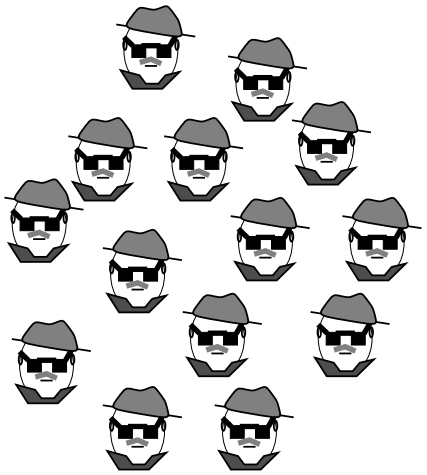
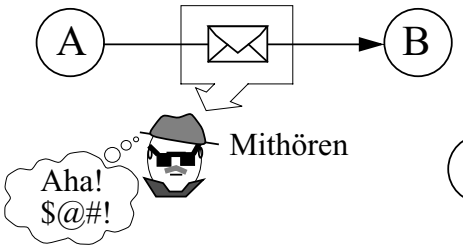
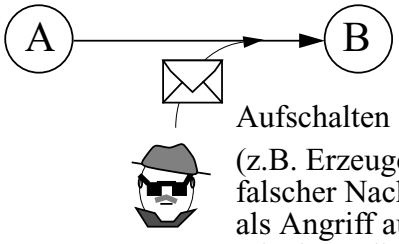
Sicherheit



Sicherheit in verteilten Systemen

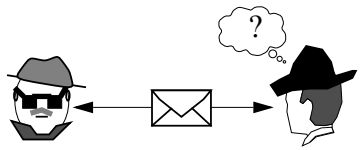


Mithören

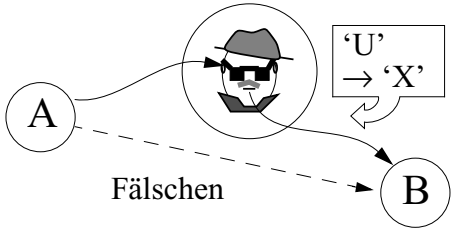


Aufschalten

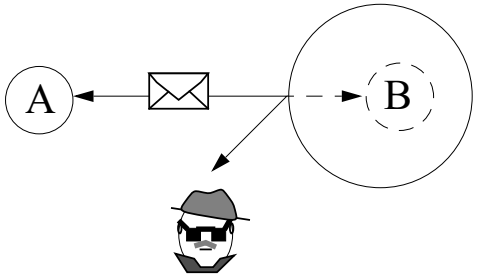
(z.B. Erzeugen falscher Nachrichten als Angriff auf die Glaubwürdigkeit)



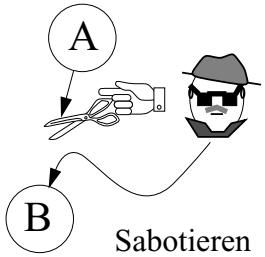
Wechselseitiges Misstrauen



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. „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
  - logische Offenheit führt zu Anonymität
  - technologische Gegebenheiten: z.B. Wireless LAN
- *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":

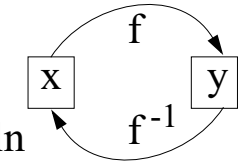
- *Verschlüsselung*
- *Autorisierung* ("der darf das!")
- *Authentisierung* ("X ist wirklich X!")

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

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!)

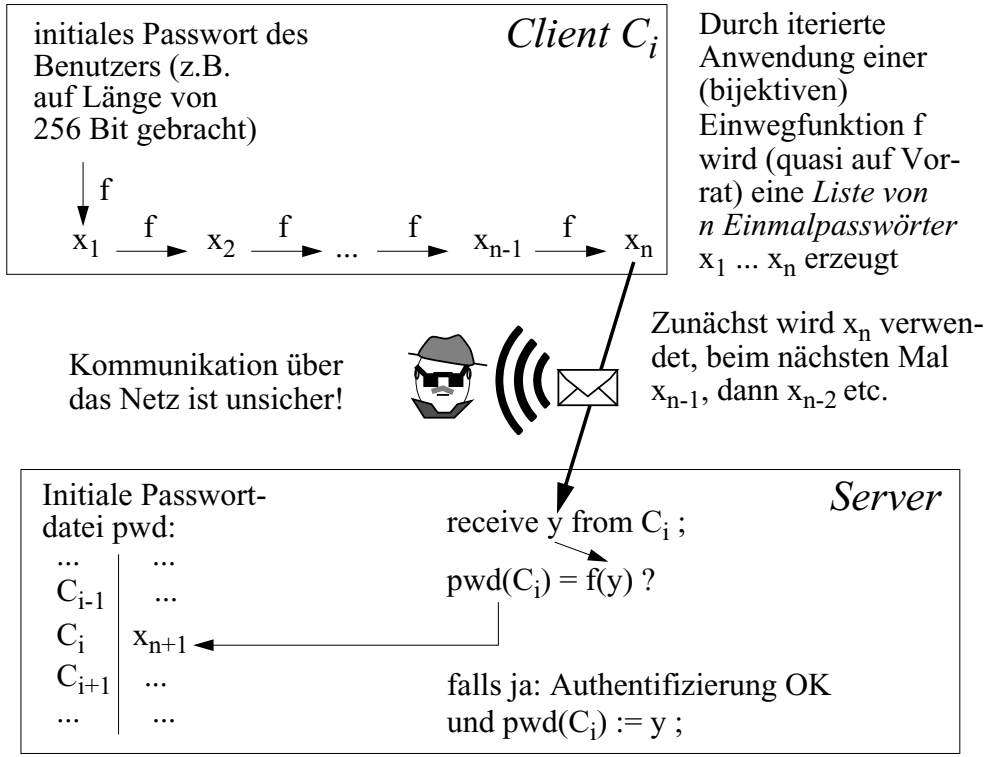
- Einwegfunktionen erscheinen zunächst ziemlich sinnlos: Ein zu y = f(x) verschlüsselter Text x kann nie wieder entschlüsselt werden!

  ⇒ 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!)

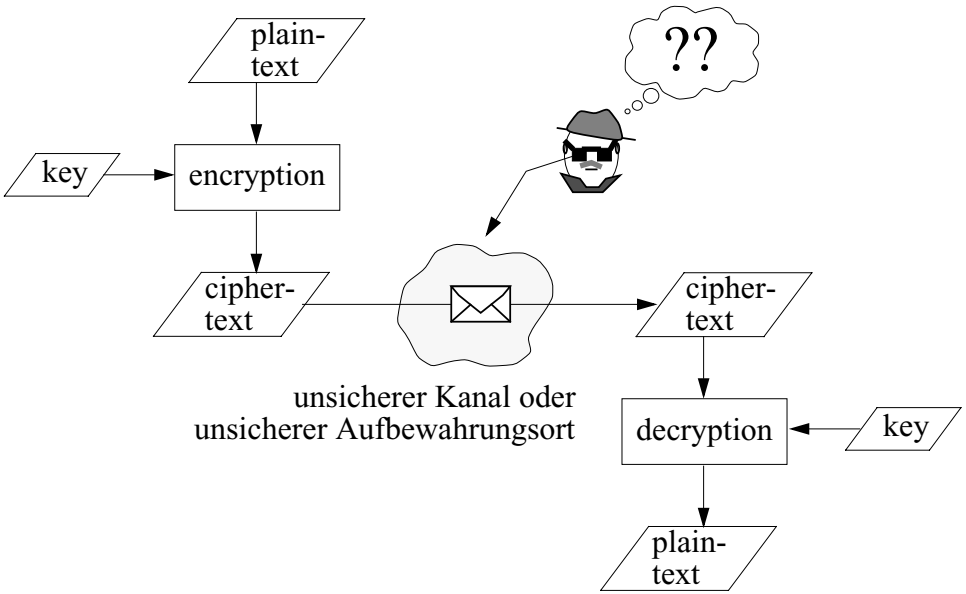
Einmalpasswörter mit Einwegfunktionen

- 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	C0	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”)

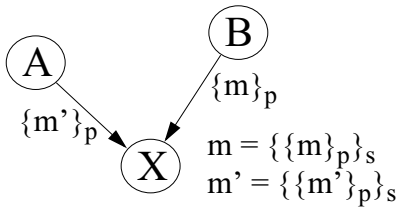
Asymmetrische Kryptosysteme

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

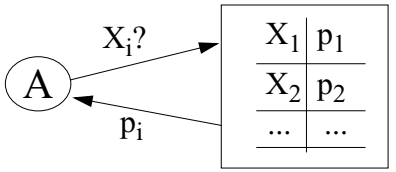
- Schlüssel zum Ver- / Entschlüsseln sind *verschieden*
  - z.B. *RSA-Verfahren* (Rivest, Shamir, Adleman, 1978), beruht auf der Schwierigkeit von Faktorisierung
  - andere Verfahren beruhen z.B. auf diskreten Logarithmen
- Für jeden Prozess X existiert ein Paar (p,s)

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

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

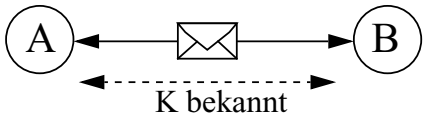
- Vorteil gegenüber symmetrischen Verfahren: vereinfachter Schlüsselaustausch
  - jeder darf den übermittelten Verschlüsselungsschlüssel p mithören
  - Entschlüsselungsschlüssel s braucht grundsätzlich nie mitgeteilt zu werden
  - bei n Teilnehmern genügen 2n Schlüssel (statt $O(n^2)$ wie etwa bei DES)

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

- *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?)

sA bzw. pA secret bzw. public key von A

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
B: überprüfe, ob $\{m\}_K$ = m'

Damit B den richtigen Schlüssel (für A) wählt

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

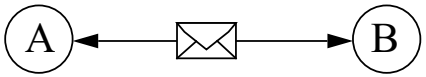
2. Verfahren:

A→B: "Ich bin A"
B→A: n
A: n' := $\{n\}_K$
A→B: n'
B: überprüfe, ob $\{n\}_K$ = n'

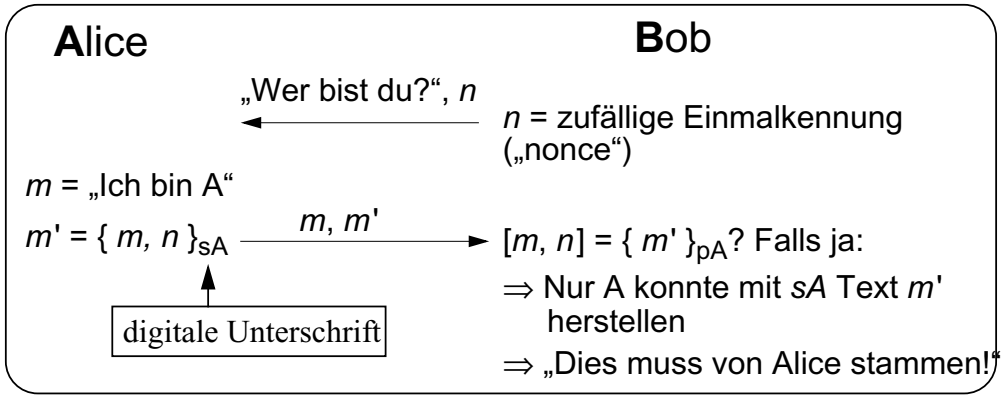
Einmalkennung ("nonce")

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

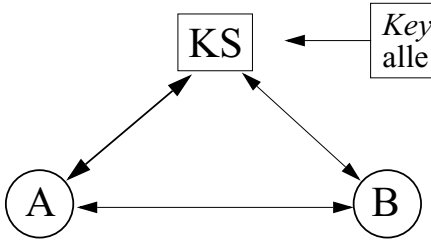
Authentifizierung mit asymmetrischen Schlüsseln



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



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



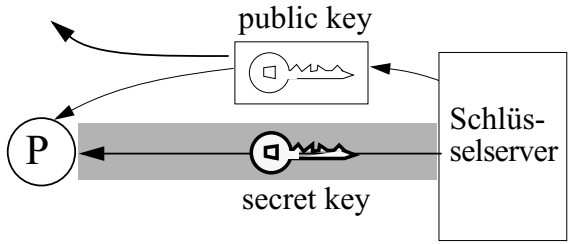
Key Server: Kennt alle public keys

- 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

Schlüsselvergabe

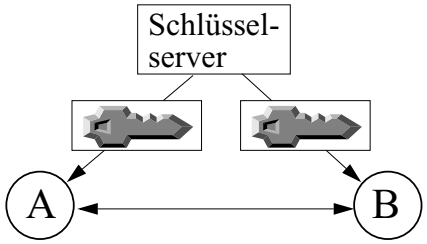
- Zur Vergabe eines Paares von public-, secret-keys:



- 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”)

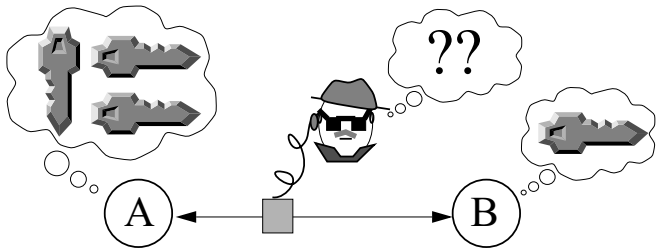
→ erhöhte Sicherheit gegen Angreifer



- 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

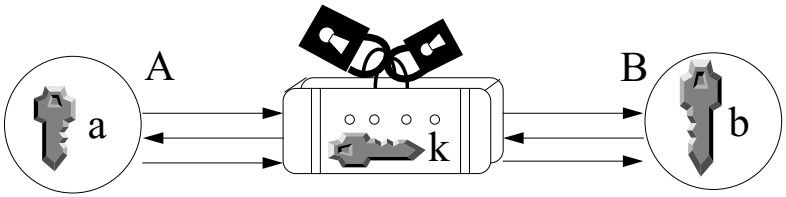


- Problem: A und B wollen sich über einen unsicheren Kanal auf einen gemeinsamen Schlüssel einigen, ohne einen Schlüsselserver zu verwenden

- 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

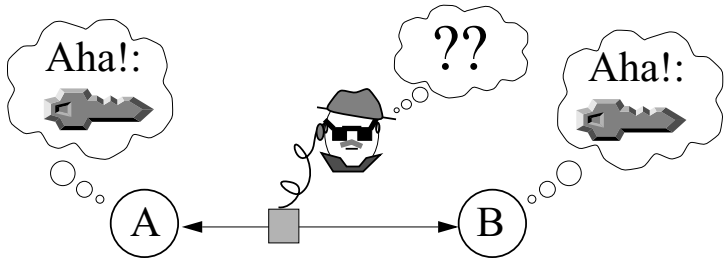
Beachte: k wird nie offen transportiert!

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*: $f(x) = c^x \bmod p$

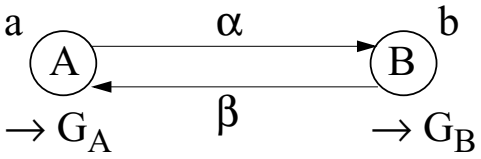
  ($1 < c < p$; i.a. ist p eine grosse Primzahl)

  - 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):

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

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

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

$G_A = G_B$

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...

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

Bemerkungen:

- Lässt sich auch auf $k > 2$ Benutzer verallgemeinern
- Der Algorithmus (entdeckt ‘76) ist patentiert
  - U.S.-Patent Nummer 4200770 (Sept. ‘77)

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!

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).* Im Internet: ftp://ftp.isi.edu/in-notes/rfc1510.txt oder http://ds0.internic.net/rfc/rfc1510.txt

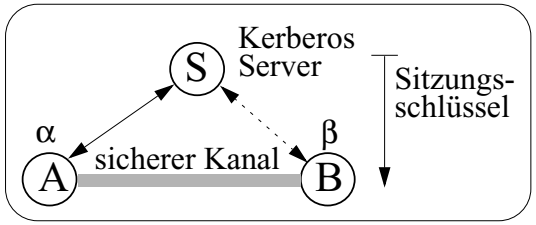
Kerberos-"Philosophie"

- 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

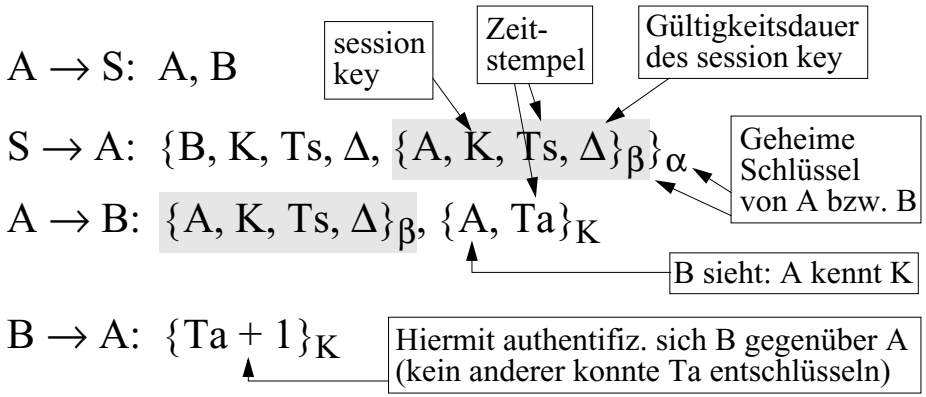
  "Kerberos-Server"
- Alle Benutzer, Clients und Server sind bei zentraler Instanz (Key Distribution Center: "KDC") akkreditiert
  - vereinbaren mit dem KDC auch ihren Geheimschlüssel ("master key")
  - ohne Akkreditierung keine Server-Berechtigungsscheine ("Tickets")
  - ohne Tickets kein Service
  - 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

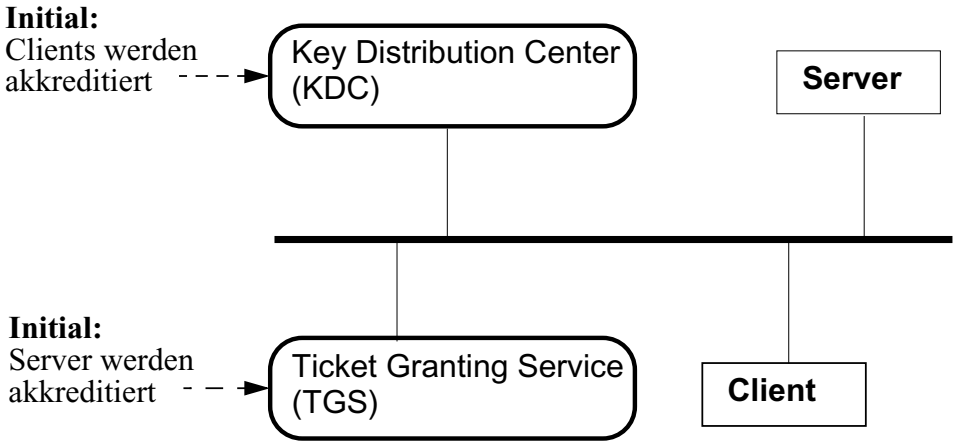


Hier: Version 4 (MIT-Version eingefroren Dez. '92); spätere Versionen im Prinzip nur leicht unterschiedlich



- 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

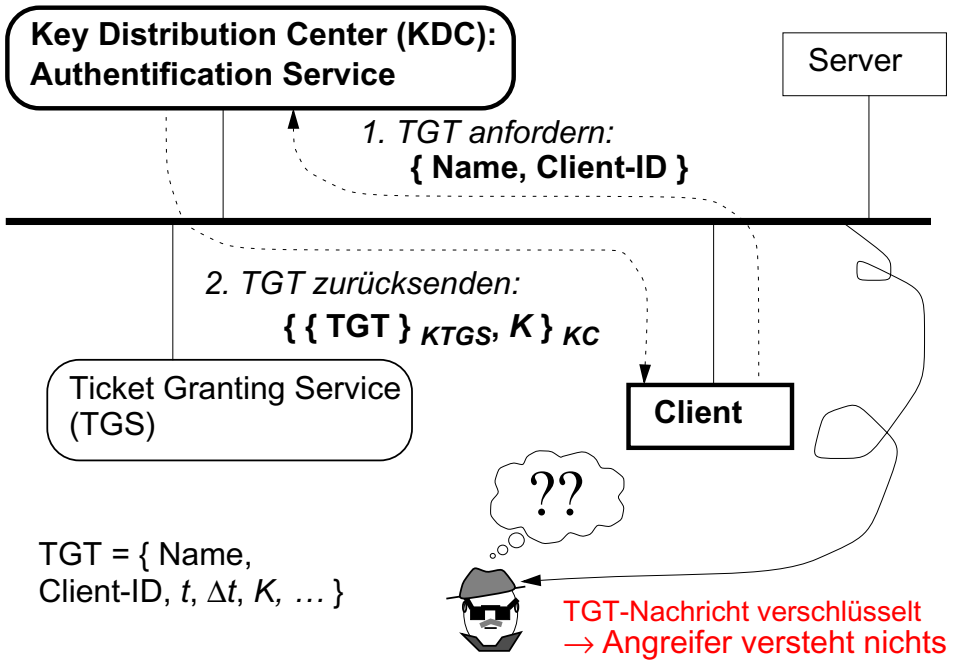
Kerberos: Akkreditierung



- 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

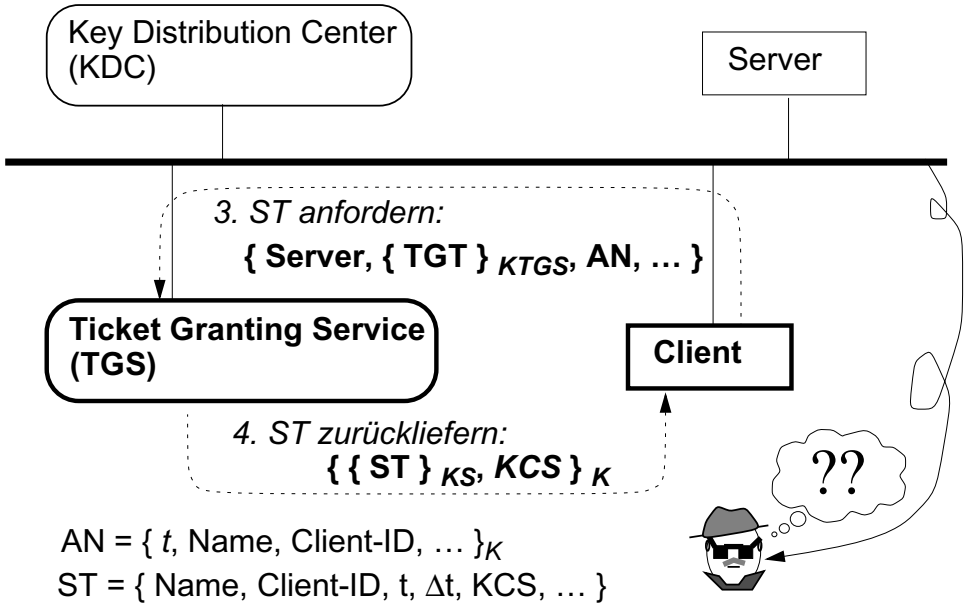
→ KDC-Nachricht ist authentisch: Nur KDC kennt noch Schlüssel KC !

→ 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

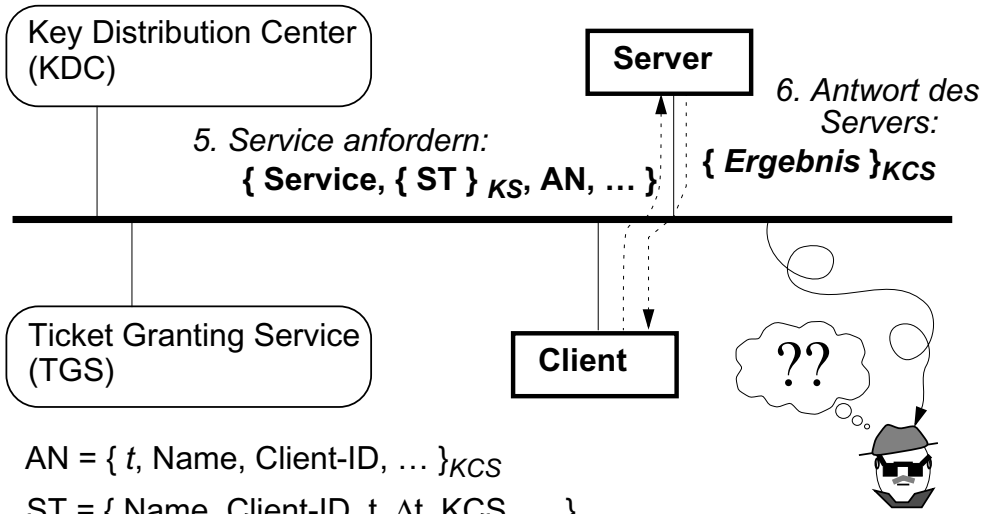


- Client: erstellt Authentizitätsnachweis AN = { t, Name, Client-ID, … } $_{K}$
- Client sendet an TGS { Server, { TGT } $_{KTGS}$, AN, … } als Request
- TGS: entschlüsselt TGT mit Schlüssel KTGS, erhält damit K;
  entschlüsselt AN mit K, vergleicht Inhalt mit TGT;
  erstellt Service Ticket ST = { Name, 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

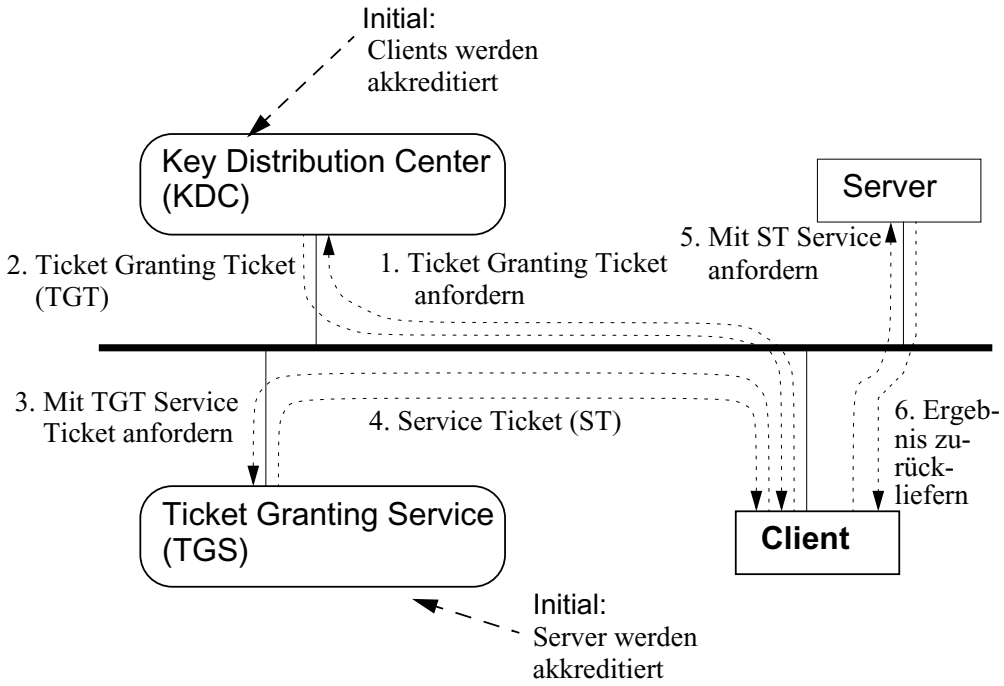


- Client: erstellt Authentizitätsnachweis AN = { t, Name, Client-ID, … }$_{KCS}$
- Client an Server: sendet { Service, { ST } $_{KS}$, AN, … } als Service Request
- Server: entschlüsselt ST mit KS, erhält damit KCS;
  entschlüsselt AN mit KCS, vergleicht Inhalt mit ST;
  leistet Service und erzeugt Ergebnisdaten
- 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...)
  - Software zur Replikation der Datenbank (Verteilung ist wichtig, da bei Ausfall des KDC fast nichts mehr im ganzen Netz geht!)
- Version 5 (inkompatibel zu Version 4!): mehr Funktionalität und allgemeiner verwendbar, z.B.:
  - Datenformate mit ASN.1 - Basic Encoding Rules
  - 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

Schlüsselgenerierung

- Der Schlüsselgenerierungsalgorithmus von Kerberos (z.B. für TGT oder session keys) funktionierte ursprünglich so:

das ist gelogen!

bitweise xor

```
p = getpid() ^ gethostid ;
gettimeofday(&time, (struct timezone *) 0) ;
/* randomize start */
srandom(time.tv_usec ^ time.tv_sec ^ p ^ n++)
```

Initialisierung für eine (deterministischen) Folge von "random"-Werten

- Startwert des Zufallszahlengenerators ("seed")
- gettimeofday liefert auf "time" zwei Werte zurück:
  tv_sec: Sekunden seit dem 01.01.1970
  tv_usec: Mikrosekunden...

- Lässt sich aus einem Schlüssel der folgende berechnen?
  - p ist eine "Konstante"
  - time of day ist (ungefähr) bekannt
  - n++ unterscheidet sich i.a. nur in wenigen Bits von n

- Könnte jemand vielleicht absichtlich die Uhr des Servers auf einen falschen (d.h. bekannten) Wert synchronisieren?
  - welche Granularität hat eigentlich die Uhr?

- Angeblich soll nun der Algorithmus "verbessert" sein...

- Und die Moral der Geschichte?

Resümee (1)

- Institut für Pervasive Computing
- Gruppe für verteilte Systeme
- Organisatorisches zur Vorlesung

- Einordnung der Vorlesung
- Verteilte Systeme: Begriff, Sichtweisen, Eigenschaften...
- Motivation; Gründe für verteilte Systeme
  - Kooperation von geographisch verteilten Einheiten
  - qualitatives Wachstum des Internet
  - Beispiel-Anwendungszenarin
- Middleware für das Internet

Resümee (2)

- Transparenzeigenschaften (Verbergen von Verteiltheit etc.)
- Charakteristika und praktische Problembereiche verteilter Systeme
- Historische Entwicklung von Systemen und Konzepten
- Phänomene und konzeptionelle Probleme
  - Schnappschussproblem (inkonsistente globale Sicht)
  - Phantom-Deadlocks
  - Uhrensynchronisation
  - kausal inkonsistente Beobachtungen
  - Geheimnisaustausch über unsicheren Kanal

- Multiprozessoren (gemeinsamer Speicher)
  - Buskoppelung
  - Schaltnetzkoppelung (Crossbar, Permutationsnetze)
- Multicomputer (verteilter Speicher)
  - Transputer
  - Bewertungskriterien für Verbindungstopologien
  - Hypercube (rekursives Konstruktionsprinzip)
- Mehrrechner-Verbindungstopologien
  - Torus
  - Cube Connected Cycle
  - Zufallstopologien

Resümee (3)

- Mehrrechner-Verbindungstopologien
  - Torus
  - Cube Connected Cycle
  - Zufallstopologien
- Nachrichtenkommunikation
  - Message-passing-Systeme / -Bibliotheken
  - Prioritäten von Nachrichten
  - Zuverlässigkeitsgrade
- Fehlermodelle
  - fehlerhaftes Senden / Empfangen
  - Verlust von Nachrichten
  - crash, fail-stop
  - byzantinische Fehler

Resümee (4a)

- Kommunikationsmuster
  - Mitteilung ↔ Auftrag
  - synchron ↔ asynchron
- Synchrone Kommunikation
  - Definition
  - Realisierung
  - virtuelle Gleichzeitigkeit; Gummibandtransformation
  - Blockaden und Deadlocks
- Asynchrone Kommunikation
  - Vor- / Nachteile gegenüber synchroner Kommunikation

Resümee (4b)

- Synchrone Kommunikation mit asynchroner simulieren
  - Warten auf ein explizites Acknowledgement
- Asynchrone Kommunikation mit synchroner simulieren
  - Puffer(prozess!) zur Entkoppelung dazwischenschalten
- Implementierung von Pufferprozessen
  - durch Inversion der Kommunikationsbeziehung
- Puffer beschränkter Kapazität
  - Implementierungsaspekte
- Alternatives Empfangen von Nachrichten
  - “select”-Anweisung: elegantes und mächtiges Konstrukt
  - aber: Semantik genau festlegen
- Verschiedene Kommunikationsmuster
  - no-wait-send; RPC; asyn. RPC; rendezvous
- Datagramm
- Rendezvous-Protokoll
- RPC
  - Implementierung
  - Parameter-Marshalling
  - Stubs
  - Transparenzproblematik

Resümee (5)

- RPC-Fehlerproblematik
  - Fehlerursachen (verlorene Nachrichten, Crash von Server / Client)
  - Fehlersemantik (maybe, at-least-once, at-most-once, exactly once)
  - Orphans
- RPC
  - Binding
  - Protokolle
  - Effizienz
  - asynchroner RPC
- Socket-Programmierschnittstelle

Resümee (6)

- Socket-Programmierschnittstelle
  - Client-Server-Beispiel in C
  - Sockets in Java (in den Übungen)
- Java als “Internet-Programmiersprache”
- Adressierungsarten
  - 1:1, direct naming
  - m:n, mailbox
  - n:1, port
  - Kanäle
- Empfangen von Nachrichten
  - non-blocking (→ aktives Warten)
  - Zeitüberwachung
  - selektives Empfangen
  - implizites Empfangen
- Sprachaspekte beim verteilten Programmieren
  - kommunizierbare Datentypen?

Resümee (7)

- Gruppenkommunikation (Broadcast / Multicast)
  - Anwendungen
  - idealisierte Sicht
  - Fehlerproblematik
  - Zuverlässigkeitsgrad (“best effort”, k-zuverlässig)
  - “reliable Broadcast” mit ACK, NACK
- Algorithmus für “reliable Broadcast”
- FIFO-Broadcasts
  - zwei nacheinander ausgeführte Broadcasts ein und desselben Senders erreichen alle Empfänger in dieser Reihenfolge
  - nicht stark genug, um akausale Beobachtungen zu verhindern
- Kausale Broadcasts
  - kausale Abhängigkeit zweier Nachrichten
  - “Causal Order”: Nachrichtenempfang “respektiert” kausale Abhängigkeit von Nachrichten (“kausaltreu”)
- Atomare Broadcasts
  - logisch gleichzeitiger Empfang der Einzelnachrichten eines Broadcasts

Resümee (8)

- Atomare Broadcasts
  - logisch gleichzeitiger Empfang der Einzelnachrichten eines Broadcasts
  - Realisierung über zentralen Sequencer bzw. Token auf einem logischen Ring
- Kausal atomare Broadcasts
  - virtuelle Synchronität
- Multicast
  - Zweck
  - Adressierung von Multicast-Gruppen
- Gruppenüberlappung
  - lokale / globale Gültigkeit von Reihefolgebedingungen etc.
- Multicast: Membership-Problem
  - atomare Änderung der Gruppenzugehörigkeit
  - Tolerieren von Prozessausfällen

Resümee (9)

- Push-Prinzip und Publish & Subscribe

- Ereigniskanäle als “Softwarebus”

- Tupelräume
  - Linda-Modell
  - JavaSpaces

- Logische Zeit
  - Raum-Zeitdiagramme, Ereignisse
  - Zeitstempel von Ereignissen
  - Uhrenbedingung. kausale Unabhängigkeit

- Logische Uhren von Lamport
  - Definition
  - Realisierung
  - injektive Abbildung, eindeutige Zeitpunkte

- Wechselseitiger Ausschluss (mit logischer Zeit)
  - replizierte Warteschlangen von Lamport (request, reply, ack)

Resümee (10)

- Wechselseitiger Ausschluss (mit logischer Zeit)
  - Lösung von Ricart / Agrawala 1981
  - safety, liveness, fairness?
  - sind 2(n-1) Nachrichten optimal
    (bei symmetrischen, dezentalen Strukturen)?

- Namensverwaltung
  - Zweck von Namen
  - Namen und Adressen
  - Binden
  - Namenskontexte, hierarchische Namensräume
  - Aufgaben einer Namensverwaltung
  - Namensverwaltung in verteilten Systemen

- Nameserver
  - Replikation und Caching

- Zufall
  - Symmetrisierungstrick von J. v. Neumann

Resümee (11)

- Internet Domain Name Service (DNS)
  - Namensauflösung im Internet
  - resource records
  - nslookup (bzw. “dig” oder “host”)

- Client-Server-Modell (<==> Peer-to-Peer-Strukturen)
  - Prinzip
  - Client/Server-*Maschinen*
  - Client/Server-*Rollen*

- Zustandsändernde / -invariante Dienste und Server
  - idempotente und wiederholbare Aufträge
  - stateless / statefull

- Zustandsändernde / -invariante Dienste und Server
  - Beispiel WWW-Server
  - cookies

- Konkurrente Server
  - dynamische / statische Handler-Prozesse (“slaves”)
  - quasi-konkurrente Server (internes Multiplexen)

- X-Window als “klassisches” Client-Server-System
  - aber: events zur asynchronen Rückmeldung Server → Client

- Servergruppen / verteilte Server
  - Strukturen kooperierender Server
  - Server-Auswahl bei einem Lastverbund
  - Replikation von Servern (“Überlebensverbund”)

Resümee (12)

- Middleware: Der Weg zum “Netzwerk als Computer”

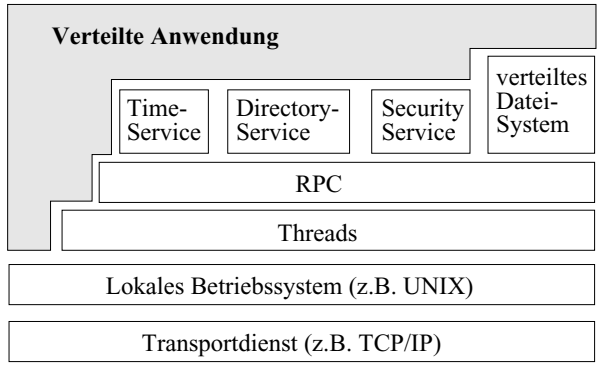
- Sun-RPC
  - Identifikation entfernter Prozeduren (host, Programm-, Version-, Prozedur-Nummer)
  - Registrieren eines Dienstes auf Serverseite

- Portmapper
  - Zuordnung Port / Programmnummer eines Dienstes

- Sun-RPC: rpcgen
  - Generieren von Prozedurstubs und Serverskelett aus Schnittstellenspezif.

- Schutzaspekte bei Sun-RPC
  - “UNIX flavor”: Automatisches Mitsenden von Benutzerkennung etc.
  - “Secure RPC”: Authentifizierung und Verschlüsselung (DES bzw. Kerberos)

- DCE
  - Hauptkomponenten
  - Zellenarchitektur
  - Threads (Problematik)
  - RPC (Unterschied zu Sun-RPC; Fehlersemantik; Bindevorgang)
  - Sicherheitsaspekte



Resümee (13)

- CORBA
  - CORBA-Architektur
  - Object Services und Common Facilities
  - neuere Erweiterungen bei CORBA
- Jini
  - Motivation: Dienstgparadigma, Netzzentrierung,...
  - Java-Bezug
  - Lookup-Service
  - Discovery
  - Join
  - Proxies und smart Proxies
  - Code-Mobilität
  - Leases
  - verteilte Ereignisse
  - Vorteile und Probleme von Jini

Resümee (14)

- Sicherheit in verteilten Systemen: Anforderungen
- Einmalpasswörter mit Einwegfunktionen
- One-time-Pads mit XOR
- Symmetrische und asymmetrische Kryptosysteme
- Authentifizierung mit sym. und asym. Schlüsseln
- “Geheime” Schlüsselvergabe
  - Schlüsselaustausch mit Diffie-Hellman-Prinzip
- Kerberos
  - Protokoll für Ticket-Granting-Ticket- und Service-Ticket-Erwerb
  - Anwendungsbeispiel: Einrichtung sicherer Kanäle
  - Sicherheitsaspekte