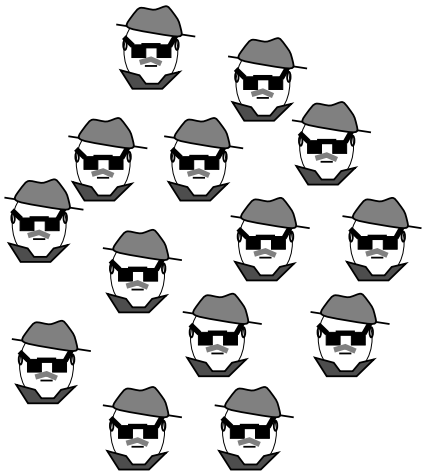
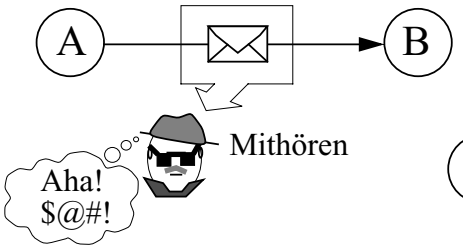
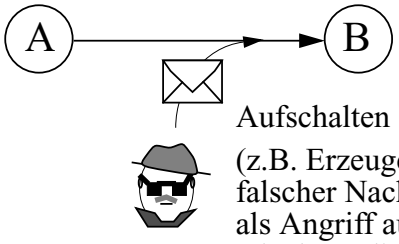
Sicherheit



Sicherheit in verteilten Systemen

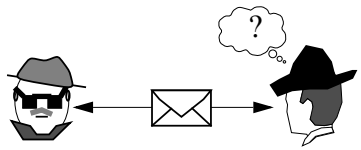


Mithören

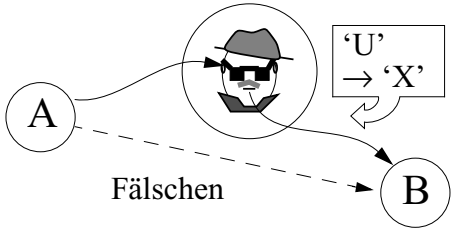


Aufschalten

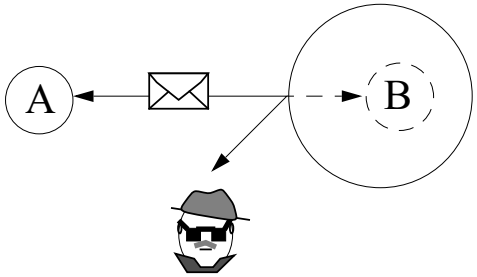
(z.B. Erzeugen falscher Nachrichten als Angriff auf die Glaubwürdigkeit)



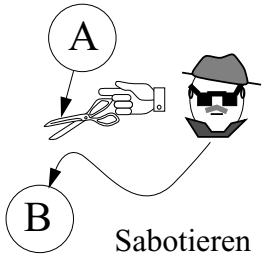
Wechselseitiges Misstrauen



Fälschen



Vorspiegeln falscher Identität



Sabotieren

Sicherheit: Anforderungen

- Autorisierung / Zugriffsschutz
  - Einschränkung der Nutzung auf den Kreis der Berechtigten
- Vertraulichkeit
  - Daten / Nachrichteninhalt gegen Lesen Unberechtigter schützten
  - Kommunikationsverhalten (wer mit wem etc.) geheim halten
- Authentizität
  - Absender "stimmt" (z.B. Server ist der, für den er sich ausgibt)
  - Daten sind "echt" und aktuell (→ Integrität)
- Integrität
  - Wahrung der Unversehrtheit von Nachrichten, Programmen und Daten
- Verfügbarkeit der wichtigsten Dienste
  - keine Zugangsbehinderung ("denial of service") durch andere
  - kein provozierter Absturz ("Sabotage")

- Weitergehende Anforderungen, z.B.:
  - Nichtabstreitbarkeit, accountability
  - strafrechtliche Verfolgbarkeit (z.B. Protokollierung; „Key Escrow")
  - Konformität zu rechtlich / politischen Vorgaben
  - ...

Sicherheit: Verteilungsaspekte

- *Offenheit* in verteilten Systemen "fördert" Angriffe
  - grosse Systeme → vielfältige Angriffspunkte
  - standardisierte Kommunikationsprotokolle → Angriff *einfach*
  - räumliche Distanz → Ortung des Angreifers schwierig, Angriff *sicher*
  - breiter Einsatz, allgemeine Verwendung → Angriff *reizvoller*
  - physische Abschottung nicht durchsetzbar
  - technologische Gegebenheiten: z.B. Wireless LAN ("broadcast")
- *Heterogenität*
  - sorgt für zusätzliche Schwachstellen
  - erschwert Durchsetzung einer einheitlichen Schutzphilosophie
- *Dezentralität*
  - fehlende netzweite Sicherheitsautorität

→ Gewährleistung der Sicherheit ist in verteilten Systemen *wichtiger* und *schwieriger* als in alleinstehenden Systemen!

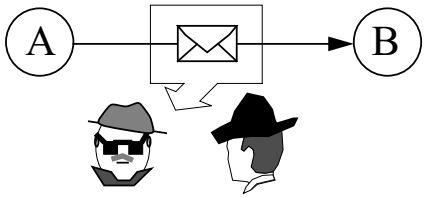
Typische Techniken und "Sicherheitsdienste":

- *Verschlüsselung*
- *Autorisierung* ("der darf das!")
- *Authentisierung* ("X ist wirklich X!")

} Hierfür Krypto-systeme und Protokolle als "Security Service", z.B. *Kerberos*

Angriffsformen

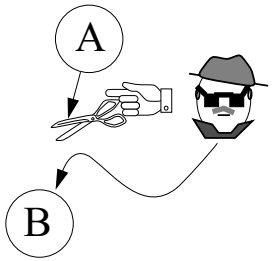
- *Passive Angriffe*: Beobachten der Kommunikation
  - Inhalt von Nachrichten in Erfahrung bringen ("eavesdropping")
  - Kommunikationsverhalten analysieren ("wer mit wem wie oft?")



→ Verschlüsselung

→ Anonymisierung

- *Aktive Angriffe*: vorsätzliche Täuschung; Eindringen
  - Durchbrechen von Zugangsschranken
  - Verändern des Nachrichtenstroms (Verändern, Vernichten, Erzeugen, Vertauschen, Verzögern, Wiederholen ("replay") von Nachrichten)
  - Vorspiegelung falscher Identitäten (Maskerade: Nachahmen anderer Prozesse oder Nutzung eines fremden Passwortes)



  - Missbräuchliche Nutzung von Diensten
  - Denial of Service durch Sabotage oder Verhindern des Dienstzugangs, z.B. auch durch Überfluten mit Nachrichten

Authentifizierung

…Seid auf eurer Hut vor dem Wolf; wenn er hereinkommt, so frisst er euch alle mit Haut und Haar. Der Bösewicht verstellt sich oft, aber an seiner rauhen Stimme und seinen schwarzen Füssen werdet ihr ihn gleich erkennen. …

(„Der Wolf und die sieben Geisslein" aus den Märchen der Gebrüder Grimm)

- *Authentizität* ist essentiell für die Sicherheit eines verteilten Systems
  - zu authentischen Nachrichten / Daten vgl. auch den Begriff "Integrität"
- Authentizität eines *Subjekts (Client)*
  - ist er wirklich der, der er vorgibt zu sein?
  - darf ich als Server daher ihm (?) den Zugriff gewähren?
- Authentizität eines *Dienstes (Server)*
  - Bsp.: Handelt es sich wirklich um den Druckdienst oder um einen böswilligen Dienst, der die Datei ausserdem noch heimlich kopiert?
- Authentizität einer *Nachricht*
  - hat mein Kommunikationspartner dies wirklich so gesagt?
  - soll ich als Geldautomat wirklich so viel Geld ausspucken?
- Authentizität *gespeicherter Daten*
  - ist dies wirklich der Vertragstext, den wir gemeinsam elektronisch hinterlegt haben?
  - hat der Autor Casimir von Hinkelstein wirklich *das* geschrieben?
  - ist das Foto nicht eine Fälschung?
  - ist dieser elektronische Schlüssel wirklich echt?

Hilfsmittel zur Authentifizierung

- Wahrung der Nachrichten-Authentizität
  - Verschlüsselung, so dass inhaltliche Änderungen auffallen (Signatur)
  - Fälschung dann nur bei Kenntnis der Verschlüsselungsfunktion möglich
  - Beachte: Authentizität des Nachrichteninhalts garantiert nicht Authentizität der Nachricht als solche! (Replay-Attacke: Neuversenden einer früher abgehörten Nachricht)
  - Massnahmen gegen Replays: mitcodierte Sequenznummer etc.

- Subjekt-/Objekt-Authentifizierung mit *Frage-Antwort-Spiel*
  - "challenge / response": Antworten sollte nur der echte Kommunikationspartner kennen
  - idealerweise stets neue Fragen verwenden (Replay-Attacken!)

- Subjekt-/Objekt-Authentifizierung mit *Passwort*
  - typischerweise zurAuthentifizierung eines Benutzers ("Client") zum Schutz des Dienstes vor unbefugter Benutzung (Autorisierung)
  - Kenntnis des Passworts gilt als Beweis der Identität (?!?)

- Potentielle *Schwächen von Passwörtern*
  - Geheimhaltung (Benutzer kann Passwörter "verleihen" etc.)
  - Raten oder systematische Suche ("dictionary attack")
    - Zurückweisung zu "simpler" Passwörter
    - Zeitverzögerung nach jedem Fehlversuch
    - security logs

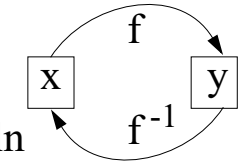
    beachte aber Crack-Programme
  - Abhörgefahr (kein Passwortaustausch im Klartext; Speicherung des Passworts nur in codierter Form, so dass Invertierung prakt. unmöglich
  - Replay-Attacke (Gegenmassnahme: Einmalpasswörter)

    hierfür geeignet: Einwegfunktionen

Einwegfunktionen

- Bilden die Basis für viele kryptographische Verfahren

- Prinzip: $y = f(x)$ *einfach* aus x berechenbar, aber $x = f^{-1}(y)$ ist extrem *schwierig* aus y zu ermitteln



zeitaufwendig (→ praktisch nicht durchführbar)

z.B. $f = O(n)$, $O(n \log n)$,...
aber $f^{-1} = O(2^n)$

- Es gibt (noch) keinen mathematischen Beweis, dass es Einwegfunktionen gibt (aber es gibt einige Funktionen, die es allem Anschein nach sind!)

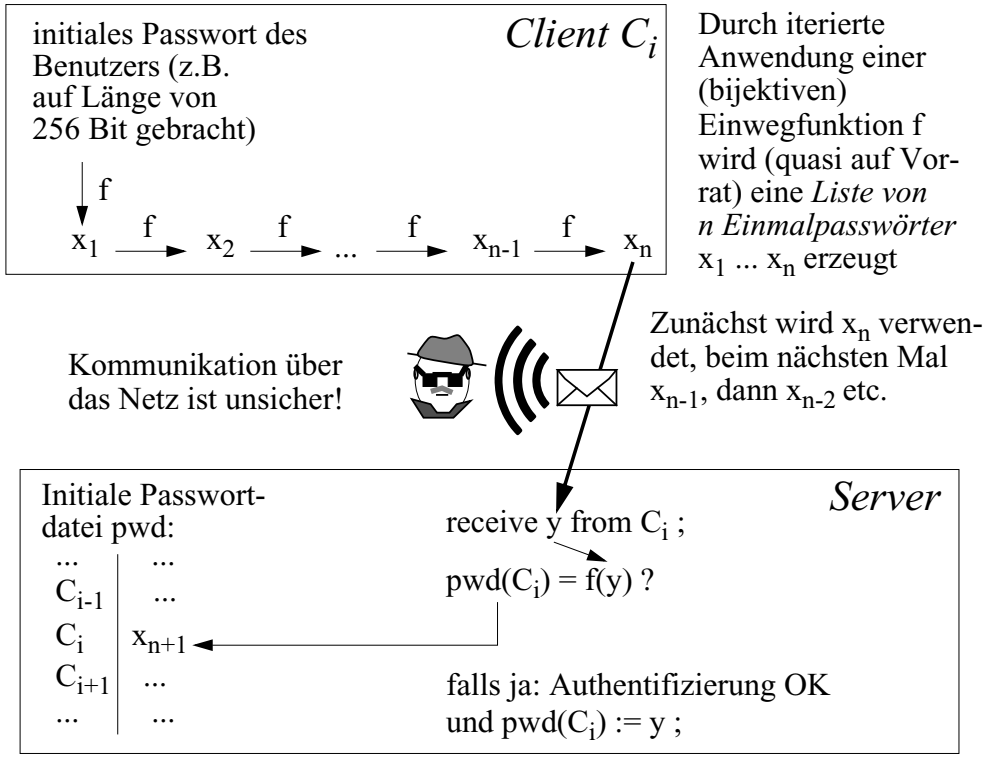
- Einwegfunktionen erscheinen zunächst ziemlich sinnlos: Ein zu $y = f(x)$ verschlüsselter Text x kann nie wieder entschlüsselt werden!

  ⇒ Einwegfunktionen mit "trap-door" (ein Geheimnis, das es erlaubt, f^{-1} effizient zu berechnen)
  - Idee: Nur der "Besitzer" oder "Erfinder" von f kennt dieses
  - Beispiel Briefkasten: Einfach etwas hineinzutun; schwierig etwas herauszuholen; mit Schlüssel (= Geheimnis) ist das aber einfach!
  - Anwendung z.B.: Public key-Verschlüsselung

- Prinzipien typischer (vermuteter) Einwegfunktionen:
  - Das *Multiplizieren* zweier (grosser) Primzahlen p, q ist effizient; das Zerlegen einer Zahl (z.B. $n = pq$) in Primfaktoren i.a. schwierig
  - In einem *Restklassenring* (mod m) ist die Bildung der *Potenz* a^k einfach; die k-te *Wurzel* oder den (diskreten) *Logarithmus* zu berechnen, ist i.a. schwierig. (Aber: k-te Wurzel einfach, wenn Primzerlegung von $m = pq$ bekannt → trap-door!)

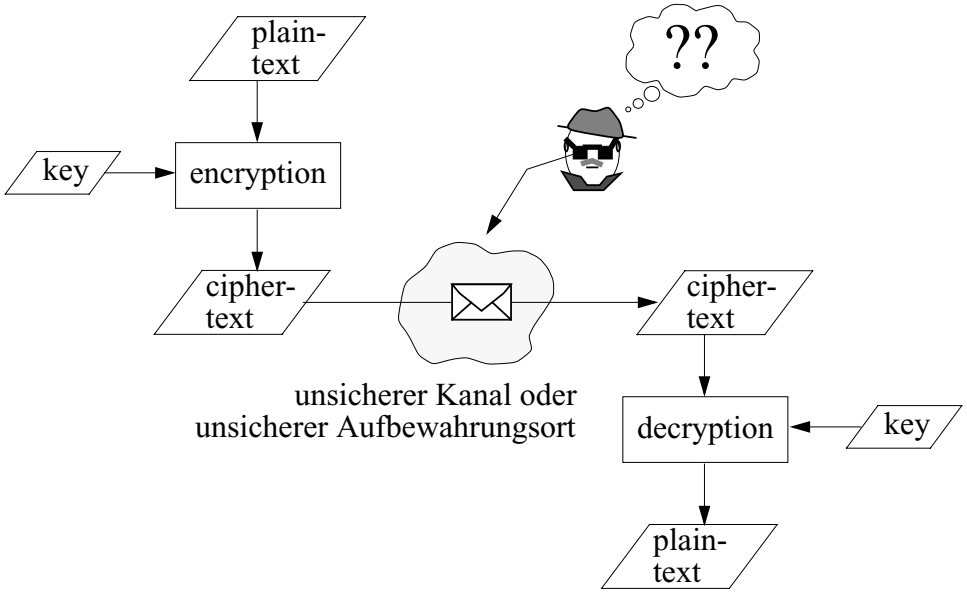
Einmalpasswörter mit Einwegfunktionen

- Szenario: Client gehört dem Benutzer (Notebook, Chipkarte...); Passwörter sind dort sicher aufgehoben



- Ein abgehörtes Passwort x_i nützt nicht viel
  - Berechnung von x_{i-1} aus x_i ist (praktisch) nicht möglich
- Ein Lesen der Passwortdatei des Servers ist nutzlos
  - dort ist das *vergangene* Passwort vermerkt
- Einwegfunktion f muss nicht geheimgehalten werden
  - gute Einwegfunktion prinzipiell nicht effizient umkehrbar
- Realisiert z.B. im S/KEY-Verfahren (RFC 1760)

Kryptosysteme



- Schreibweisen
  - *Verschlüsseln* mit Schlüssel K_1: Schlüsseltext = { Klartext }$_{K1}$
  - *Entschlüsseln* mit Schlüssel K_2: Klartext = { Schlüsseltext }$_{K2}$
- *Symmetrische* Kryptosysteme: K1 = K2
- As*ymmetrische* Kryptosysteme: K1≠ K2

Kryptosysteme (2)

- Geheimhalten des Verschlüsselungsverfahrens i.a. kein Sicherheitsgewinn!
  - organisatorisch kaum lange durchhaltbar
  - kein öffentliches Feedback über erkannte Schwächen des Verfahrens
  - Verfahren, die Geheimhaltung nötig hätten, erscheinen “verdächtig”

- Verschlüsselungsfunktion prinzipiell umkehrbar
  - ohne Kenntnis der Schlüssel jedoch höchstens mit unverhältnismässig hohem Rechenaufwand

---

- Nachteile symmetrischer Schlüssel:
  - Schlüssel muss geheimgehalten werden (da Verfahren i.a. bekannt)
  - mit allen Kommunikationspartnern separaten Schlüssel vereinbaren
  - hohe Komplexität der Schlüsselverwaltung bei vielen Teilnehmern
  - Problem des geheimen Schlüsselaustausches

- Vorteile symmetrischer Schlüssel:
  - ca. 100 bis 1000 Mal schneller als typische asymmetrischeVerfahren

- Beispiele für symmetrische Verfahren:
  - IDEA (International Data Encryption Algorithm): 128-Bit Schlüssel, Einsatz in PGP
  - DES (Data Encryption Standard)
  - AES (Advanced Encryption Standard) als Nachfolger von DES

One-Time Pads

- “Perfektes” Kryptosystem
  - Denkübung: unter welchen Voraussetzungen?

- Prinzip: Wähle zufällige Sequenz von Schlüsselbits
  - Chiffre (Schlüsseltext) = Klartext XOR Schlüsselbitsequenz
  - Entschlüsselung analog: Klartext = Chiffre XOR Schlüsselbitsequenz

Klartext	**V**	**E**	**R**	**T**	**E**	**I**	**L**	**T**	**E**		**S**	**Y**	**S**	**T**	**E**	**M**	**E**
in ASCII	56	45	52	54	45	49	4C	54	45	20	53	59	53	54	45	4D	45
								XOR									
Schlüssel	**4C**	**93**	**EF**	**20**	**B7**	**55**	**92**	**7C**	**DA**	**69**	**23**	**F8**	**BB**	**72**	**0E**	**81**	**00**
= Chiffre	**1A**	**D6**	**BD**	**74**	**F2**	**1C**	**DE**	**28**	**9F**	**49**	**70**	**A1**	**E8**	**26**	**4B**	**CC**	**45**

- Anforderungen an Schlüsselbitsequenz:
  - keine periodische Wiederholung von Bitmustern → Schlüssellänge = Klartextlänge
  - Schlüsselbitsequenz ohne Bildungsgesetz (“echte” Zufallsfolge)
  - Schlüsselbitsequenz ist wirklich “one-time“ (keine Mehrfachverwendung!)

- Kryptoanalyse ohne Kenntnis der Schlüsselbitsequenz ist dann nicht möglich

- Nachteile von One-Time Pads:
  - Verwendung unhandlich (enormer Bedarf an frischen Schlüsselbits, dadurch sehr aufwendiger Schlüsselaustausch)
  - Synchronisationsproblem bei Übertragungsstörungen (wenn Empfang ausser Takt gerät, ist aller Folgetext verloren)
  - nur für hohe Sicherheitsanforderungen gebräuchlich (z.B. “rotes Telefon”)

Asymmetrische Kryptosysteme

Schlimm sind die Schlüssel, die nur schliessen auf, nicht zu;
Mit solchem Schlüsselbund im Haus verarmest du.
Friedrich Rückert, *Weisheit des Brahmanen*

- Schlüssel zum Ver- / Entschlüsseln sind *verschieden*
  - z.B. *RSA-Verfahren* (Rivest, Shamir, Adleman, 1978), beruht auf der Schwierigkeit von Faktorisierung
  - andere Verfahren beruhen z.B. auf diskreten Logarithmen

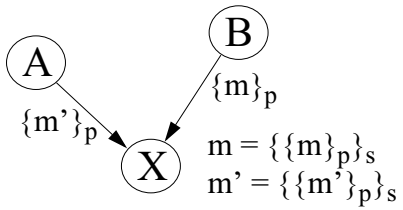
- Für jeden Prozess X existiert ein Paar (p,s)

  p = *public key* ← zum *Verschlüsseln* von Nachrichten an X

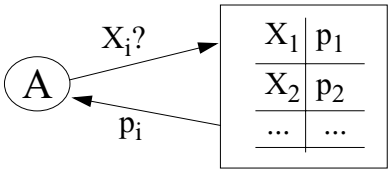
  s = *secret key* ← zum *Entschlüsseln* von mit p verschlüsselten Nachrichten
  (oder "private" key)

- Jeder Prozess, der an X sendet, kennt p

- Nur X selbst kennt s



- *Public-key-Server:*
  Welchen Schlüssel hat Prozess X_i?



  - Server muss allerdings vertrauenswürdig sein
  - Kommunikation zum Server darf nicht manipuliert sein
  - Vielleicht tut es auch ein "Telefonbuch"?

Asymmetrische Kryptosysteme (2)

- Sinnvolle *Forderungen*:
  1) m lässt sich nicht allein aus $\{m\}_p$ ermitteln
  2) s lässt sich aus p oder einer verschlüsselten, bekannten Nachricht nicht (mit vertretbarem Aufwand) ableiten
  3) $m = \{\{m\}_p\}_s$
  4) ggf. zusätzlich: $m = \{\{m\}_s\}_p$
  (Rolle von Verschlüsselung und Entschlüsselung austauschbar)

- Beachte: "Chosen-Plaintext"-Angriff möglich:
  - beliebige Nachrichten M und deren Verschlüsselung { M }p jederzeit generierbar, falls p tatsächlich öffentlich
  - dies darf asymmetrischen Systemen nichts anhaben

- Vorteil gegenüber symmetrischen Verfahren: vereinfachter Schlüsselaustausch
  - jeder darf den übermittelten Verschlüsselungsschlüssel p mithören
  - Entschlüsselungsschlüssel s braucht grundsätzlich nie mitgeteilt zu werden
  - bei n Teilnehmern genügen 2n Schlüssel (statt $O(n^2)$ wie etwa bei DES)

- Kenntnis von s *authentifiziert* zugleich den Besitzer
  - "wer $\{ M \}_{pA}$ entschlüsseln kann, der ist wirklich *A*" (wirklich?)

sA bzw. pA secret
bzw. public key von A

- *Digitale Unterschrift*
  - "wenn (zu M) ein $\{ M \}_{sA}$ existiert mit $\{\{ M \}_{sA} \}_{pA} = M$, dann muss dies (M bzw. $\{ M \}_{sA}$) von A erzeugt worden sein" (wieso?)

Authentifizierung mit symmetrischen Schlüsseln

Sei K der zwischen A und B vereinbarte (und geheim-zuhaltende!) Schlüssel

Problem: B soll die Authentizität von A feststellen.

Idee (Geheimdienstprinzip): "Wenn X *das* weiss und kann, dann muss X wirklich X sein, denn sonst weiss und kann das niemand"

Bemerkung: Oft ist eine *gegenseitige* Authentifizierung nötig

1. Verfahren:

A: m := "Ich bin A"
m' := $\{m\}_K$

A→B: m', m ← (Damit B den richtigen Schlüssel (für A) wählt)

B: überprüfe, ob $\{m\}_K$ = m'

- *Idee*: Überprüfe die Fähigkeit, Nachrichten mit einem geheimen Schlüssel zu kodieren.
- *Nachteil*: Möglichkeit von replays durch Abhören

2. Verfahren:

A→B: "Ich bin A"

B→A: n ← (Einmalkennung ("nonce"))

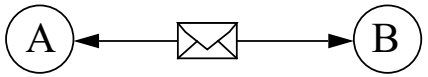
A: n' := $\{n\}_K$

A→B: n'

B: überprüfe, ob $\{n\}_K$ = n'

- *Nachteil*: Viele individuelle Schlüsselpaare für jede Client/Server-Beziehung

Authentifizierung mit asymmetrischen Schlüsseln



Notation: sX = secret key von X;
pX public key von X

Alice **Bob**

„Wer bist du?", *n* ←
n = zufällige Einmalkennung („nonce")

m = „Ich bin A"

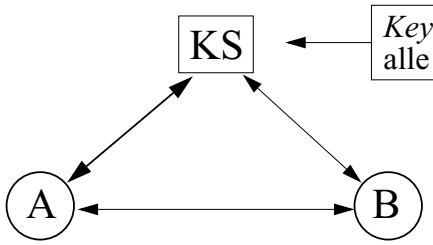
m' = { *m*, *n* }$_{sA}$ — *m*, *m'* → [*m*, *n*] = { *m'* }$_{pA}$? Falls ja:

⇒ Nur A konnte mit *sA* Text *m'* herstellen

digitale Unterschrift

⇒ „Dies muss von Alice stammen!"

- geschützt gegen Replays (wieso?)
- Vorsicht: "Man in the middle"-Angriff möglich (wie?)
- Nachteil: B muss viele public keys speichern; alternativ:



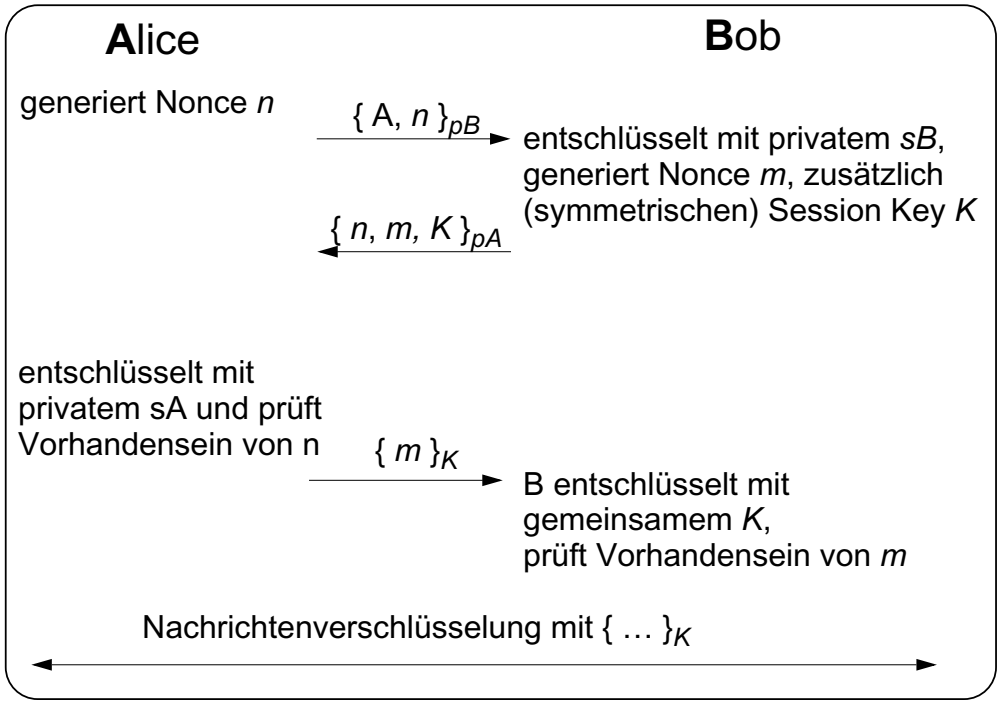
Key Server: Kennt alle public keys

- B erfragt public key von A bei KS
- KS signiert alle seine Nachrichten
- jeder kennt public key von KS (um Unterschrift von KS zu verifizieren)

- Angriff auf den Schlüsselserver KS liefert keine Geheimnisse; erlaubt aber u.U., in dessen Rolle zu schlüpfen und falsche Auskünfte zu geben!
- KS ist ggf. repliziert oder verteilt

Gegenseitige Authentifizierung mit Schlüsselvereinbarung

- Im Prinzip möglich wie oben beschrieben nacheinander in beide Richtungen
- Gleich beides zusammen erledigen ist aber effizienter!
- Hier zusätzlich: Vereinbarung eines symmetrischen "session keys" K, der nach der Authentifizierung zur effizienten Verschlüsselung benutzt wird
- Voraussetzung: A und B kennen die public keys pB bzw. pA des jeweiligen Partners



Replays

- Generelles Problem: Angreifer kann vielleicht eine Nachricht nicht entschlüsseln, jedoch u.U. kopieren und später wieder einspielen
  - elektronische Schecks, Autorisierungscodes für Geldautomaten...

---

1) Verwendung von *Einmalkennungen*, die vom Empfänger vorgegeben werden ("nonce")

→ (fast) alle Nachrichten sind verschieden
- aufwendiges Protokoll aus mehreren Nachrichten

2) Verwendung von mitkodierten *Sequenznummern*
- nur bei einer Nachrichtenfolge zwischen 2 Prozessen möglich

3) Mitverschlüsseln der *Absendezeit*

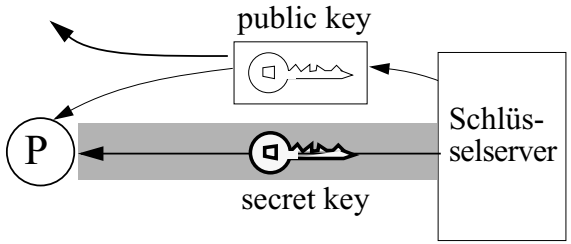
- Empfänger akzeptiert Nachricht nur, wenn seine Zeit max Δt abweicht.

- lokale Uhrzeit
- globale Zeitapproximation aus Zeitservice (z.B. NTP-Protokoll)
- Empfängerzeit vorher erfragen

- Zeitfenster ΔT geschickt wählen!
  - Nachrichtenlaufzeiten berücksichtigen!
  - *zu gross* → unsicher durch mögliche Replays
  - *zu klein* → exakte oder häufige Uhrensynchronisation nötig (z.B. vor jedeNachricht oder nach einem 'reject')
- Angreifer darf Zeitservice nicht manipuliern können!

Schlüsselvergabe

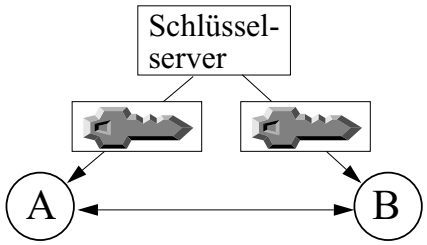
- Zur Vergabe eines Paares von public-, secret-keys:



- secret key muss auf sicherem Kanal zum Client gelangen
- public key von P kann an beliebige Prozesse offen verteilt werden (jedoch i.a. “zertifiziert”, dass der Schlüssel authentisch ist)

- Zur Generierung von temporären symmetrischen Schlüsseln (z.B. “conversation key” / “session key”)

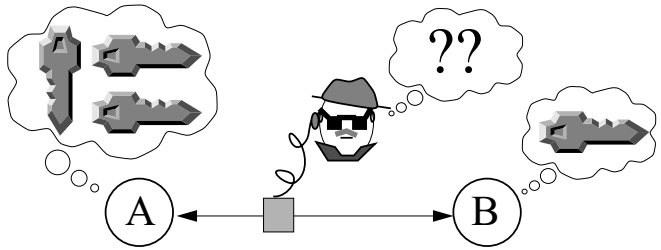
→ erhöhte Sicherheit gegen Angreifer



- z.B.: Schlüsselserver generiert DES-keys
- diese werden sicher und authentisch mit einem Public-key-Verfahren an zwei Kommunikationspartner übertragen

- Schlüsselserver kann DES-keys nach Übertragung bei sich löschen
- Aufwendiges Public-key-Verfahren nur ein Mal pro “Session”, tatsächliche Nachrichten zwischen A und B effizienter per DES

Direkter Schlüsselaustausch

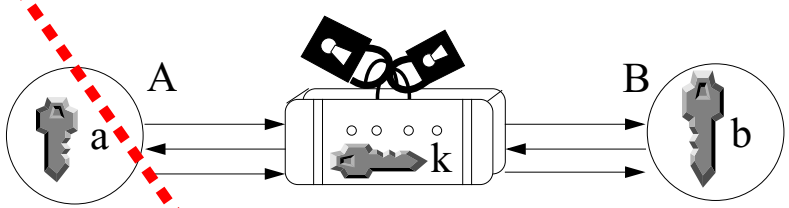


- Problem: A und B wollen sich über einen unsicheren Kanal auf einen gemeinsamen Schlüssel einigen, ohne einen Schlüsselserver zu verwenden

- Sinnvoll z.B. bei dynamisch gegründeten Prozessen, die vorher noch nie kommuniziert haben
  - z.B. wenn keine public keys vorhanden bzw. nicht bekannt

- Wie geht dies?
  - wir erinnern uns an die “Schatzkiste mit zwei Vorhängeschlössern”

Kommutative Schlüssel



1. A generiert einen Sitzungsschlüssel k
2. A verschlüsselt k mit einem geheimen Schlüssel a
3. A→B: $\{k\}_a$
4. B verschlüsselt dies mit seinem Schlüssel b
5. B→A: $\{\{k\}_a\}_b$
6. A entschlüsselt mit seinem Schlüssel a:
   $\{\{\{k\}_a\}_b\}_{\bar{a}} = \{\{\{k\}_a\}_{\bar{a}}\}_b = \{k\}_b$
7. A→B: $\{k\}_b$
8. B entschlüsselt mit seinem Schlüssel: $\{\{k\}_b\}_{\bar{b}} = k$

a und b sind "lokal erfunden"

Forderung!

Bezeichne $\bar{x}$ den zu x inversen Schlüssel (oft: $\bar{x}=x$)

gemeinsames Geheimnis

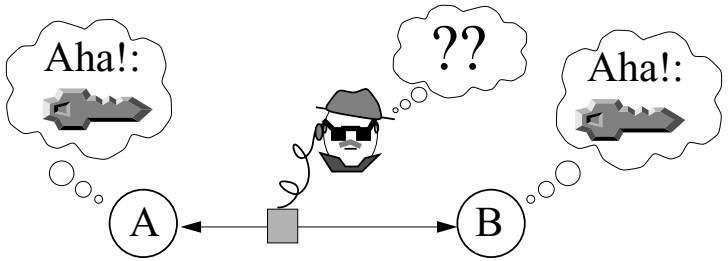
Beachte: k wird nie offen transportiert!

Denkübung: Geht hier *xor* mit "one-time pads" a, b ?

- *xor* erfüllt die Forderung (ist assoziativ und kommutativ)
- *xor* mit one-time pads ist sicher (wirklich?) und effizient
- *aber:* Wenn Schritt 3 ($\{k\}_a$) und Schritt 5 ($\{\{k\}_a\}_b$) abgehört wird, dann kann daraus der Schlüssel b ermittelt werden, so dass aus dem abgehörten Schritt 7 ($\{k\}_b$) das geheime k ermittelt werden kann!
- gibt es anstelle von xor andere (sichere!) Verschlüsselungsoperationen?

Schlüsselvereinbarung mit Diffie-Hellman-Verfahren

Ziel: A und B sollen sich über einen unsicheren Kanal auf ein gemeinsames "Geheimnis" G einigen, ohne dass ein Angreifer es erfährt



- Nutzung einer *Einwegfunktion*: $f(x) = c^x \bmod p$

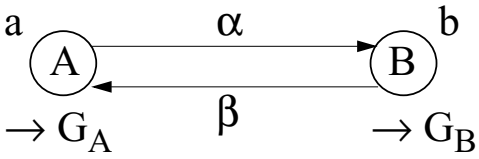
  $(1 < c < p$; i.a. ist p eine grosse Primzahl)

- in einem Restklassenring ist die Bestimmung *diskreter Logarithmen* (und k-ter Wurzeln) wesentlich schwieriger als die Bildung von Potenzen

- im RPC-Protokoll von Sun wird z.B. c=3 gewählt und p = d4a0ba0250b6fd2ec626e7efd637df76c716e22d0944b88b (hex.); eine Zahl aus 192 Bits (die Parameter c und p sind kein Geheimnis)

- gelegentlich wird für p auch ein Produkt aus zwei grossen Primzahlen empfohlen, oder es wird $p = 2^n$ gewählt, da dann die mod-Operation besonders einfach zu berechnen ist

Der Algorithmus



- wenig Nachrichten
- effizient

1. A wählt eine Zufallszahl a
2. A berechnet $\alpha = f(a)$
3. A → B: α
4. B wählt eine Zufallszahl b
5. B berechnet $\beta = f(b)$
6. B → A: β
7. A berechnet $G_A = \beta^a \bmod p$
8. B berechnet $G_B = \alpha^b \bmod p$

(a und b sind nur lokal bekannt und bleiben geheim)

Behauptung: $G_A = G_B$ (gemeinsames Geheimnis!)

Beispiel (für c = 5 und unrealistisch kleines p = 7):

$f(x) = 5^x \bmod 7$

$a = 3 \rightarrow \alpha = 6 \rightarrow G_B = 6^4 \bmod 7 = 1$

$b = 4 \rightarrow \beta = 2 \rightarrow G_A = 2^3 \bmod 7 = 1$

$G_A = G_B$

Zu zeigen: $\beta^a \bmod p = \alpha^b \bmod p$, also:

$$(c^b \bmod p)^a \bmod p = (c^a \bmod p)^b \bmod p$$

Lemma: $(k \bmod p)^n \bmod p = k^n \bmod p$ ← Restklassen-arithmetik...

$$\begin{aligned}
\mathbf{(c^b \bmod p)^a \bmod p} &= (c^b)^a \bmod p && \text{[Lemma]} \\
&= c^{(b\,a)} \bmod p \\
&= c^{(a\,b)} \bmod p \\
&= (c^a)^b \bmod p && \text{[Lemma]} \\
&= \mathbf{(c^a \bmod p)^b \bmod p}
\end{aligned}$$

Bemerkungen:

- Lässt sich auch auf k > 2 Benutzer verallgemeinern
- Der Algorithmus (entdeckt 1976) ist patentiert
  - U.S.-Patent Nummer 4200770 (Sept. 1977)

United States Patent [19]

Hellman et al.

[11] **4,218,582**

[45] **Aug. 19, 1980**

[54] **PUBLIC KEY CRYPTOGRAPHIC APPARATUS AND METHOD**

[75] Inventors: **Martin E. Hellman**, Stanford; **Ralph C. Merkle**, Palo Alto, both of Calif.

[73] Assignee: **The Board of Trustees of the Leland Stanford Junior University**, Stanford, Calif.

[21] Appl. No.: **839,939**

[22] Filed: **Oct. 6, 1977**

[51] **Int. Cl.**[2] **H04L 9/04**

[52] **U.S. Cl.** **178/22**; 364/900

[58] **Field of Search** 178/22

[56] **References Cited**

PUBLICATIONS

"New Directions in Cryptography," Diffie et al., *IEEE Transactions on Information Theory*, vol. II22, No. 6, Nov. 1976, pp. 644–654.

"A User Authentication Scheme not Requiring Secrecy in the Computer," Evans, Jr., et al., *Communications of the ACM*, Aug. 1974, vol. 17, No. 8, pp. 437–442.

"A High Security Log-In Procedure," Purdy, *Communications of the ACM*, Aug. 1974, vol. 17, No. 8, pp. 442–445.

Diffie et al., "Multi-User Cryptographic Techniques," *AFIPS Conference Proceedings*, vol. 45, pp. 109–112, Jun. 8, 1976.

Primary Examiner—Howard A. Birmiel

[57] **ABSTRACT**

A cryptographic system transmits a computationally secure cryptogram that is generated from a publicly known transformation of the message sent by the transmitter; the cryptogram is again transformed by the authorized receiver using a secret reciprocal transformation to reproduce the message sent. The authorized receiver's transformation is known only by the authorized receiver and is used to generate the transmitter's transformation that is made publicly known. The publicly known transformation uses operations that are easily performed but extremely difficult to invert. It is infeasible for an unauthorized receiver to invert the publicly known transformation or duplicate the authorized receiver's secret transformation to obtain the message sent.

17 Claims, 13 Drawing Figures

US4218582: Public key cryptographic apparatus and method

Inventor(s): Hellman; Martin E. , Stanford, CA Merkle; Ralph C. , Palo Alto, CA

Issued/Filed Dates: Aug. 19, 1980 / Oct. 6, 1977

Abstract:

A cryptographic system transmits a computationally secure cryptogram that is generated from a publicly known transformation of the message sent by the transmitter; the cryptogram is again transformed by the authorized receiver using a secret reciprocal transformation to reproduce the message sent. The authorized receiver's transformation is known only by the authorized receiver and is used to generate the transmitter's transformation that is made publicly known. The publicly known transformation uses operations that are easily performed but extremely difficult to invert. It is infeasible for an unauthorized receiver to invert the publicly known transformation or duplicate the authorized receiver's secret transformation to obtain the message sent.

What is claimed is:

1. In a method of communicating securely over an insecure communication channel of the type which communicates a message from a transmitter to a receiver, the improvement characterized by: providing random numbers at the receiver; generating from said random numbers a public enciphering key at the receiver; generating from said random numbers a secret deciphering key at the receiver such that the secret deciphering key is directly related to and computationally infeasible to generate from the public enciphering key; communicating the public enciphering key from the receiver to the transmitter; processing the message and the public enciphering key at the transmitter and generating an enciphered message by an enciphering transformation, such that the enciphering transformation is easy to effect but computationally infeasible to invert without the secret deciphering key; transmitting the enciphered message from the transmitter to the receiver; and processing the enciphered message and the secret deciphering key at the receiver to transform the enciphered message with the secret deciphering key to generate the message.

2. ...

...

17. ...

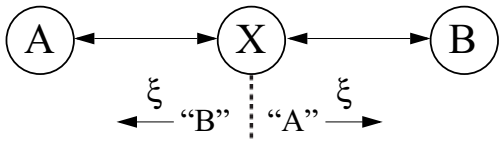
Sweet Little Secret G

- A und B könnten beide $G = G_A = G_B$ als symmetrischen DES-Schlüssel zur Verschlüsselung ihrer Nachrichten verwenden

- *Besser*: G nur als Schlüssel verwenden, um einen zufällig bestimmten Session-key zu kodieren und dem Kommunikationspartner diesen mitzuteilen
  - so wird es im Sun-RPC-Protokoll gemacht
  - Motivation: G so selten wie möglich benutzen

- Einzusehen bliebe noch, dass aus Kenntnis von α und β (sowie von c und p aus f) G von einem passiven Angreifer nicht effizient ermittelt werden kann!
  - $\alpha = c^a \bmod p \rightarrow$ a ist ein *diskreter Logarithmus*; dieser ist i.a. "schwierig" zu berechnen!
  - Bem.: nicht jedes p ist "gut"; sollte auch einige 100 Bit gross sein
  - "Probieren" aller a, bis $\alpha = c^a \bmod p$ gefunden $\rightarrow$ langwierig
  - α und β sind *unabhängig* voneinander! (Wieso ist das ein Argument?)

- Wie ist es aber bei *aktiven Angreifern*?

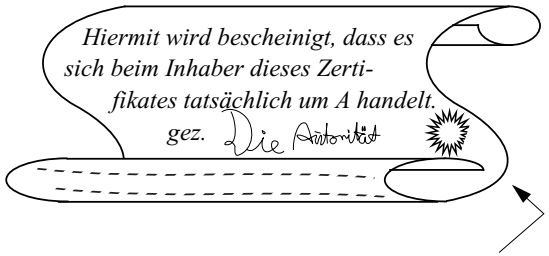


- "man in the middle"
- ein ξ für ein β bzw. α vormachen!

- X kann unter Vortäuschung falscher Identitäten eigene Schlüssel für die Teilstrecken AX und XB vereinbaren!

Authentifizierung mit Zertifikaten ?

- Die Idee:



- A lässt sich einmalig von einer Autorität ein *Zertifikat* Z_A mitgeben (sollte von der Autorität signiert sein)
  - Autorität gilt als vertrauenswürdig und hat A ggf. persönlich in Augenschein genommen (oder einem fremden Zertifikat vertraut)

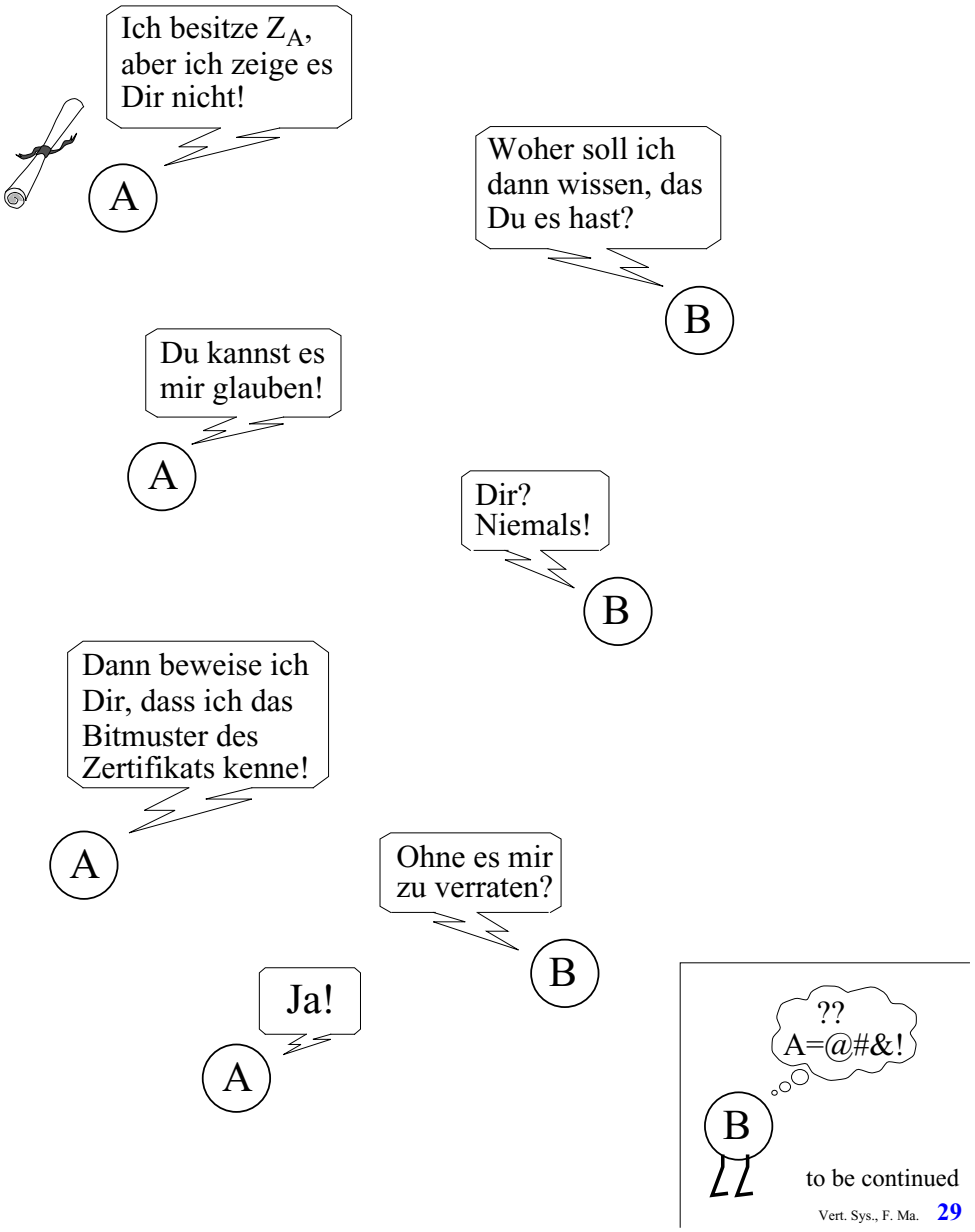
- Wenn B an der Identität von A zweifelt, wird B von A auf sein Zertifikat Z_A hingewiesen
  - Besitz des Zertifikates = Authentifizierung

- Aber: A darf Z_A nie B zeigen - sonst könnte B es sich kopieren und sich fortan als A ausgeben!
  - wie vermeidet man "raubkopierte Zertifikate"?
  - in der digitalen Welt lassen sich Bitfolgen perfekt kopieren (im Unterschied zu "fälschungssicheren Ausweisen")

- Z_A muss offenbar ein *Geheimnis* bleiben, das ausser der Autorität und A niemand kennt!

- Taugt ein solches Geheimnis als Zertifikat??
  - wie beweist man den Besitz eines Zertifikates, ohne es zu zeigen?

Geheime Zertifikate ?



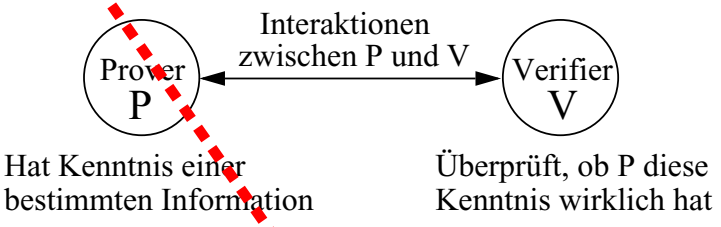
Geheime Zertifikate !

- Im Prinzip wissen wir schon, dass das geht: Der secret key sA eines asymmetrischen Verfahrens stellt ein solches Zertifikat dar
  - braucht von A nicht verraten zu werden
  - B kann dennoch überprüfen, ob A das Zertifikat hat (z.B. indem sich B von A etwas mit sA verschlüsseln lässt und anschliessend durch Anwenden von pA prüft; oder indem B ein $\{M\}_{pA}$ an A schickt und A dies mit sA entschlüsselt)

- Eine andere Realisierung geht mit “zero knowledge”
  - beweist Kenntnis eines Geheimnisses G, ohne geringste Information preiszugeben

Zero-Knowledge-Proofs

- “Beweis” = Nachweis, dass P eine bestimmte Folge von Bits (= Zahl, Algorithmus, Zertifikat...) kennt.



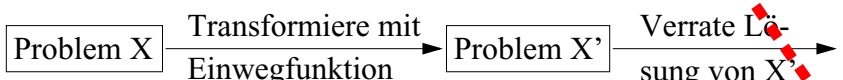
- P kann V *(praktisch) nicht betrügen:* Wenn P die Information nicht hat, sind seine Chancen, V zu überzeugen, verschwindend gering

- V *erfährt nichts* über die eigentliche Kenntnis von P

- V erfährt auch sonst nichts durch P, was V nicht auch alleine in Erfahrung bringen könnte

---

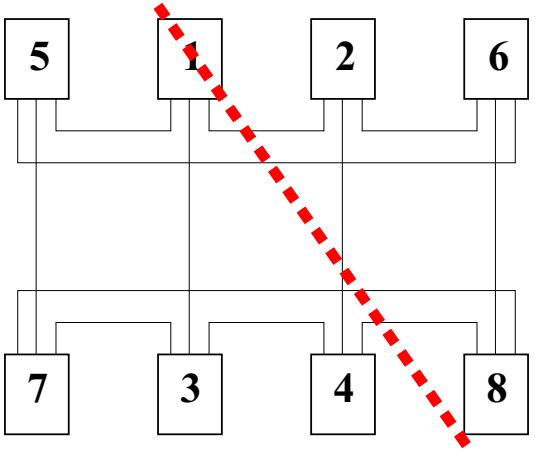
Idee:



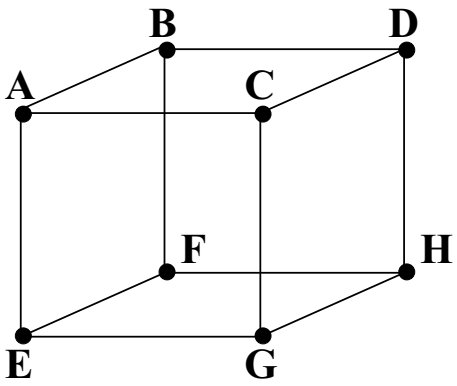
(Wobei die Lösung von X’ die Lösung von X logisch impliziert, sie jedoch nicht effektiv-konstruktiv liefert)

Beispiel: Isomorphie von Graphen

Bemerkung: Ob zwei grosse (z.B. in Form von Adjazenzmatrizen) gegebene Graphen G_1, G_2 topologisch isomorph ($G_1 \sim G_2$) sind (d.h. bis auf Umbenennung von Knoten und ggf. Kanten identisch sind), ist ein *schwieriges* Problem.



≡



A = 7
B = 5
C = 8
D = 6
E = 3
F = 1
G = 4
H = 2

Überprüfung eines durch eine Knoten-zuordnung gegebenen Isomorphismus ist allerdings “einfach”!

Zero-Knowledge mit Graphisomorphie

- P behauptet, einen Beweis zu haben, dass zwei gegebene Graphen G_1, G_2 isomorph sind, möchte den Beweis aber nicht verraten

- Folgendes Protokoll *überzeugt* V davon:

  - P erzeugt durch zufällige Permutation der Knoten einen Graphen H mit $H \sim G_1$ (und damit $H \sim G_2$). Für P ist dies einfach. V aber kann $H \sim G_1$ oder $H \sim G_2$ nicht einfacher entscheiden als $G_1 \sim G_2$

  - P sendet H an V

  - Entweder bittet V dann P

    - $H \sim G_1$ nachzuweisen, *oder*
    - $H \sim G_2$ nachzuweisen

  zufällig; bzw. von P nicht vorhersehbar

  - Da P den Graphen H konstruiert hat, kann P das gewünschte einfach tun (P hütet sich jedoch davor, auch noch die von V nicht gewünschte Alternative nachzuweisen - wieso?)

  - P und V wiederholen alles n Mal, wobei von P jedesmal ein anderer "Zeuge" H konstruiert wird (Beweissicherheit = $1\text{-}2^{-n}$)

- Der Isomorphismus bleibt dabei ein Geheimnis von P!

Zero-Knowledge: Eigenschaften

- Falls P *keinen* Isomorphismus zwischen G_1 und G_2 kennt (also *lügt*), kann P keinen Graphen H konstruieren, der nachweislich isomorph zu beiden ist

  - *Verschiedene* H_1, H_2 zu finden mit $H_1 \sim G_1$ und $H_2 \sim G_2$ ist einfach; mit 50% Wahrscheinlichkeit wird P dann allerdings der Lüge überführt!

- V *lernt nichts* über die Isomorphie $G_1 \sim G_2$, *glaubt* aber schliesslich, dass P eine solche kennt

- Zur Minimierung der Interaktionen lassen sich die "Runden" *parallelisieren*: P sendet *mehrere* "isomorphe Zeugen" an V, und V sendet einen Bitvektor zurück, der die Einzelnachweise auswählt

- V kann einem Dritten W gegenüber nicht beweisen, dass P den Isomorphismus kennt: Selbst ein exaktes Protokoll der Kommunikationsvorgänge muss W nicht überzeugen: P und V könnten sich *verschworen* haben!

- Da V nichts gelernt hat, kann V sich anderen gegenüber auch nicht mit der Kenntnis schmücken, sich also *nicht für P ausgeben* (wenn die Kenntnis P identifiziert)

Es gilt: *Jeder mathematische Beweis kann in einen Zero-knowledge-Proof transformiert werden!* (Hier nur Beweisansatz: Jedes math. Theorem kann als Graph dargestellt werden, so dass Theorem gilt gdw. Graph Hamilton-Zykel besitzt; Zero-Knowledge-Beweis, dass Graph Hamilton-Zykel besitzt, wie folgt: für isomorphen Graphen H entweder Isomorphie zu G oder aber Hamilton-Zykel offenlegen)

Der Kerberos-Sicherheitsdienst

- Protokoll zur Schlüsselvergabe, Authentifizierung und Einrichtung sicherer Kommunikationskanäle
- Am MIT entwickelt im Rahmen des Athena-Projekts
  - war dort ab 1986 im Einsatz

erstes grosses Client-Server-Campusnetz

- Basiert auf Needham-Schroeder-Protokoll mit symmetrischen Schlüsseln (i.a. DES)
- Public domain; es gibt auch kommerzielle Varianten
- In heutigen "offenen" verteilten Systemen gibt es noch andere Systemdienste zur Erhöhung der Sicherheit
  - z.B. ssh, VPN etc.

R.M. Needham, M.D. Schroeder: *Using Encryption for Authentication in Large Networks of Computers.* CACM 21(12), pp. 993-999, 1978

J.I. Schiller: *Sicherheit im Daten-Nahverkehr.* Spektrum der Wissenschaften 1/1995, pp. 50-57, Januar 1995

B. Clifford Neuman and Theodore Ts'o: *Kerberos: An Authentication Service for Computer Networks.* IEEE Communications Magazine, Volume 32, Number 9, pp. 33-38, September 1994. Im Internet: http://nii.isi.edu/publications/kerberos-neuman-tso.htm

RFC 1510: *The Kerberos Network Authentication Service (V5),* ftp://ftp.isi.edu/in-notes/rfc1510.txt

Kerberos-"Philosophie"

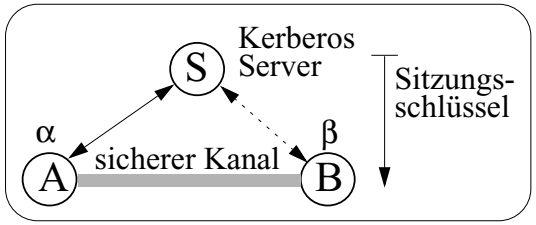
- Offenes Campusnetz → Nachrichten prinzipiell unsicher
- Kommunikation daher i.a. verschlüsselt und nur mit authentifizierten Partnern
  - Kenntnis des Sitzungsschlüssels als Authentitätsbeweis
- Passwörter niemals im Klartext übertragen
  - auch keine Passwortspeicherung

"Kerberos-Server"

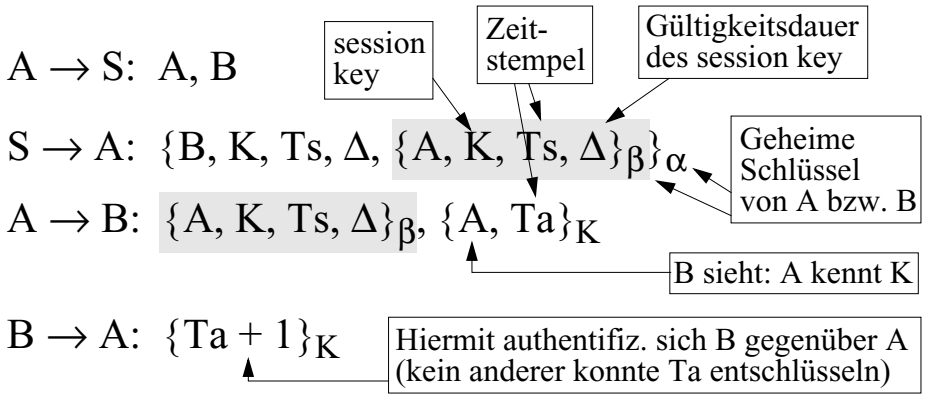
- Alle Benutzer, Clients und Server sind bei zentraler Instanz (Key Distribution Center: "KDC") akkreditiert
  - vereinbaren mit dem KDC auch ihren Geheimschlüssel ("master key")
  - ohne Akkreditierung keine Server-Berechtigungsscheine ("Tickets")
  - ohne Tickets kein Service
  - Ticket nur in Verbindung mit Authentitätsnachweis gültig
- Gültigkeit von Tickets / Sitzungsschlüsseln zeitlich befristet
- Drei Sicherheitsstufen möglich
  (1) Authentifizierung nur bei Einrichtung eines Kommunikationskanals
  (2) Authentifizierung bei jeder Nachricht zwischen A und B
  (3) zusätzlich Verschlüsselung der Nachrichten

Kerberos-Anwendungsbeispiel: Einrichtung eines sicheren Kanals

- Wechselseitige Authentifizierung (via Kerberos Server)
- Verwendung eines Sitzungsschlüssels ("session key")
- $\{X, K, Ts, \Delta\}_\gamma$ heisst "Ticket"
  - Tickets kann man an andere ("vertrauenswürdige") Instanzen weitergeben

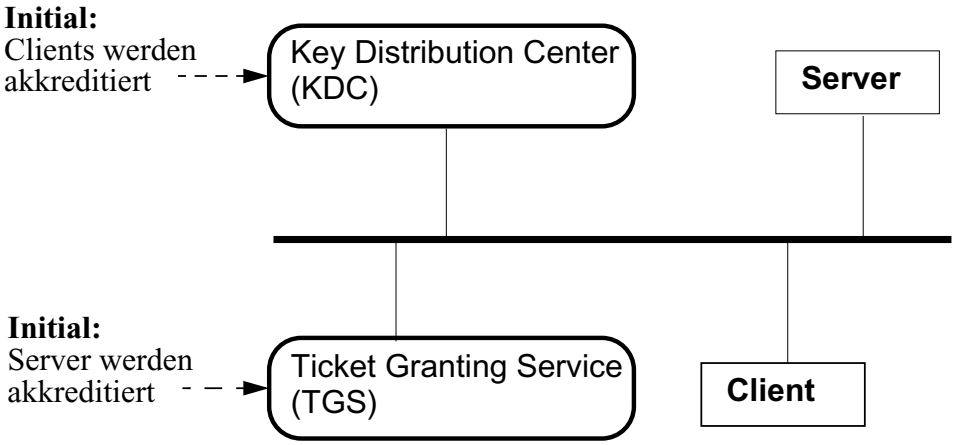


Hier: Version 4 (MIT-Version eingefroren Dez. '92); spätere Versionen im Prinzip nur leicht unterschiedlich



- Geheimschlüssel α und β darf ausser S und A bzw. B niemand kennen! (Kenntnis wird als Identitätsnachweis betrachtet)
- A reicht hier ein von S erhaltenes Ticket an B weiter

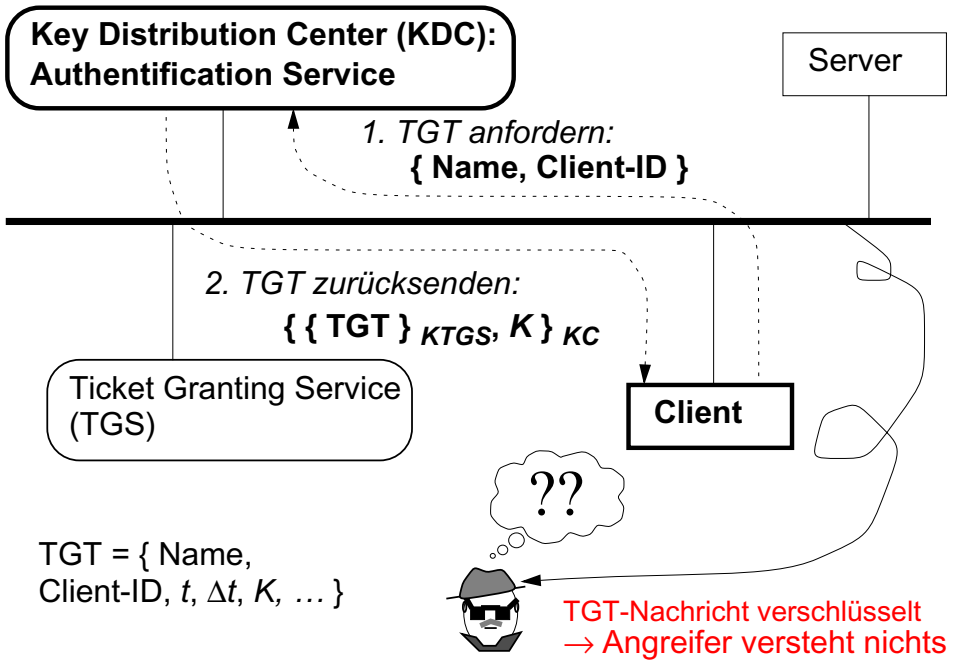
Kerberos: Akkreditierung



- Benutzer und deren Passwörter (= Schlüssel) werden dem KDC bekannt gemacht
- TGS und dessen geheimer Schlüssel werden ebenfalls beim KDC akkreditiert
- Server und deren geheime Schlüssel werden dem TGS bekannt gemacht
  - es kann mehrere TGS-Server geben (→ Lastverteilung)

Kerberos: TGT-Anforderung

- Client erwirbt zunächst ein Ticket Granting Ticket (TGT)



- Client an KDC: sendet { Name, Client-ID } im Klartext
- KDC: wählt K; erstellt TGT = { Name, Client-ID, t, Δt, K, … }
- KDC an Client: sendet { { TGT } $_{KTGS}$, K } $_{KC}$ zurück;
  KC = h(Passwort); KTGS = TGS-Schlüssel; K = Sitzungsschlüssel
- Client: gewinnt { TGT } $_{KTGS}$ und K durch Entschlüsselung mit Passwort:
  - (chiffriertes) TGT berechtigt zum Erwerb von Service Tickets;
  - K sichert Kommunikation mit TGS gegen Angreifer

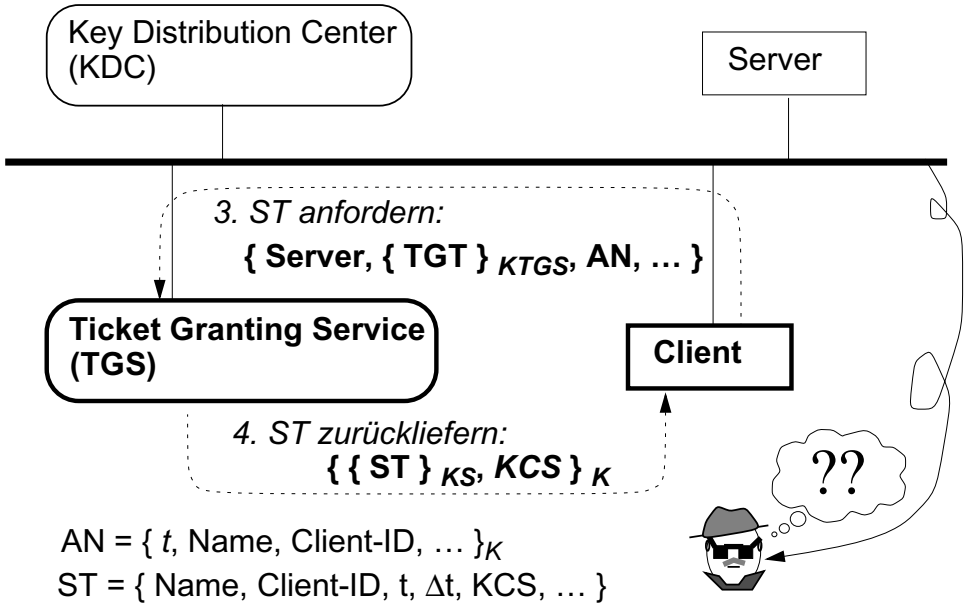
→ KDC-Nachricht ist authentisch: Nur KDC kennt noch Schlüssel KC !

→ Nur der echte Client kann TGT mittels KC nutzbar machen

→ Passwort verlässt Client-Rechner nicht und wird sofort wieder gelöscht

→ TGT ist verschlüsselt, nur für Zeitspanne Δt gültig, geht nur an Client

Kerberos: Service Ticket erwerben

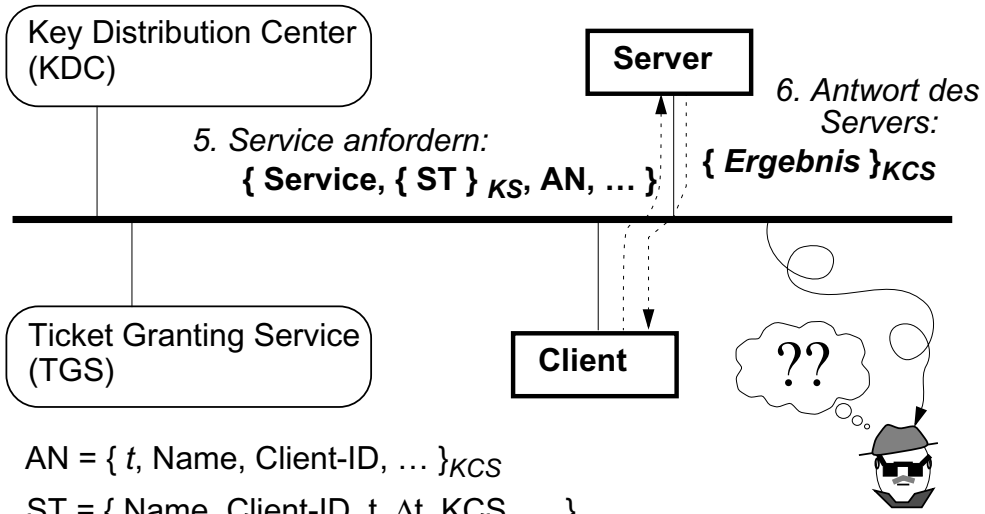


- Client: erstellt Authentizitätsnachweis AN = { t, Name, Client-ID, … } $_{K}$
- Client sendet an TGS { Server, { TGT } $_{KTGS}$, AN, … } als Request
- TGS: entschlüsselt TGT mit Schlüssel KTGS, erhält damit K;
  entschlüsselt AN mit K, vergleicht Inhalt mit TGT;
  erstellt Service Ticket ST = { Name, Client-ID, t, Δt, KCS, … }
- TGS sendet an Client { { ST } $_{KS}$, KCS } $_{K}$ zurück
- Client: gewinnt { ST } $_{KS}$ und KCS durch Entschlüsselung mit K:
  - (chiffriertes) ST berechtigt zur Nutzung des Servers
  - KCS sichert Kommunikation zwischen Client und Server

→ Ohne Sitzungsschlüssel K ist ST nicht nutzbar: Nur Client kennt K !

→ ST höchstens für Zeitspanne Δt gültig

Kerberos: Nutzung des Service

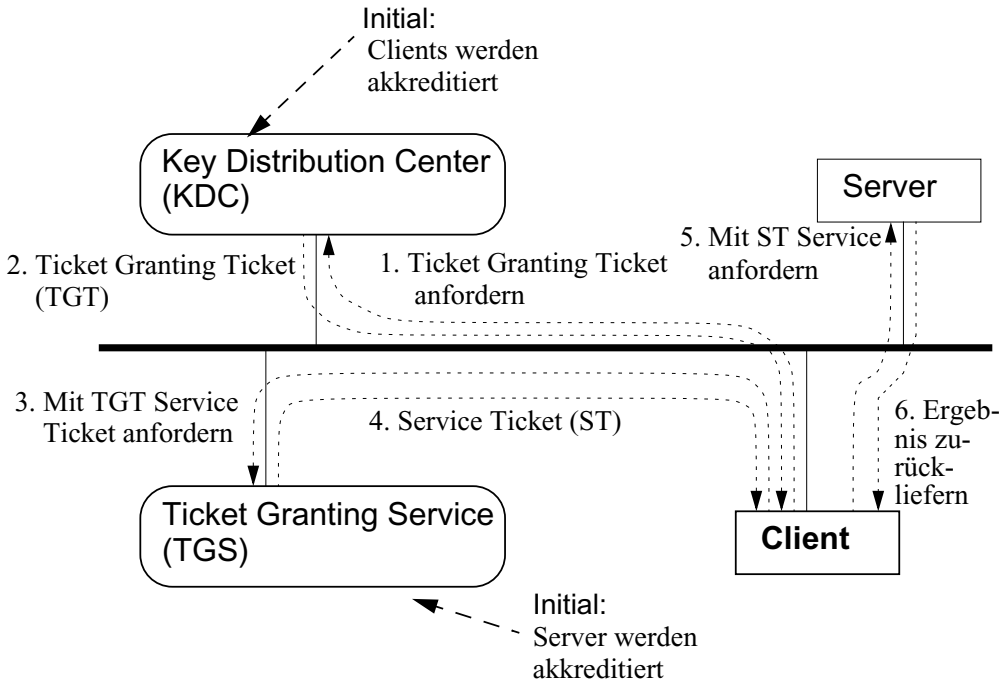


- Client: erstellt Authentizitätsnachweis AN = { t, Name, Client-ID, … }$_{KCS}$
- Client an Server: sendet { Service, { ST } $_{KS}$, AN, … } als Service Request
- Server: entschlüsselt ST mit KS, erhält damit KCS;
  entschlüsselt AN mit KCS, vergleicht Inhalt mit ST;
  leistet Service und erzeugt Ergebnisdaten
- Server an Client: antwortet mit { Ergebnisdaten }$_{KCS}$
- Client: authentifiziert und entschlüsselt das Ergebnis mittels KCS

→ Folgedialoge zwischen Client und Server mittels KCS verschlüsselbar

→ ST als Einmal-Ticket oder ggf. innerhalb Δt mehrfach nutzbar

Kerberos: Protokollübersicht



- Protokoll ist zweistufig:
  - Client kommuniziert nur selten mit dem KDC (1,2) → eigentlicher Geheimschlüssel (Passwort-basiert) wird nur selten benutzt
  - ein TGT ist für mehrere Anfragen beim Ticket-Service gültig

Kerberos - weitere Aspekte

- Nachrichten enthalten noch weitere (technische) Angaben
  - z.B. Versionsnummer, Nachrichtentyp, Prüfsumme, Netzwerkadresse...
- Es gibt dezentrale Zuständigkeitsbereiche (“realms”)
  - lok. KDC vermittelt Zugangsticket zu KDC eines fremden Bereichs
- Kerberos-Software enthält u.a.:
  - Library mit Routinen, um Authentifizierungsanforderungen erzeugen und lesen zu können, Nachrichten zu authentifizieren und zu verschlüsseln
  - Datenbank und Verwaltungsroutinen für registrierte Nutzer (Geheimschlüssel, Gültigkeitsdauer, Verwaltungsdaten...)
  - Tools zur Replikation der Datenbank (Verteilung ist wichtig, da bei Ausfall des KDC fast nichts mehr im ganzen Netz geht!)
- Neuere Versionen (gegenüber Version 4): mehr Funktionalität und allgemeiner verwendbar, z.B.:
  - standardisierte Datenformate
  - Verbesserung einiger Sicherheitskonzepte; Alternativen zu DES möglich
  - besser skalierbare “Cross Realm”-Authentifikation
  - Unterstützung erneuerbarer und transferierbarer Tickets
- Weiterentwicklungen
  - z.B. asymm. Schlüssel, Einbindung von Chipkarten, verteilte Datenbank...
- Kerberos ist weit verbreitet (“Quasi-Standard”)
  - z.B. um verteilte Dateiserver (NFS, AFS) zu sichern oder modifizierte Versionen von telnet, rlogin, rcp, rsh, ftp etc. zu ermöglichen
  - kommerzielle Varianten z.T. nicht kompatibel zueinander
  - Microsoft: ab Windows 2000

Kerberos - Sicherheitsaspekte

- KDC und TGS müssen geschützt werden
  - z.B. gegen unbefugtes Lesen der Datenbank, Verändern der Daten, Einschleichen, denial of service...
- Tickets müssen vom Client in einem “sicheren Speicherbereich” aufbewahrt werden
  - Master key (aus Passworteingabe des Benutzers abgeleitet) wird sobald wie möglich aus dem Speicher gelöscht
- Uhren der Kommunikationspartner und der Kerberos-Server müssen “verlässlich” synchronisiert werden
  - innerhalb eines gewissen Toleranzintervalls von einigen Minuten
  - Störung des Uhrenabgleichs erlaubt ggf. mehrfachen Ticketmissbrauch
- Replays sind innerhalb der Gültigkeitsdauer (typw.: einige Minuten bis Stunden) prinzipiell möglich!
  - Server sollte alte, noch gültige Tickets speichern, um Replays ggf. erkennen zu können (man beachte aber, dass z.B. NFS ein zustandsloses Protokoll besitzt!)
- Auf public domain Servern (und CDs etc.) könnte gefälschte Software vorhanden sein (“trojanische Pferde”)
- “Erster” Schlüssel basiert auf einem Passwort → Off-line-Attacke durch Raten gängiger Passworte
- Hintertüren ausserhalb von Kerberos
  - fremde Tickets lesen (Netz-sniffer, Superuser-Rechte beschaffen...)
  - “Hijacking” von TCP-Verbindungen