Vorlesung Sicherheit

Dennis Hofheinz

ITI, KIT

16.06.2017

Überblick

Überblick

Erinnerung Schlüsselaustausch

- Symmetrisch (mit Schlüsselzentrale): Kerberos
  - Heute erweitert und zur Authentifizierung benutzt
- Asymmetrisch: Public-Key Transport, Diffie-Hellman
- Quasi-Standard für sichere Kanäle: TLS
  - TLS-Handshake: asymmetrischer Schlüsselaustausch
  - Viele Varianten, viele Optionen, technisch veraltet
  - Unbedingt gepatchte/neueste Version einsetzen!

Überblick

Beispielangriff auf RSA-Padding

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

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

  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)

- Konkreteres PKCS#1.5-Padding (vereinfacht):

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

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

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

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

Beispielangriff auf RSA-Padding (konkreter)

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

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

  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 $\mathrm{comp}(M)$ statt M
  - TLS nutzt DEFLATE-Kompression (LZ77/Huffman-Variante)
  - Prinzip: $\mathrm{comp}(\texttt{Fliegen fliegen}) = \texttt{Fliegen f(-8,6)}$
- **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 $C := \mathrm{Enc}(K, \mathrm{comp}(1234\texttt{ABCD}))$
  - Länge $|C|$ hängt direkt ab von Länge $|\mathrm{comp}(1234\texttt{ABCD})|$
  - Insbesondere: Chiffrat 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 $C := \mathrm{Enc}(K, \mathrm{comp}(\texttt{WXYZABCD}))$, erfährt so Länge $|\mathrm{comp}(\texttt{WXYZABCD})|$, 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

Zusammenfassung TLS

- TLS historisch gewachsen, Quasi-Standard, hochrelevant
- Viele Versionen, Einstellungsmöglichkeiten, Angriffe, Fixes
- **Frage:** warum unterstützen viele Browser (bis vor etwa 5 Jahren) 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

Überblick

IPsec

- Internet Protocol Security (IPsec): eigentlich gar kein KE
  - Ähnlich wie TLS, nur ohne Handshake (also *ohne* Schlüsselaustausch), und auf niedrigerer Protokollebene
  - „Sieht“ auch z.B. IP-Adressen und Portnummern
  - Unterstützt AES, 3DES, HMAC, SHA-1, MD5, ...
  - Schlüsselaustausch muss getrennt vorgenommen werden
- IPsec bei weitem nicht so populär wie TLS
- IPsec theoretisch nicht gut untersucht
- **Aber:** einige Angriffe auf spezielle Modi (CBC) existieren

Password-Authenticated Key Exchange (PAKE)

- **Ziel:** gemeinsamen geheimen Schlüssel K aushandeln

$$\texttt{Alice}_{\texttt{pw}} \quad \longleftrightarrow \quad \texttt{Bob}_{\texttt{pw}}$$

- Kommunikationskanal unsicher, aber $\texttt{pw}$ geheim
- **Problem:** vollständige Suche über alle $\texttt{pw}$ möglich
  - Angreifer kann immer $\texttt{pw}$ raten
  - **Aber:** wir wollen, dass es nicht besser geht
- **(Nicht-)Beispiel:** $\texttt{Alice}$ nutzt SKE $(\mathrm{Enc}, \mathrm{Dec})$ wählt K und...

$$\texttt{Alice}_{\texttt{pw}} \quad \xrightarrow{\mathrm{Enc}(\texttt{pw},K)} \quad \texttt{Bob}_{\texttt{pw}}$$

Frage: warum nicht optimal?

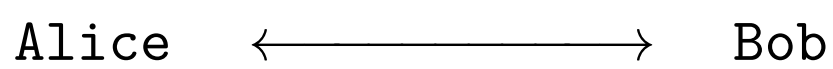
Password-Authenticated Key Exchange (PAKE)

- **Beispiele:**
  - Encrypted KE (übertrage $\mathrm{Enc}(\mathtt{pw}, pk)$, dann Check)
  - Simple Password Exponential KE (DH mit $g = \mathrm{H}(\mathtt{pw})^2$)
  - Beweisbarer PAKE (Goldreich-Lindell): nutzt Zero-Knowledge
- Grundidee: einfacher KE „mit anschließender Überprüfung“
- Anwendung z.B. bei EAP (WPA-Authentifikationsvariante)
- PAKE-Sicherheit formal modellierbar und beweisbar (unter zahlen-/komplexitätstheoretischen Annahmen)

Überblick

Zusammenfassung Schlüsselaustausch

- Ziel: gemeinsamen geheimen Schlüssel K aushandeln



- Mit Schlüsselzentrum: Kerberos
- Asymmetrisch: Key Transport, Diffie-Hellman
  (mit PKI authentifiziert)
- TLS: Standard, nur aktuelle Version verwenden
- Formale Sicherheitsuntersuchung möglich, aber komplex

Aktuelle Forschung

- Angriffe auf TLS
- Sicherheitsbeweis für TLS(-Varianten)
- Weitere KE-Varianten:
  - Non-interactive KE (NIKE)
  - ID-basierter (NI)KE
- Realistisches Sicherheitsmodell (für modulare Analyse)

Überblick

Überblick

Motivation

- Ziel: asymmetrische Authentifikation (von Parteien)

$$\texttt{Alice}_{sk_A} \quad \longleftrightarrow \quad \texttt{Bob}$$

- pk_A öffentlich
- `Alice` möchte sich bei `Bob` authentifizieren
  - `Bob` möchte sicher sein, dass er mit `Alice` redet
  - Genauer: `Bob` möchte sicher sein, dass er mit der Partei redet, die sk_A besitzt
  - Noch genauer: `Bob` möchte sicher sein, dass er mit *einer* Partei redet, die *einen* passenden secret key zu pk_A kennt
- Ab jetzt heißt `Bob` V („Verifier“) und `Alice` P („Prover“)

Einfachste Lösung

- Einfachste Lösung:

$$\mathrm{P}_{sk_A} \xrightarrow{sk_A} \mathrm{V}$$

  (Annahme: sk_A kann als „passend“ zu pk_A erkannt werden)
- V kann sich sicher sein, dass Gesprächspartner sk_A kennt
- **Aber:** dieses Protokoll scheint nicht sehr nützlich zu sein
  - Bei mehrmaliger Verwendung schwindet Garantie
  - Problem: sk_A bleibt bei Protokoll nicht geheim

Nächster Versuch

- **Neue Anforderung:** nach Protokoll...
  1. ...lernt V sk_A nicht
  2. ...ist V sicher, dass Gegenüber sk_A kennt
- Nächster Versuch: nutze Signaturschema

$$\mathrm{P}_{sk_A} \xrightarrow{\sigma := \mathrm{Sig}(sk_A, \text{„}\texttt{ich bin's, P}\text{“})} \mathrm{V}$$

(V überprüft mit $\mathrm{Ver}(pk_A, \text{„}\texttt{ich bin's, P}\text{“}, \sigma)$)

- **Analyse:** (informell)
  1. Würde V sk_A lernen, wäre Signaturschema unsicher
  2. Allerdings unklar, in welchem Sinne P sk_A kennt
- **Problem:** σ kann von Angreifer verwendet werden!

Noch ein Versuch

- **Beobachtung:** nicht-interaktive Protokolle problematisch:

$$\mathrm{P}_{sk_A} \xrightarrow{X} \mathrm{V}$$

  - X kann von Angreifer verwendet werden!
- Ein interaktiver Versuch (mit Signaturschema):

1. $\mathrm{P} \xleftarrow{R} \mathrm{V}$
2. $\mathrm{P} \xrightarrow{\sigma := \mathrm{Sig}(sk_A, R)} \mathrm{V}$

  (R Zufallszahl, V überprüft mit $\mathrm{Ver}(pk_A, R, \sigma)$)
- Passiert implizit bei TLS, mehr hierzu später

Überblick

Sicherheitsmodell

- **Frage:** was wollen wir eigentlich wirklich?

Überblick

Formalisierung PK-Identifikationsprotokoll

- (Public-Key-)Identifikationsprotokoll: $(\mathrm{Gen}, \mathrm{P}, \mathrm{V})$
- PPT-Algorithmus $\mathrm{Gen}(1^k)$ gibt Schlüsselpaar (pk, sk) aus
- Zwei PPT-Algorithmen P, V *mit Zustand* interagieren:
  1. V wird mit Eingabe pk gestartet, Ausgabe sei out_V
  2. P wird mit Eingaben sk und out_V gestartet, Ausgabe out_P
  3. V wird mit Eingabe out_P gestartet, Ausgabe out_V
     - Ist $\mathrm{out}_\mathrm{V} \in \{0, 1\}$, dann beende die Interaktion
     - Andernfalls zurück zu Schritt 2 (sk-Eingabe nicht mehr nötig)
- **Notation:** $\langle \mathrm{P}(sk), \mathrm{V}(pk) \rangle$ ist Transkript der Interaktion
- **Korrektheit:** V gibt schließlich 1 aus für $(pk, sk) \leftarrow \mathrm{Gen}(1^k)$

Sicherheit eines PK-Identifikationsprotokolls

- PK-ID-Protokoll $(\mathrm{Gen}, \mathrm{P}, \mathrm{V})$ sicher $:\Leftrightarrow$ kein PPT-Angreifer $\mathcal{A}$ gewinnt folgendes Spiel mehr als vernachlässigbar oft:
  - **Phase 1:** $\mathcal{A}$ darf mit beliebig vielen P-Instanzen (mit sk_i) in der Rolle des Verifiers V (mit Eingabe pk_i) interagieren.
    Die verwendeten $(pk_i, sk_i) \leftarrow \mathrm{Gen}(1^k)$ sind vom Spiel gewählt.
  - **Phase 2:** $\mathcal{A}$ sucht sich ein schon vom Spiel gewähltes pk_{i^*} aus und interagiert mit einer V-Instanz (mit Eingabe pk_{i^*})
  - **Entscheidung:** $\mathcal{A}$ gewinnt, wenn V schließlich 1 ausgibt
- **Intuition:** Kein $\mathcal{A}$ schafft es, andere zu impersonieren
- **Allerdings:** Verhindert keinen Man-in-the-Middle-Angriff

Überblick

Ein sicheres PK-Identifikationsprotokoll

- Erinnerung Kandidat $(\mathrm{Gen}, \mathrm{P}, \mathrm{V})$

1. $\mathrm{P} \xleftarrow{R} \mathrm{V}$
2. $\mathrm{P} \xrightarrow{\sigma := \mathrm{Sig}(sk_A, R)} \mathrm{V}$

Theorem (Sicherheit von $(\mathrm{Gen}, \mathrm{P}, \mathrm{V})$)

Ist das verwendete Signaturverfahren EUF-CMA-sicher, so ist das obige PK-Identifikationsprotokoll $(\mathrm{Gen}, \mathrm{P}, \mathrm{V})$ sicher.

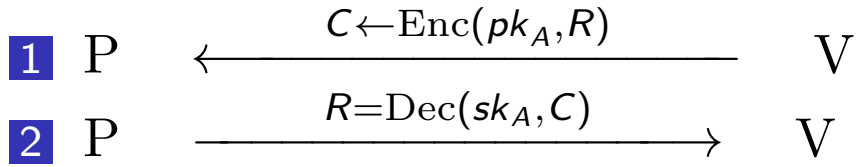
Beweisidee.

Konstruiere EUF-CMA-Angreifer $\mathcal{B}$ aus PK-ID-Angreifer $\mathcal{A}$. □

Überblick

Noch ein sicheres PK-Identifikationsprotokoll

- Ähnlicher Kandidat $(\mathrm{Gen}, \mathrm{P}, \mathrm{V})$ mit Verschlüsselung:



Theorem (Sicherheit von $(\mathrm{Gen}, \mathrm{P}, \mathrm{V})$)

Ist das verwendete Verschlüsselungsverfahren IND-CCA-sicher[1]*, so ist das obige PK-Identifikationsprotokoll* $(\mathrm{Gen}, \mathrm{P}, \mathrm{V})$ *sicher.*

- Beweisidee wie im signaturbasierten Protokoll
- **Achtung:** (pk_A, sk_A) nicht auch zur Verschlüsselung benutzen

[1] „[Ciphertext] Indistinguishability under Chosen-Ciphertext Attacks“, wie IND-CPA, modelliert aber aktive Angriffe: $\mathcal{A}$ erhält $\mathrm{Dec}(sk, \cdot)$-Orakel

Zusammenfassung Identifikationsprotokolle

- Identifikation notwendig interaktiv
- Signatursysteme (oder aktiv sichere PKE-Verfahren) hinreichend für sichere Identifikation
  - Ähnliche einfache Identifikationsprotokolle implizit z.B. in TLS verwendet
- Nicht behandelt: stärkere Sicherheitsbegriffe

  (Man-in-the-Middle-Angriffe, Reset-Angriffe, ...)

Überblick

Überblick

Motivation Zero-Knowledge

- Einige intuitive (naive?) Anforderungen an PK-ID-Protokoll noch nicht oder nur teilweise erfüllt:
  1. V lernt sk_A nicht
  2. V ist sicher, dass Gegenüber sk_A kennt
- Erinnerung Identifikation mit Signaturen:
  1. $\mathrm{P} \xleftarrow{R} \mathrm{V}$
  2. $\mathrm{P} \xrightarrow{\sigma := \mathrm{Sig}(sk_A, R)} \mathrm{V}$
- Zwar lernt V nicht den kompletten sk_A...
  ... aber vielleicht Teilinformationen über sk_A
- Vielleicht kennt P nur „Ersatz“-sk_A
- **Frage:** Für Identifikation beides nicht schlimm...
  ... können wir trotzdem intuitive Anforderungen erfüllen?

Überblick

Zero-Knowledge-Eigenschaft

- **Erste Anforderung:** V lernt sk_A nicht
- **Keine halben Sachen:** V lernt nichts über sk_A
- **Zurückrudern:** V lernt nichts über sk_A, was er nicht schon aus pk_A berechnen kann (Bsp.: $sk_A = x$, $pk_A = g^x$)
- **Anders gesagt:** Alles, was V über sk_A berechnen kann, kann er schon aus pk_A berechnen
- **Randbedingung:** Natürlich muss das auch für „bösen V“ (d.h. für Angreifer $\mathcal{A}$ in der Rolle von V) gelten

Ununterscheidbarkeit

- Hilfsformalismus: Ununterscheidbarkeit

Definition (Ununterscheidbarkeit)

Zwei (möglicherweise vom Sicherheitsparameter $k \in \mathbb{N}$ abhängige) Verteilungen X, Y sind *ununterscheidbar* (geschrieben $X \overset{c}{\approx} Y$), wenn für alle PPT-Algorithmen $\mathcal{A}$ die Differenz

$$\Pr\left[\mathcal{A}(1^k, x) = 1 \mid x \leftarrow X\right] - \Pr\left[\mathcal{A}(1^k, y) = 1 \mid y \leftarrow Y\right]$$

vernachlässigbar in k ist.

- **Intuition:** X und Y nicht (effizient) unterscheidbar

Zero-Knowledge-Eigenschaft (formal)

Definition (Zero-Knowledge)

Ein PK-Identifikationsprotokoll $(\mathrm{Gen}, \mathrm{P}, \mathrm{V})$ ist *Zero-Knowledge (ZK)*, falls für jeden PPT-Algorithmus $\mathcal{A}$ (den Angreifer) ein PPT-Algorithmus $\mathcal{S}$ (der Simulator) existiert, so dass die folgenden Verteilungen ununterscheidbar sind (wobei $(pk, sk) \leftarrow \mathrm{Gen}(1^k)$):

$$\Big(pk, \langle \mathrm{P}(sk), \mathcal{A}(1^k, pk)\rangle\Big) \qquad \text{und} \qquad \Big(pk, \mathcal{S}(1^k, pk)\Big)\,.$$

- **Intuition:** Interaktionstranskripte simulierbar
- **Bemerkung:** $\mathcal{A}$ kann ganzes Wissen in Transkript packen
- Varianten möglich (z.B. Gleichheit statt Ununterscheidbarkeit)