

Vorlesung Sicherheit

Dennis Hofheinz

ITI, KIT

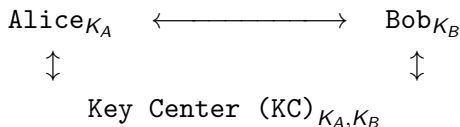
05.06.2014

- 1 Schlüsselaustauschprotokolle
 - Symmetrische Verfahren
 - Asymmetrische Verfahren
 - Transport Layer Security (TLS)

- 1 Schlüsselaustauschprotokolle
 - Symmetrische Verfahren
 - Asymmetrische Verfahren
 - Transport Layer Security (TLS)

Erinnerung

- Ziel: gemeinsamen geheimen Schlüssel K aushandeln
- Szenario: Secret-Key-Infrastruktur (mit Schlüsselzentrale):



- Vorschlag:

- 1 Alice $\xrightarrow{(Alice, Bob)}$ KC
- 2 KC $\xrightarrow{Enc(K_A, K), Enc(K_B, K)}$ Alice
- 3 Alice $\xrightarrow{Enc(K_B, K)}$ Bob

- Sehr gängig: Kerberos (im folgenden ursprüngliches Protokoll)

1 Alice $\xrightarrow{(Alice, Bob)}$ KC

2 KC $\xrightarrow{Enc(K_A, (T_{KC}, L, K, Bob)), Enc(K_B, (T_{KC}, L, K, Alice))}$ Alice

Hier T_{KC} Zeitstempel, L Gültigkeitsdauer, K frischer Schlüssel

3 Alice $\xrightarrow{Enc(K, (Alice, T_A)), Enc(K_B, (T_{KC}, L, K, Alice))}$ Bob

Hier T_A Zeitstempel

4 Bob $\xrightarrow{Enc(K, T_A+1)}$ Alice

Eigenschaften von Kerberos

- Geschachtelte Struktur verhindert Man-in-the-Middle-Angriffe
- Zeitstempel/Gültigkeitsdauern verhindern Replay-Angriffe
- **Aber:** aktiv sichere Verschlüsselung nötig!
- Sicherheit von Kerberos nicht formal geklärt

- **Wichtig:** Kerberos authentifiziert Alice und Bob auch
 - Reales Kerberos hauptsächlich zur Authentifizierung genutzt
 - Kerberos-Standard/-Implementierung historisch zur Single-Sign-On-Lösung gewachsen

(Quelle: wikipedia.org)

- **Wichtig:** Kerberos authentifiziert Alice und Bob auch
 - Reales Kerberos hauptsächlich zur Authentifizierung genutzt
 - Kerberos-Standard/-Implementierung historisch zur Single-Sign-On-Lösung gewachsen

(Quelle: wikipedia.org)

- **Wichtig:** Kerberos authentifiziert Alice und Bob auch
 - Reales Kerberos hauptsächlich zur Authentifizierung genutzt
 - Kerberos-Standard/-Implementierung historisch zur Single-Sign-On-Lösung gewachsen

(Quelle: wikipedia.org)

Eigenschaften von Kerberos

- **Wichtig:** Kerberos authentifiziert Alice und Bob auch
 - Reales Kerberos hauptsächlich zur Authentifizierung genutzt
 - Kerberos-Standard/-Implementierung historisch zur Single-Sign-On-Lösung gewachsen



(Quelle: wikipedia.org)

- 1 Schlüsselaustauschprotokolle
 - Symmetrische Verfahren
 - Asymmetrische Verfahren
 - Transport Layer Security (TLS)

- **Szenario:** Public-Key-Infrastruktur (pk_A, pk_B öffentlich)

Alice_{sk_A} \longleftrightarrow Bob_{sk_B}

- Schlüsselgenerierung von Parteien selbst vorgenommen
- Schlüsselverteilung/-zertifizierung als gegeben angenommen
- **Frage:** Wie könnte man hier Schlüssel austauschen?

- **Szenario:** Public-Key-Infrastruktur (pk_A, pk_B öffentlich)

Alice_{sk_A} \longleftrightarrow Bob_{sk_B}

- Werkzeug: Public-Key-Verschlüsselung (Gen, Enc, Dec)
- Schlüsselpaare mittels Gen generiert
- Basisverfahren überaus simpel:

1 Alice $\xrightarrow{\text{Enc}(pk_B, K)}$ Bob (mit K von Alice frisch gewählt)

Eigenschaften von Public-Key Transport

$$\text{Alice}_{sk_A} \xrightarrow{C := \text{Enc}(pk_B, K)} \text{Bob}_{sk_B}$$

- Erzielt so nur passive Sicherheit
 - Replay-Angriff möglich (wiederhole C)
 - (Ist das überhaupt ein Angriff?)
 - Nicht authentifiziert (wenn Kanal nicht authentifiziert)
- Allerdings passive Sicherheit erreicht
 - Passive Sicherheit : \Leftrightarrow nur lauschender \mathcal{A} (hier: der C erhält) kann K nicht von Zufall unterscheiden
(genaues Modell lässt \mathcal{A} mehrere Sitzungen beobachten)
 - PKE-Verfahren IND-CPA \Leftrightarrow Schlüsselaustausch passiv sicher

Eigenschaften von Public-Key Transport

$$\text{Alice}_{sk_A} \xrightarrow{C := \text{Enc}(pk_B, K)} \text{Bob}_{sk_B}$$

- Kann durch zusätzliche Signatur authentifiziert werden:

$$\text{Alice}_{\substack{sk_{\text{PKE},A}, \\ sk_{\text{Sig},A}}} \xrightarrow{(C := \text{Enc}(pk_{\text{PKE},B}, K), \sigma = \text{Sig}(sk_{\text{Sig},A}, C))} \text{Bob}_{\substack{sk_{\text{PKE},B}, \\ sk_{\text{Sig},B}}$$

- Allerdings immer noch anfällig z.B. für Replay-Angriffe

Diffie-Hellman-Schlüsselaustausch

- Basisschema sehr ähnlich zu ElGamal ($\mathbb{G} = \langle g \rangle$ gegeben)

1 Alice $\xrightarrow{g^x}$ Bob

2 Bob $\xrightarrow{g^y}$ Alice

3 Gemeinsamer Schlüssel: $K = g^{xy} = (g^y)^x = (g^x)^y$

- Diffie-Hellman (1976) lange vor ElGamal (1985), erstaunlich
- Passiv sicher unter geeigneter zahlentheoretischer Annahme
- Wird durch Signieren aller Nachrichten authentifiziert
 - Allerdings immer noch anfällig z.B. für Replay-Angriffe

- Geeignetes Modell für Schlüsselaustausch nichttrivial
 - Komplexes Szenario: viele Parteien, viele Sitzungen pro Partei
 - Angreifer passiv oder aktiv?
 - Was „darf“ ein aktiver Angreifer genau?
 - In der Realität subtile Angriffe (?) möglich (später mehr)
 - Reales Netzwerk (TCP/IP) spezifisch und komplex
 - Wie können einzelne Sitzungen unterschieden werden (SIDs)?
 - Zahlreiche verschiedene theoretische Modelle
- Deshalb hier kein abstraktes Sicherheitsmodell

- 1 Schlüsselaustauschprotokolle
 - Symmetrische Verfahren
 - Asymmetrische Verfahren
 - Transport Layer Security (TLS)

- **Problem:** http, imap, smtp, ftp, ... ungeschützt
 - Interneteinkauf, Mailabholung/-transport, Dateitransfer
- **Mögliche Lösung:** auf höherer Protokollebene sichern
 - E-Mail verschlüsseln/signieren
 - Aufpassen: *alles* verschlüsseln/signieren!
(Nur Kreditkartendaten verschlüsseln könnte Manipulation erlauben)
- **Besser:** Universelle Lösung auf Transportebene*

Transport Layer Security (TLS)

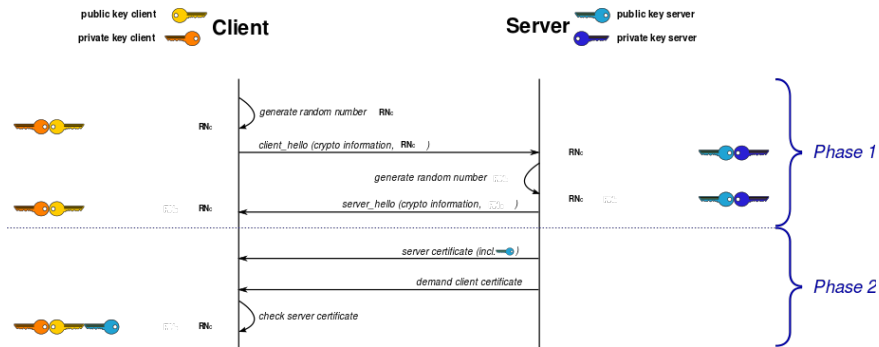
- TLS: Protokoll für den Aufbau und Betrieb sicherer Kanäle
 - Zuerst authentifizierter asymmetrischer Schlüsselaustausch...
 - ... danach symmetrische Verschlüsselung der Nutzdaten
 - Renegotiation („Neuaushandlung“ des Schlüssels) möglich
 - Viele Algorithmen möglich (RSA/DH, 3DES/AES/RC4, ...)
 - **Problem:** viele Kombinationen von Algorithmen/Varianten
- Aktuell Standard (OpenSSL/GnuTLS, https, imaps, ...)

TLS Handshake (Schlüsselaustauschprotokoll)

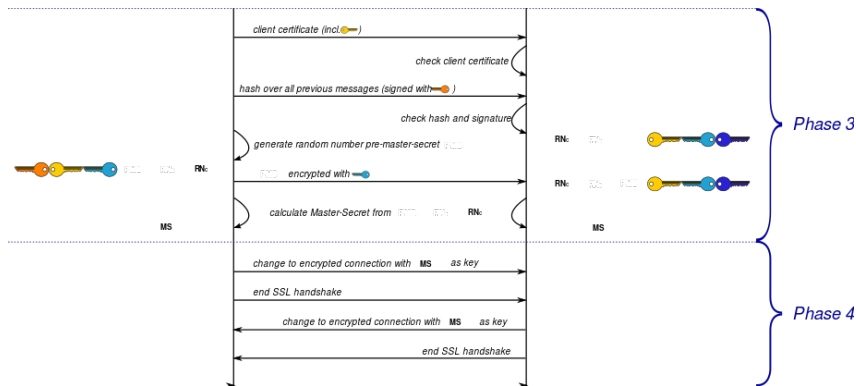


(Quelle: wikipedia.org)

TLS Handshake (Schlüsselaustauschprotokoll)



TLS Handshake (Schlüsselaustauschprotokoll)



- Vorläufer SSL von TLS:
 - SSL 1.0 (1994) von Netscape für Mosaic-Browser
 - **SSL 3.0 (1996) komplette Überarbeitung**
- TLS aus SSL hervorgegangen:
 - **TLS 1.0 (1999) \approx SSL 3.0, erstmals in Windows 98 (SE)**
 - TLS 1.1 (2006) und TLS 1.2 (2008) noch nicht gut unterstützt
($< 20\%$ aller Browser unterstützen TLS > 1.0 , aber $> 98\%$ TLS 1.0)
- Geschichtliche Einordnung (erfolgreiche Musik/Kinofilme):
 - 1994: „Hyper Hyper“ (Scooter), „Pulp Fiction“
 - 1996: „Wannabe“ (Spice Girls), „Independence Day“
 - 1999: „Baby One More Time“ (Britney Spears), „Star Wars I“
- \Rightarrow **Am häufigsten benutzte SSL/TLS-Versionen sehr alt**

Demo: Browserunterstützung

- **Demo:** <https://cc.dcsec.uni-hannover.de/>
(Welche TLS-Version unterstützt mein Browser?)

- Viele Patches (im Hinblick auf Angriffe) für konkrete Implementierungen
- Nach heutiger Kenntnis (**in neuester Version, mit Patches, mit richtigen Parametern/Algorithmen betrieben!**) sicher
- In idealen Modellen (**ideale Verschlüsselung**) sicher
 - Bedeutet, dass TLS-Verbindung „so sicher“ wie sicherer Kanal
 - **Allerdings:** Einige Angriffe (folgend) widersprechen dem nicht
- **Aber:** Angriffe auf konkrete Varianten/Kombinationen

- Einige konkrete Angriffe auf TLS(-Varianten):
 - ChangeCipherSpec Drop (1996, auf SSL < 3.0)
 - Angriff auf schwaches RSA-padding (1998, auf SSL)
 - CRIME (2002/2012, bei eingeschalteter Komprimierung)
 - TLS Renegotiation Attack (2009)
 - BEAST (2011, auf chained IVs im CBC-Mode, TLS < 1.1)
 - Angriff auf RC4 (2013, braucht viele Sessions)
 - Lucky 13 (2013, Angriff auf CBC, braucht viele Sessions)
Problem: TLS gibt aus, *warum* Dec fehlschlägt
 - (Heartbleed kein Protokoll- sondern Implementierungsproblem)

- Deshalb:
 - Aktuelle TLS-Version (1.1/1.2, mit Patches) benutzen
 - Kompression ausschalten
 - RC4 nicht benutzen (d.h. als Möglichkeit entfernen)
 - AES in nicht-CBC-Modus benutzen (z.B. AES-GCM)

Transport Layer Security (TLS)



(Quelle: wikipedia.org)

Beispielangriff: ChangeCipherSpec Drop

- **Beobachtung:** Handshake-Ablauf bei SSL <3.0:
 - 1 Client und Server tauschen unverschlüsselt Informationen aus (public keys, CipherSuite-Präferenz, Sitzungsnummer, Authentifikation)
 - 2 Client sendet ChangeCipherSpec (mit Parametern), um auf verschlüsselte Kommunikation umzuschalten
 - 3 Client und Server senden Finished und beenden Handshake
- Angenommen, Server sendet sofort danach Nutzdaten
- **Angriff:** aktiver Angreifer unterdrückt ChangeCipherSpec
- **Konsequenz:** Server sendet Nutzdaten unverschlüsselt
- **Fix:** zusätzliche Bestätigung von Client vor Nutzdaten

Beispielangriff auf RSA-Padding

- **Beobachtung:** von SSL zum Key Transport genutzte RSA-Variante berechnet

$$C = \text{Enc}(pk, \text{pad}(M)) = (\text{pad}(M))^e \bmod N$$

mit „schlechtem“ (d.h. naivem) Padding (PKCS#1.5)

- Angriff auf schlechtes Padding seit 1998 bekannt
 - Idee: verändere C homomorph, beobachte, ob verändertes C noch als gültig akzeptiert wird
 - Allein aus Gültigkeit lässt sich Information über M ableiten
 - Benötigt viele (mehrere Tausend) Versuche

Beispielangriff auf RSA-Padding (konkreter)

- Konkretes PKCS#1.5-Padding (vereinfacht):

$$C = \text{pad}(K)^e = (0x0002 || \text{rnd} || 0x00 || K)^e \bmod N$$

- K wird kurz sein (und immer mit vielen Nullbits anfangen)
- **Ziel:** finde viele gültige Faktoren α_i , so dass

$$\text{Dec}(\alpha_i^e \cdot C \bmod N) = \underbrace{\alpha_i \cdot (0x0002 || \text{rnd} || 0x00 || K)}_{=: M_i} \bmod N$$

gültig ist (so dass M_i mit 0x0002 beginnen muss)

Beispielangriff auf RSA-Padding (konkreter)

- **Ziel:** finde viele gültige Faktoren α_i , so dass

$$\text{Dec}(\alpha_i^e \cdot C \bmod N) = \underbrace{\alpha_i \cdot (0x0002 || \text{rnd} || 0x00 || K)}_{=: M_i} \bmod N$$

gültig ist (so dass M_i mit $0x0002$ beginnen muss)

- Gültigkeit wird mit Injizieren von $\alpha_i^e \cdot C \bmod N$ überprüft
- In alten SSL-Versionen sagt Server „RSA-Padding falsch“ oder „Key K zu groß“, sagt also, ob Padding gültig
- Viele gültige $M_i \Rightarrow$ grobes Intervall, in dem K liegt

Beispielangriff auf RSA-Padding (konkreter)

- Viele gültige $M_i \Rightarrow$ grobes Intervall, in dem K liegt
- Kombination geeignet vieler Intervallbestimmungen liefern K
- Bricht PKCS#1.5 (auch in TLS)
- **Gewählte Gegenmaßnahme:** patche SSL/TLS (wähle K zufällig, wenn Padding ungültig), obwohl RSA-OAEP bekannt

Beispielangriff: CRIME (Prinzip)

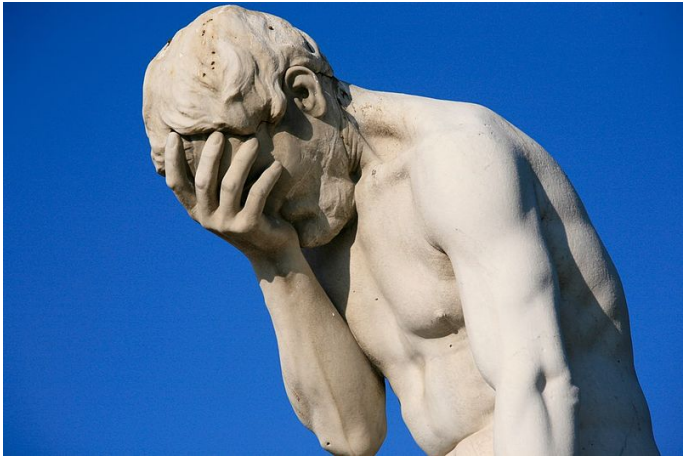
- **Annahme:** Kompression in TLS eingeschaltet
 - Übertragen wird komprimiertes $\text{comp}(M)$ statt M
 - TLS nutzt DEFLATE-Kompression (LZ77/Huffman-Variante)
 - Prinzip: $\text{comp}(\text{Fliegen fliegen}) = \text{Fliegen } f(-8,6)$
- **Strategie:** verändere *vorangehenden* Klartextteil, um zu sehen, ob *folgender* (unbekannter) Klartextteil komprimiert
- **Beispiel:** bringe Client dazu, zunächst 1234 zu senden, anschließend seine vierstellige numerische PIN **ABCD**
 - Client verschlüsselt $C := \text{Enc}(K, \text{comp}(1234\text{ABCD}))$
 - Länge $|C|$ hängt direkt ab von Länge $|\text{comp}(1234\text{ABCD})|$
 - Insbesondere: Chiffre am kürzesten, wenn **ABCD** = 1234

Beispielangriff: CRIME (konkreter)

- **Konkreter:** Angreifer möchte Cookie von Client stehlen
 - Client hat laufende Session mit Server (z.B. `www.ebay.com`)
 - Client hat *geheimes* HTTP-Cookie `ABCD`, das Browser automatisch an alle Nachrichten an Server anhängt
 - Angreifer schleust JavaScript-Code bei Client ein
 - JavaScript-Code lässt Client `WXYZABCD` an Server senden
 - Angreifer belauscht $C := \text{Enc}(K, \text{comp}(\text{WXYZABCD}))$, erfährt so Länge $|\text{comp}(\text{WXYZABCD})|$, und „wie nahe `WXYZ` an `ABCD` ist“
 - Viele Wiederholungen/`WXYZ` \Rightarrow Angreifer erfährt Cookie
- **Wichtig:** Böser JavaScript-Code kennt nur `WXYZ`, nicht `ABCD`, kann aber Client dazu bringen, `WXYZABCD` an Server zu senden

- TLS historisch gewachsen, Quasi-Standard, hochrelevant
- Viele Versionen, Einstellungsmöglichkeiten, Angriffe, Fixes
- **Frage:** warum unterstützen viele Browser nur 1999-Standard?
- **Antwort:** Angriffe aufwändig, Kompatibilität
- **Frage:** warum verwendet man nicht bessere Algorithmen?
(z.B. RSA-OAEP statt schlecht gepaddetem RSA mit Hotfix)
- **Antwort:** einfacher zu fixen als zu ersetzen

Zusammenfassung TLS



(Quelle: wikipedia.org)