + seed) \text{ XOR } P_SHA-1(S2, label + seed)$
 - mit secret zerlegt in zwei Teilstrings S1 und S2

SSL Abbreviated Handshake

- Erlaubt Wiederverwendung und Duplizierung eines bestehenden Sicherheitskontextes
- HTTP 1.0; Jedes Item einer Webseite wird über eigene TCP Verbindung übertragen

Alice Client

Bob Server



- ClientHello enthält SessionID der zu duplizierenden Session
- Falls Server SessionID bei sich findet und mit Duplizieren einverstanden ist, sendet er SessionID in ServerHello zurück

SSL Alert Protocol

- Rund 25 verschiedene Mitteilungen, z.B.
 - Beendigung der Session
 - Fehler in der Protokollsyntax (decryption failed, etc.)
 - Probleme mit der Gültigkeit von Zertifikaten

- Unterscheidet zwischen
 - Warnungen
 - Fehlern

- Fehler führen zu sofortigem Verbindungsende

SSL/TLS Anwendung

■ Auswahl an SSL gesicherten Diensten

Port	gesicherter Dienst	Protokoll
443	HTTP	https
465, 587	SMTP (Mail)	ssmtp oder smtps
585,993	IMAP	imap4-ssl
636	LDAP	ldaps
989, 990	FTP	ftps
992	Telnet	telnets
995	POP3	pop3s

SSL/TLS und HTTP

- HTTPS ist aufgrund der weiten Verbreitung von HTTP das häufigste SSL-Einsatzgebiet
- Problem bei virtuellen Webservern (z.B. Apache VHost):
 - Pro IP-Adresse nur 1 Zertifikat möglich
 - Zum Zeitpunkt des SSL-Handshakes liegen noch keine im HTTP-Request enthaltenen Angaben zum gewünschten VHost vor
- Gelöst mit TLS v1.2: Server Name Indication übermittelt gewünschten Servernamen bereits beim Verbindungsaufbau
- Aber: Verzögerung zwischen Spezifikation (RFC 5246), Implementierung (z.B. OpenSSL) und Rollout (z.B. Integration in Linux-Distributionen)

Problematische Aspekte des SSL-Einsatzes

■ Performance

- Verbindungsaufbau ist rechenintensiv und damit langsamer
- Hohe Belastung für Server mit vielen Clients
- (Symmetrische) Verschlüsselung benötigt nur wenig Rechenzeit
 - Aber oft Verzicht auf Kompression (Entropiereduzierung!)

■ Keine Ende-zu-Ende-Verschlüsselung

- Bei Kommunikation über mehr als zwei Stationen erhält jede Zwischenstation den Klartext

■ Usability in der Praxis suboptimal

- Beispiel HTTPS-Zertifikatsprüfung in aktuellen Browsern: Häufig eher als lästig statt als hilfreich empfunden

■ Immer wieder Implementierungsfehler in SSL-APIs

Einschub: OpenSSL Security Alerts

- Liste analysierter / behobener Sicherheitslücken:
<http://www.openssl.org/news/vulnerabilities.html>

- Vulnerabilities pro Jahr:
2010: 9, 2009: 8, 2008: 2, 2006: 5

- Zum Teil gravierende Auswirkungen:
 - Buffer Overflows
 - Ungültige Zertifikate werden für gültig gehalten
 - Crashes
 - Memory Leaks

- Aber: Updates/Patches i.d.R. sehr zeitnah verfügbar

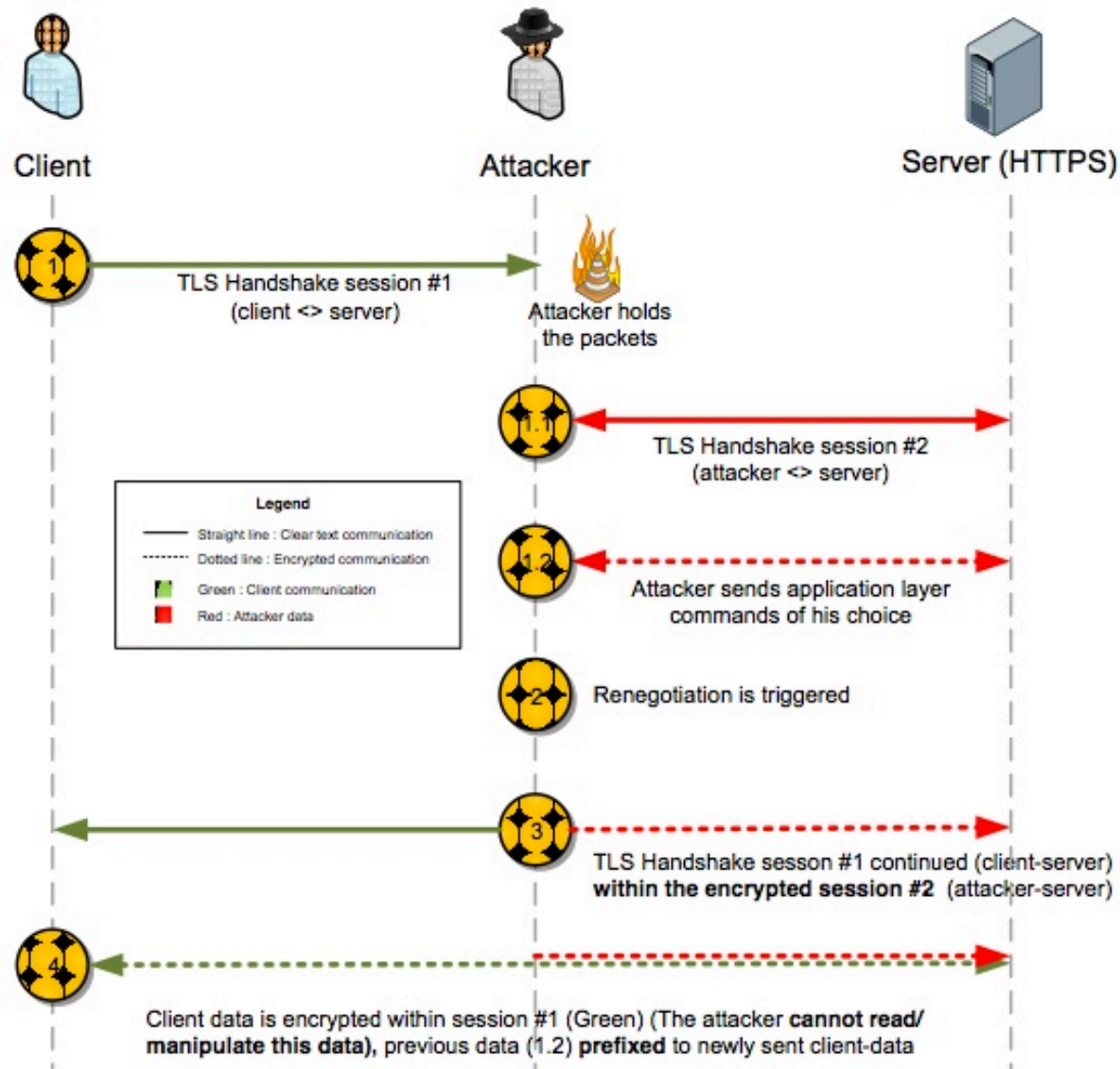
SSL-Angriffsmöglichkeiten

- Überwiegend implementierungsspezifische Angriffe
- Andere Varianten nur bedingt erfolgsversprechend:
 - Brute Force: Gesamten Schlüsselraum durchsuchen
 - Known Plaintext Attack: Viele Nachrichtenteile sind vorhersagbar, z.B. HTTP GET-Befehle (hat nur bei schwacher Verschlüsselung Aussicht auf Erfolg)
- Fehler im Protokoll-Konzept
 - Z.B. TLS & SSLv3 renegotiation vulnerability (November 2009)

TLS & SSLv3 renegotiation vulnerability

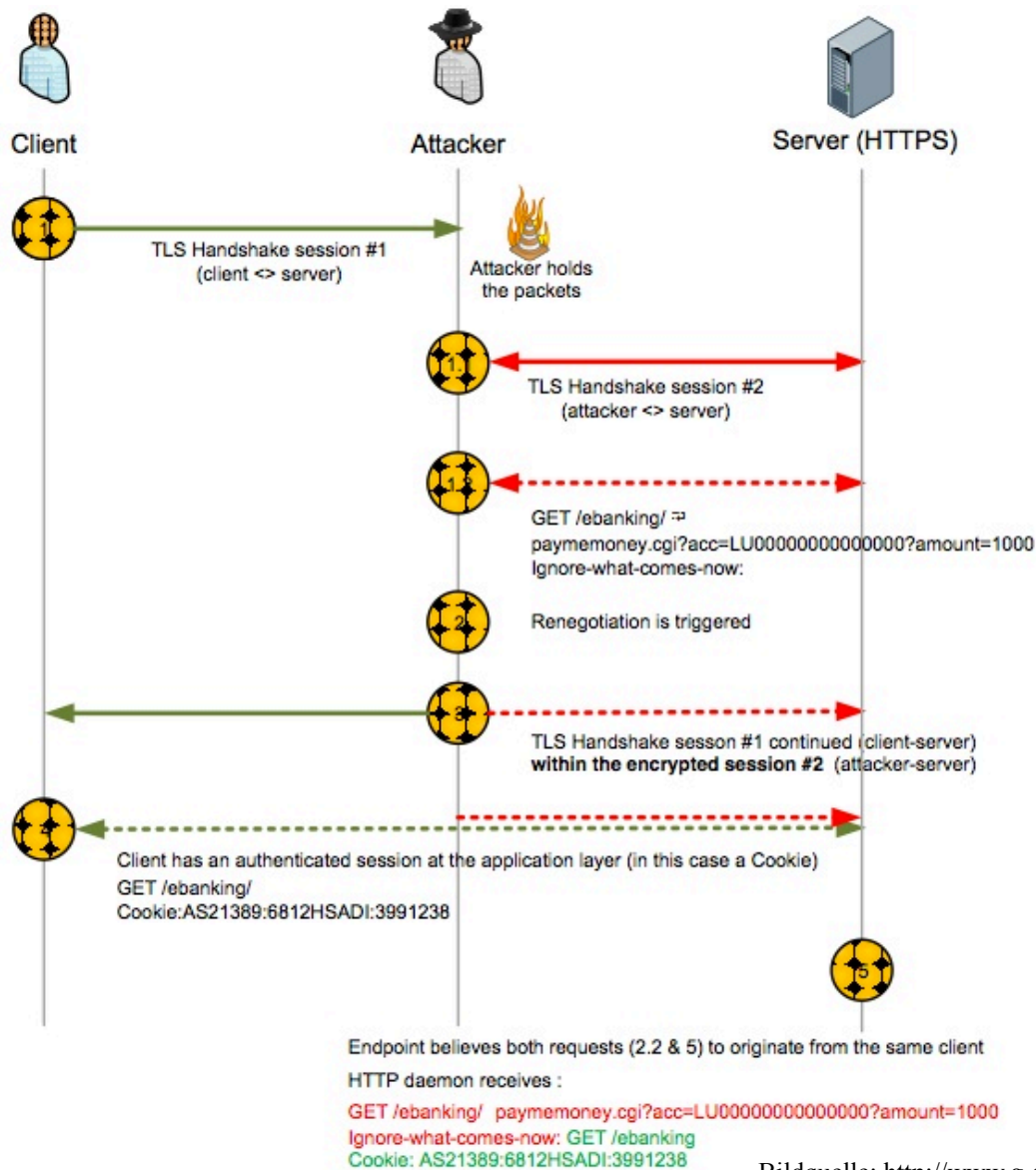
- Typische Ausgangssituation z.B. bei HTTPS:
 - Server kann erst nach Analyse der Client-Anfrage entscheiden, ob
 - Client-Zertifikat benötigt wird und/oder
 - stärkere kryptographische Verfahren eingesetzt werden sollen
 - Zu diesem Zeitpunkt besteht schon eine SSL-/TLS-gesicherte Verbindung ohne Client-Zertifikat bzw. mit zu schwachen Verfahren
- Server muss SSL-/TLS-Renegotiation anstoßen, um diesmal das Client-Zertifikat zu verlangen bzw. andere Algorithmen zu wählen.
- Nach erfolgreicher Renegotiation wird die ursprüngliche - nicht ausreichend abgesicherte - Client-Anfrage abgearbeitet.
- „Man in the middle“ kann dies ausnutzen, um eigene Anfragen am Anfang der Verbindung einzuschleusen...

Einschub: Renegotiation attack - Allgemeiner Ablauf



Bildquelle: <http://www.g-sec.lu/practicaltls.pdf>

Einschub: Renegotiation attack - bei HTTPS



- Eigentliche Client-Anfrage wird durch vorangestelltes „X-Ignore-what-comes-now:“ neutralisiert.
- Dem Angreifer unbekannte Autorisierungsdaten (z.B. Cookies) werden vom Client geliefert.

Einschub: Renegotiation vulnerability; Konsequenzen

- **Betrifft nicht nur HTTPS**
 - Beispiel SMTP: Man-in-the-middle gibt Absender und Empfänger vor, erhält so z.B. die gesamte E-Mail samt Steuerkommandos vom Client

- **Workaround in der Phase zwischen Disclosure und Patch:**
 - Renegotiation per Konfiguration verbieten/deaktivieren
 - Problematisch, da inkompatibel zu den Anwendungen, die sie bislang nutzen

- **Lösungsansatz der IETF: TLS Extension**
 - TLS-Sessions werden kryptographisch an Clients gebunden
 - Clients werden über Renegotiation informiert
 - In der initialen Aushandlung kann die Möglichkeit zur Renegotiation deaktiviert werden
 - Wurde in vielen Implementierungen kurzfristig umgesetzt

Anwendung von SSL: SSL-VPN

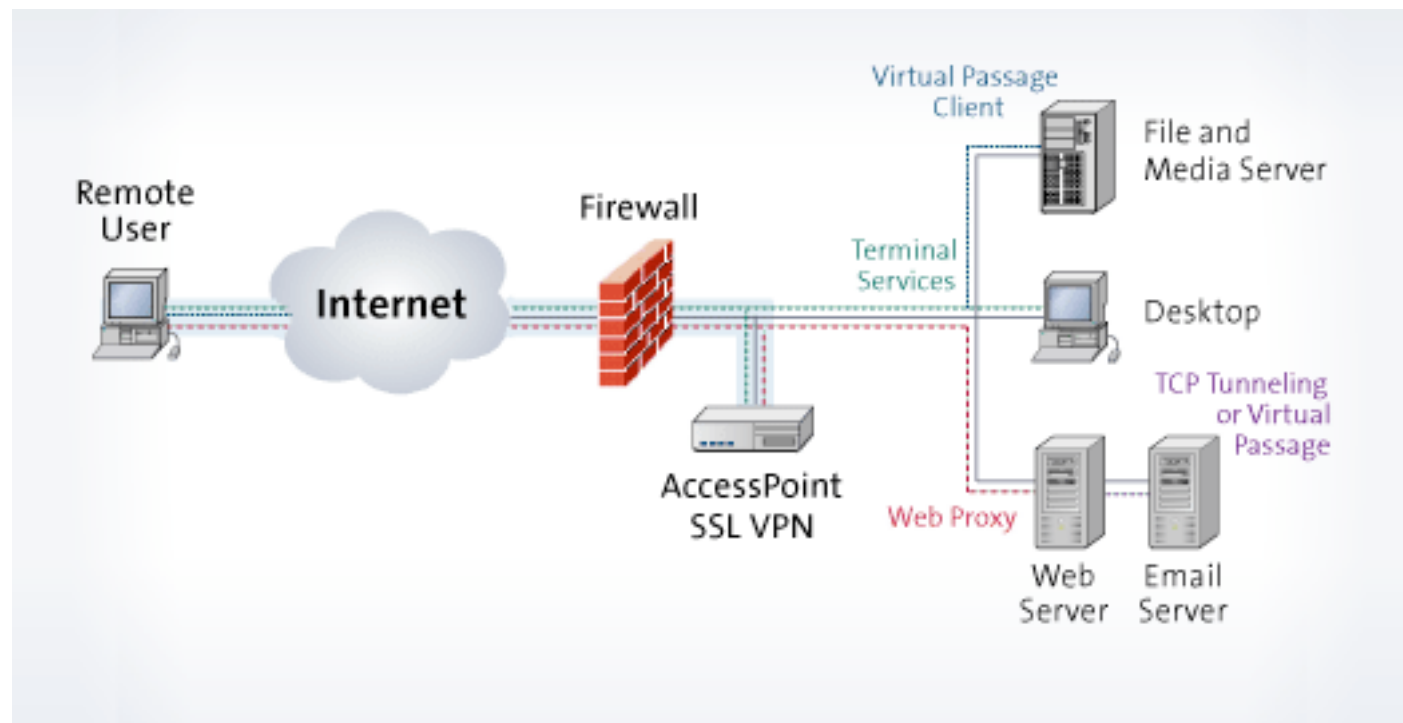
- Ziel wie z.B. bei PPTP und IPSec:
Transport geschützter Daten über öffentliche Netze

- Wiederum Site-to-Site- und End-to-End-Einsatz möglich

- Häufige Eigenschaften in der Praxis:
 - Nutzung von TCP-Port 443 (offene Firewall-Ports wg. HTTPS)
 - „Schlanker“ Client, z.T. ohne manuelle Software-Installation
 - Wichtig insb. für mobile Geräte (private Notebooks, Smartphones, ...)
 - Früher z.T. nur zur Nutzung von Webanwendungen geeignet (kein „richtiges“ VPN)
 - Start der VPN-Verbindung per Login auf einer Webseite
 - Zusätzliche Funktionalität per Browser-Plugin
 - z.B. Virens Scanner, Löschen des Cache bei Sitzungsende, ...

SSL-VPNs: Erste Generation

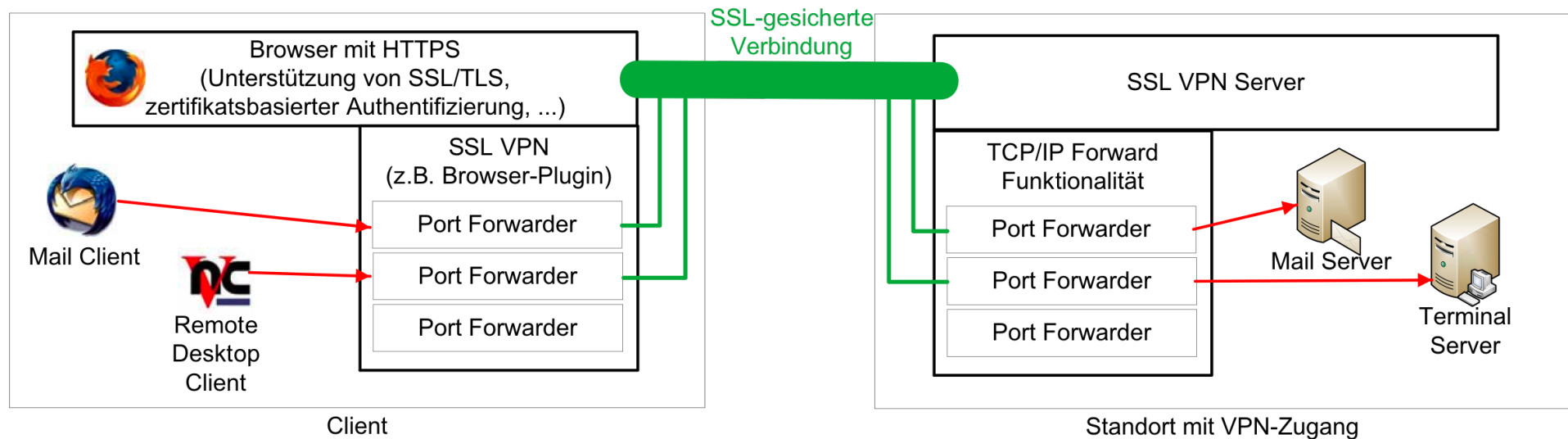
- Browserbasiert, einfache Inbetriebnahme
- Web Sessions für transparenten Client-IP-Adresswechsel (Roaming)



Bildquelle: www.menlologic.com

SSL-VPNs: Zweite Generation

- Web-basierte Installation eines „fat“ Clients
- Tunnel für TCP/IP-basierte Protokolle
- Z.T. Browser-Plugins für zusätzliche Funktionalität



SSL-VPNs: Aktuelle Trends

- IPsec zur Standort-Vernetzung
 - ausgereifte Produkte (Appliances)
 - zertifizierte, standardbasierte Lösungen (Compliance)
- SSL-VPN für mobile Anwender
 - schlanker Client
 - unkomplizierte Installation durch Anwender
- Beispiel Cisco AnyConnect SSL VPN Client
(<https://asa-cluster.lrz.de>, nutzbar mit Campus^{LMU}-Kennung)

 WebVPN Service


Login

Please enter your username and password.

GROUP:

USERNAME:

PASSWORD:

 Cisco AnyConnect VPN Client

WebLaunch

- Platform Detection
- ActiveX
- Java Detection
- Sun Java
- Download
- Connected


Using Sun Java for Installation

Launching Cisco AnyConnect VPN Client

If the software does not start properly, [Click here](#) to end the session cleanly.

VPN Client Downloader

The installer is setting up the AnyConnect application. This could take a moment. Please wait...

 Cisco AnyConnect VPN Client

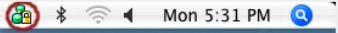
WebLaunch

- Platform Detection
- ActiveX
- Java Detection
- Sun Java
- Download
- Connected

Connection Established

The Cisco AnyConnect VPN Client has successfully connected.

The connection can be controlled from the lock icon in your menu bar.



Mon 5:31 PM

Debian OpenSSL Debacle

- OpenSSL: Freie Bibliothek zur Implementierung von SSL / TLS
- Wird genutzt von vielen Linux Derivaten, OpenSSH, Apache (mod_ssl), Onion Router (TOR), OpenVPN, etc.
- 2006: Code Analyse-Tools Valgrind und Purify erkennen potentielle Schwachstellen im Quellcode
 - Uninitialisierter Speicher
 - wird gelesen bevor er geschrieben wird
- Maintainer berichtigen diesen „Bug“ mit einem Patch
 - zwei Zeilen werden gelöscht `MD_Update(&m, buf, j)`
- Code-Abschnitt wurde verwendet, um die Entropie des Zufallszahlengenerators (PRNG) zu verbessern
- Patch bewirkt das Entropie nur noch von Prozess-ID (PID) abhängt
- In vielen Linux-Systemen 15 Bit, d.h. Entropie von 2^{15}

Debian OpenSSL Debacle Folgen

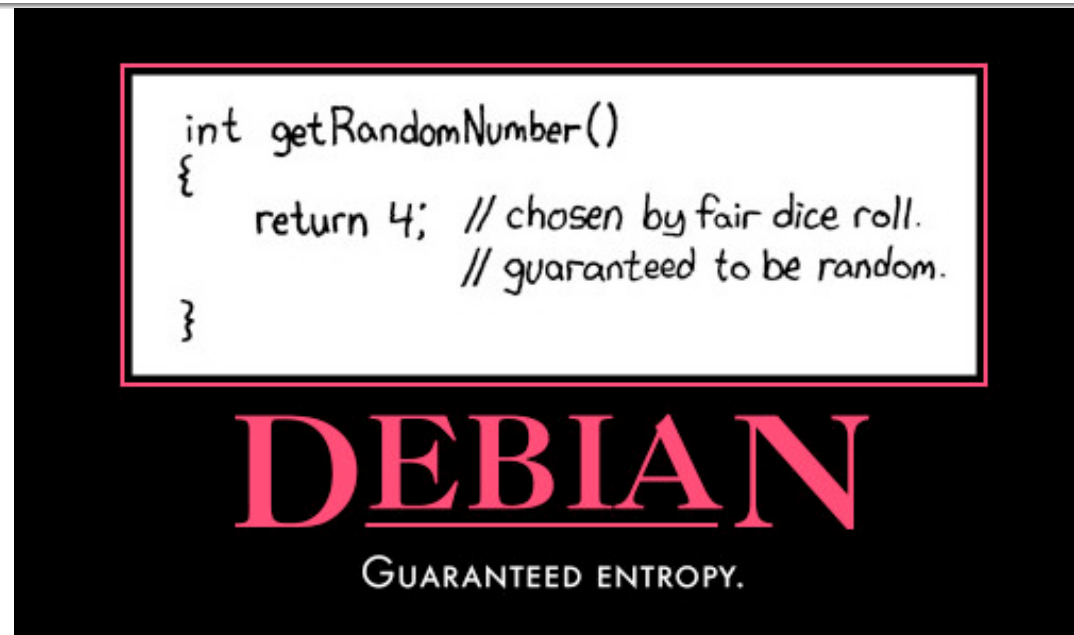
- Schlüsselraum von 32.767 Schlüsseln für jede Schlüssellänge und Schlüsseltyp
- 2008 von Luciano Belo (Argentinischer Forscher) entdeckt
- Erzeugung aller 1.024 und 2.048 Bit Schlüssel durch Moore:
 - Cluster mit 31 Xeon Cores
 - Zwei Stunden
 - Zusätzlich 6 Stunden für alle 4.096 Bit Schlüssel
- Betroffene Schlüssel:
 - SSH Host Keys
 - Benutzerschlüssel für Public Key Authentication in SSH
 - Sitzungsschlüssel
- SSH Perfect Forward Secrecy (PFS)
 - Debian Bug verhindert PFS
 - Selbst wenn Maschine sicheren Zufallszahlengenerator besitzt

Debian OpenSSL Debacle Folgen

- Schwacher Schlüssel kann einfach genutzt werden
 - `ssh -i weak-key4586 root@targetmachine`
 - Zertifikate sind einfach zu fälschen
- Damit Spoofing
 - SSL-gesicherte Web-Seiten
 - SSH
 - Bsp.: AKAMAI Server, verteilte
 - Elster (Software für Steuererklärungen)
 - Windows-Treiber von ATI für Grafikkarten
- Entschlüsselung des Verkehrs

OpenSSL Debacle: Gegenmaßnahmen

- Alle Schlüsselpaare prüfen:
 - Skript (security.debian.org/project/extra/dowkd/dowkd.pl.gz)
- Firefox oder Internet Explorer Plugin zur Überprüfung
- Wiederruf von Zertifikaten mit schwachen Schlüsseln
 - Problem: CRLs oder OCSP wird oft nicht genutzt



Quelle: <http://trailofbits.files.wordpress.com/2008/07/hope-08-openssl.pdf>