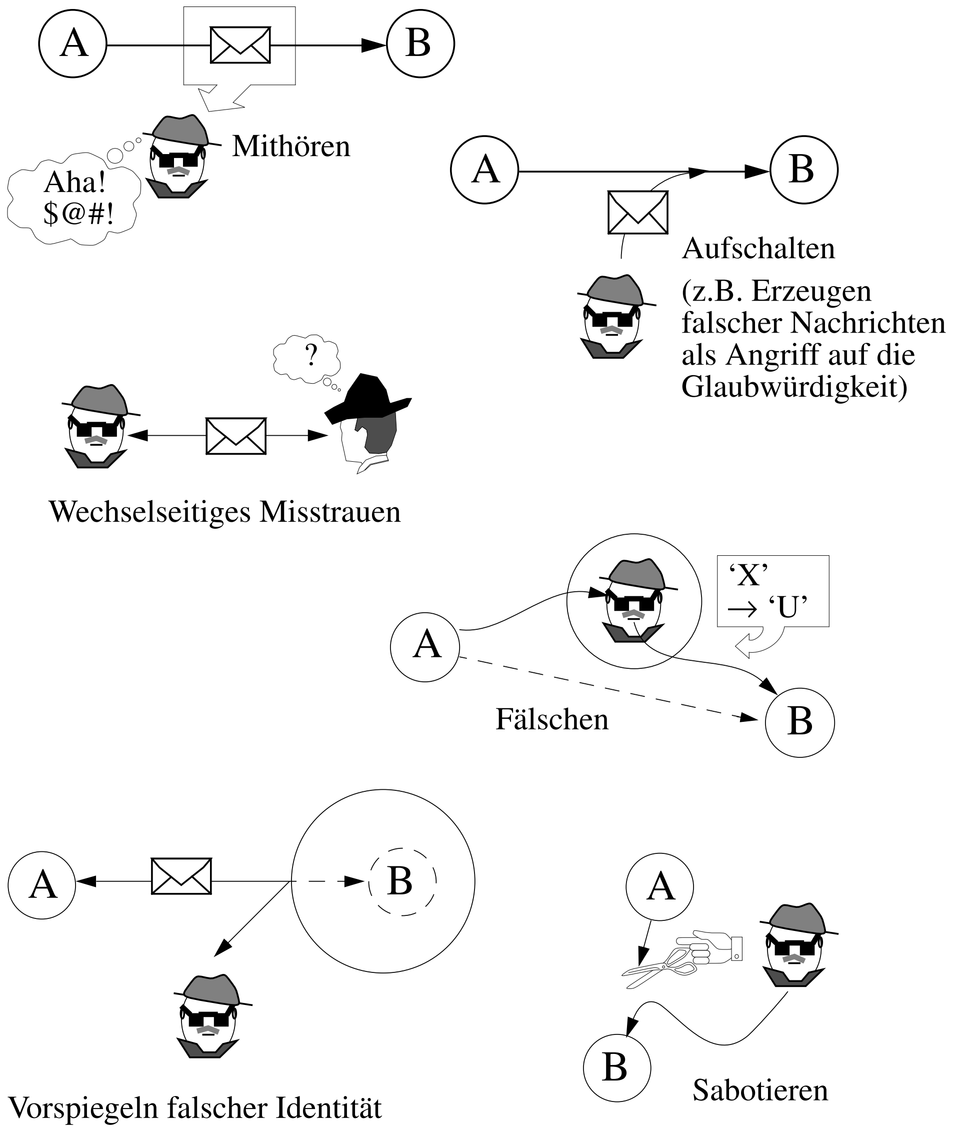
Sicherheit in verteilten Systemen



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
  - 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. LANs mit Bustopologie

- *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!

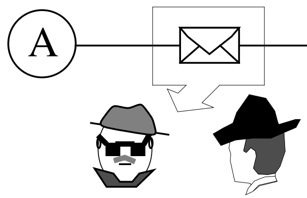
Typische Techniken und “Sicherheitsdienste”:

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

Hierfür Kryptosysteme und Protokolle als “Security Service”, z.B. *Kerberos*

Angriffsformen

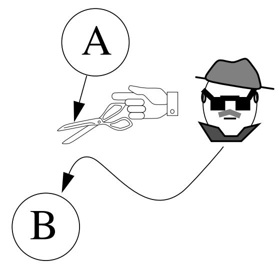
- *Passive Angriffe*: Beobachten der Kommunikation
    - Inhalt von Nachrichten in Erfahrung bringen (“eavesdropping”)
    - Kommunikationsverhalten analysieren (“wer mit wem wie oft?”)



--> 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



Autorisierung / Schutzmatrix

- Erteilen von *Rechten* (typischerweise an Clients)
  - *wer* erteilte eigentlich Rechte?
- Überprüfen der Rechte notwendig
  - Recht durchsetzen: Rechtsbrüche verhindern!
- Rechte müssen (sicher) gespeichert werden, dazu *Schutzmatrix* als konzeptuelles Gebilde
  - verknüpft Subjekte (Clients) mit Objekten (Server)

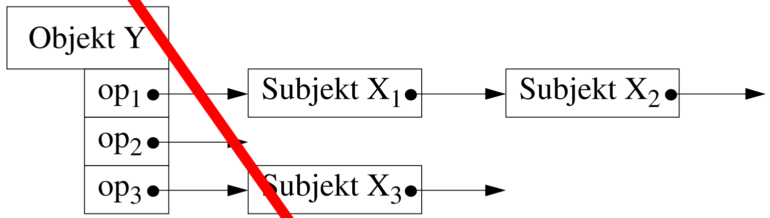
Objekte (Spalten), *Subjekte* (Zeilen)

	Y_1	Y_2	Y_3	Y_4	Y_5
X_1				Op_4	
X_2		Op_1			
X_3			Op_1 Op_2		
X_4					

  - "X darf auf Y Operation op_i ausführen" <--> $op_i \in M(X,Y)$
  - typ. Rechte: Lesen, Vergleichen, Löschen, Ändern, Rechtevergabe...
- *Schutzmatrix* i.a. dünn besetzt
  - Idee: nicht-leere Einträge spalten- oder zeilenweise speichern
  - lässt sich damit auch verteilt implementieren!
  - führt zu *Zugriffskontroll-Listen* einerseits und *Capabilites* andererseits

Zugriffskontroll-Listen

- In jedem Objekt (= Server) werden zu jeder Operation die hierfür berechtigten Subjekte (= Clients) genannt
  - Liste mit möglichen Clients mit jeweils zuerkannten Rechten
  - "Türstehenprinzip": vgl. z.B. UNIX-Dateischutzattribute



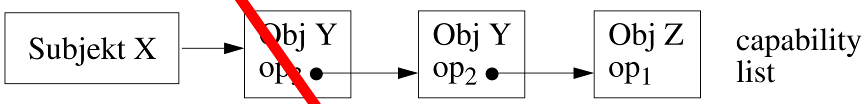
  - Subjekt X_1 und Subjekt X_2 dürfen op_1 auf Objekt Y ausführen; Subjekt X_3 darf Operation op_3 auf Objekt Y ausführen
- Wie kann ein Objekt prüfen, ob ein Subjekt nicht die falsche Identität vorgibt? (--> Authentizitätsproblem)
- Zugriffskontroll-Listen müssen gegen Manipulation geschützt werden
  - u.a. Server logisch und physisch schützen

Capabilities

- Bei den Subjekten (= Clients) angesiedelt; nennen Objekte (= Server) und spezifische Operationen (Dienste, Teildienste), die von diesen in Anspruch genommen werden dürfen

  - "Eisenbahnticket-Prinzip": Der Inhaber der Berechtigung darf den angegebenen Dienst des genannten Servers in Anspruch nehmen

  - Beispiele: Zugangspasswörter; "magic cookies"



  - Subjekt X darf auf Objekt Y die Operationen op_2, op_3 ausführen und auf Objekt Z die Operation op_1 ausführen

- Vorteile von Capabilities:

  - einfache Übertragbarkeit von Rechten und Zugriffsmöglichkeiten
  - einfaches Authentifizierungsschema inbegriffen: "Wer immer die Capability vorweisen kann, gilt als berechtigt"

- Probleme von Capabilities: Schutz vor "Fälschung" (verbotswidriges Kopieren, Erraten des Bitmusters etc.)

  - langes Bitmuster aus dünn besetztem Musterraum wählen --> zufälliges Raten einer gültigen Capability nahezu unmöglich

  - ggf. Seriennummer, Verfallsdatum, Inhaber (Client) etc. einkodieren
    --> entwendete / kopierte Capability nur zeitlich begrenzt verwendbar
    --> Kopien anhand eindeutiger Seriennummer ggf. erkennbar etc.

  - vgl. analoge Lösungsansätze bei Kreditkarten

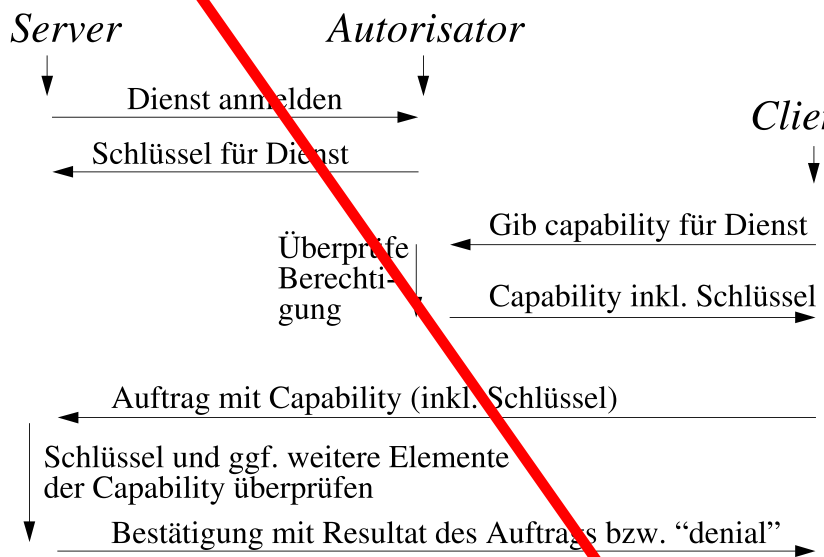
  - Bem.: gleiche Problematik bei elektronischem Geld! (Geld = "universelle" Capability...)

Capabilities (2)

- Prinzipielle Anwendungsmöglichkeit: ← Sicherheitslücken: Authentifizierung und Verschlüsselung notw.!

  - hier: Zugriffsschlüssel als eigenständige Komponente einer Capability

  - Capability enthält typischerweise u.a. Serveradresse, Port-Nummer etc.



- Beachte: Korrekte Autorisierung von sicherer Authentifizierung abhängig!

  - wie verhindert der Autorisator das Vorspiegeln falscher Identität?

- Server kann Dienstangebot zurückziehen: Alte capability ungültig machen; ggf. neue einrichten

- Capabilities können auch für langlebige *Objekte* (z.B. Dateien...) eingerichtet werden, die von Servern verwaltet werden

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
  - 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

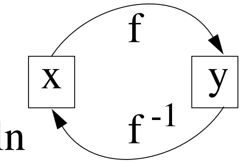
    beachte aber Crack-Programme
  - Abhörgefahr (kein Passwortaustausch im Klartext; Speicherung des Passworts nur in codierter Form, so dass Invertierung prakt. unmöglich
  - Replay-Attacke (Gegenmassnahme: Einmalpasswörter)

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

- 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)

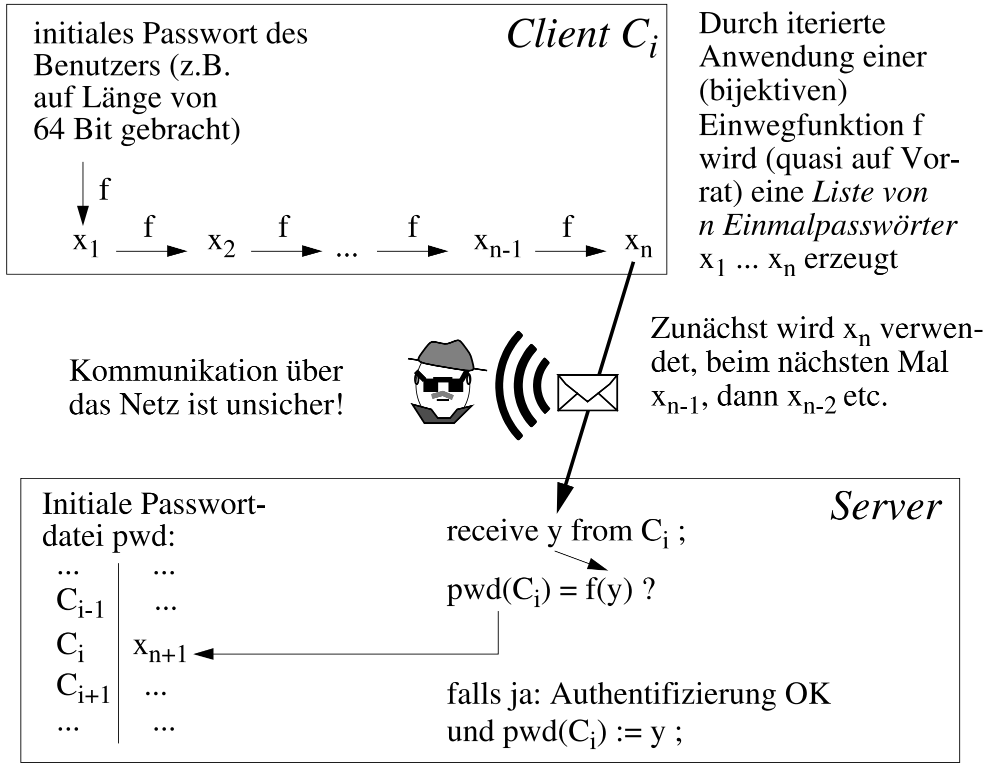
- Idee: Nur der “Besitzer” oder “Erfinder” von f kennt dieses
- Beispiel Briefkasten: Einfach etwas hineinzutun; schwierig etwas herauszuholen; mit Schlüssel (= Geheimnis) ist das aber einfach!
- Anwendung z.B.: Public key-Verschlüsselung

- 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

Einmalpasswörter mit S/KEY

"Request For Comments"

Auszug aus RFC 1760 (ftp://ds.internic.net/rfc/rfc1760.txt)

RFC 1760 The S/KEY One-Time Password System February 1995

Abstract

This document describes the S/KEY One-Time Password system...

Overview

One form of attack on computing system connected to the Internet is eavesdropping on network connections to obtain login id's and passwords of legitimate users. The captured login id and password are, at a later time, used gain access to the system. The S/KEY One-Time Password system is designed to counter this type of attack, called a replay attack.

With the S/KEY system, only a single use password ever crosses the network...

The S/KEY system one-time passwords are 64 bits in length. This is believed to be long enough to be secure and short enough to be manually entered ... when necessary...

Entering a 64 bit number is a difficult and error prone process. Some S/KEY system one-time password calculator programs insert this password into the input stream, others make it available for system cut and paste. Some arrangements require the one-time password to be entered manually. The S/KEY system is designed to facilitate this manual entry without impeding automatic methods. The one-time password is therefore converted to, and accepted as, a sequence of six short (1 to 4 letter) English words. Each word is chosen from a dictionary of 2048 words...

Because the number of hash function applications executed by the client decreases by one each time, at some point the user must reinitialize the system of be unable to login again. This is done by using the keyinit command...

The most basic calculator is the key command whose format is:

```
key [-n count] sequence seed
```

The optional count is used to display more than a single one time password. This is useful to create a paper list of one time passwords.

The most automated calculator is the termkey program that runs as a Terminate and Stay Resident (TSR) program on a PC.

...

Acknowledgements

The idea behind S/KEY authentication was first proposed by Leslie Lamport [1].

References

[1] Lamport, L., "Password Authentication with Insecure Communication", Communications of the ACM 24.11, November 1981, 770-772

NAME **key** - compute responses to S/Key challenges.
DESCRIPTION
Takes an S/Key sequence number and seed as command-line arguments, prompts for the user's secret password, and formats the response as English words.
OPTIONS
-n count: the number of one-time access passwords to print.

```
[~] key -n 10 70 1
Reminder - Do not use this program while logged in via telnet or rlogin.
Enter secret password:

61: MILL SHOW OILY LINT AVIS SEAM
62: YET BED FARM MIST CODY MALL
63: JAKE ROME TORN SILO YARN BIND
64: FUNK SHAY NIL WAD FRAU CHIC
65: DADE CHEW SON YOKE GWYN SAW
66: LOSS RAID SIN SHOD FAIN MOS
67: TWIT HALO FLY TAP TOWN TINT
68: MULL YELL FAIL BAH SOOT HORN
69: APS SKIN OHIO HAIR EVIL GETS
70: ONTO FEED COOK LOST DEN NAIL
```

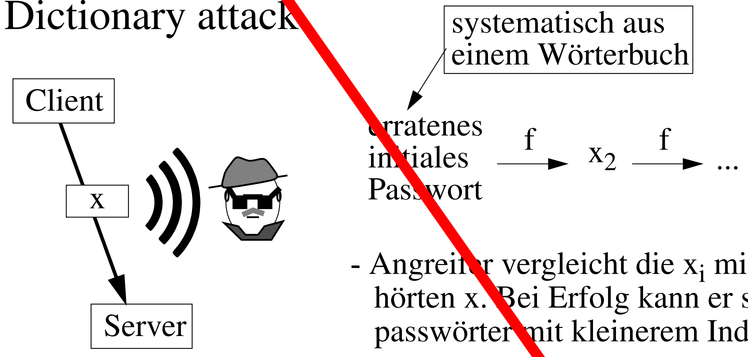
NAME **keyinfo** - display current S/Key sequence number and seed.
DESCRIPTION
keyinfo takes an optional user name and displays the user' s current sequence number and seed found in the S/Key database. The command can be useful when generating a list of passwords for use on a field trip.

Wie sicher sind Einmalpasswörter?

- Abhören des initialen ("secret") Passwortes

"Of course, if you don't take the minimal precautions (run key on a distant machine or run xskey on a display different than [unix]:0), then your secret password will be present on the network and thus could be seen by anyone and used to produce a one time password by someone other than you. In this case, your account becomes as weak as before."

- Dictionary attack



- Angreifer vergleicht die x_i mit dem abgehörten x. Bei Erfolg kann er selbst Einmalpasswörter mit kleinerem Index berechnen!

- Spoofing
  - Der Angreifer spielt Server und fragt den Client nach dem i-ten Passwort x_i, wobei i kleiner als der Index j des vom echten Server erwarteten nächsten Passworts x_j ist. Aus x_i berechnet er dann durch (ggf. mehrfache) Anwendung von f das nächste Passwort x_j.

- Es gibt noch mehr Angriffsmethoden
  - "race attacks" (konkurrentes login: Zähler ggf. noch nicht erniedrigt)
  - "hi-jacking attacks" (Verbindung nach login des Clients kapern)
  - ...

Einweg-Hashfunktion

- Andere Bezeichnungen:
  - fingerprint
  - message digest
  - cryptographic checksum
  - authentication code

Einweg-Hashfunktion

y = h(M)

Nachricht

Fingerprint von M

- *Eigenschaften:*
  - y=h(M) steht in keinem erkennbaren Zusammenhang zu M
  - y=h(M) ist i.a. wesentlich kürzer als die beliebig lange Nachricht M ==> h ist nicht injektiv!
  - typische Längen für y: 128 Bit oder mehr ==> sehr geringe Wahrscheinlichkeit, dass h(M) = h(M') für zwei M ≠ M'; aber Vorsicht:
    - Denkübung: Wie gering? Beachte das Geburtstagsparadoxon!
    - es gibt unendlich viele M ≠ M' mit h(M) = h(M')

- *Zweck:* Falls h(M) = h(M'), dann ist aller Wahrscheinlichkeit nach M = M' (aber keine Gewissheit!)

- "*Einweg*": y = h(M) ist effizient berechenbar, aber es ist nahezu unmöglich, ein m (etwa auch M selbst!) zu generieren, so dass h(m) = y ist!

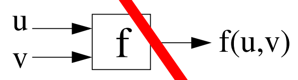
- *Ferner:* Aus M ist es sehr schwierig (z.B. mehr als 2^{64} Operationen nötig), ein m mit h(m) = h(M) zu finden

- Beispiel für *Anwendungen:*
  1) Y beweist, dass es eine Kopie einer Datei hat, indem es an X den Fingerprint der Datei sendet, den X mit seinem vergleichen kann
  2) Fingerprint als Schutzmassnahme gegem Nachrichtenverfälschung
  3) Digitale Unterschrift nur auf den Fingerprint anwenden (effizienter!)
  3) Hinterlege h(M) einer Erfindung M beim Patentamt: Priorität ist später jederzeit beweisbar, ohne M sofort zu offenbaren

Einweg-Hashfunktionen

- Konstruktion von Einweg-Hashfunktionen, die beliebig lange Argumente zulassen:

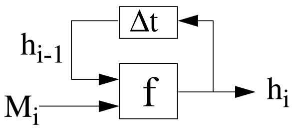
Benutze eine zweistellige Einwegfunktion f



mit |u| = |v| = |f(u,v)|

f: Typischerweise Vertauschen, shiften, xor einzelner Bits der Eingaben u, v in mehreren Runden.

Damit nun:



mit einzelnen Blöcken M_i, h_i, z.B. der Länge 128 Bit

Ausgabe von f wird zur Eingabe rückgekoppelt

Nachricht M wird blockweise von f verarbeitet

h_0 ist entweder eine Konstante oder variabel (d.h. ein Schlüssel)

- Es gibt eine Reihe von veröffentlichten Einweg-Hashfunktionen, die (anscheinend) von guter Qualität sind:
  - SHA ("Secure Hash Algorithm", NIST / NSA)
  - MD5 ("Message Digest"; RFC 1321; ca. 250 Zeilen Code, wird z.B. bei PGP ("Pretty Good Privacy") eingesetzt)

MD5 (RFC 1321)

R. Rivest
Laboratory for Computer Science
and RSA Data Security, Inc.

April 1992

This document describes the MD5 message-digest algorithm. The algorithm takes as input a message of arbitrary length and produces as output a 128-bit “fingerprint” or “message digest” of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given prespecified target message digest. The MD5 algorithm is intended for digital signature applications, where a large file must be “compressed” in a secure manner before being encrypted with a private (secret) key under a public-key cryptosystem such as RSA.

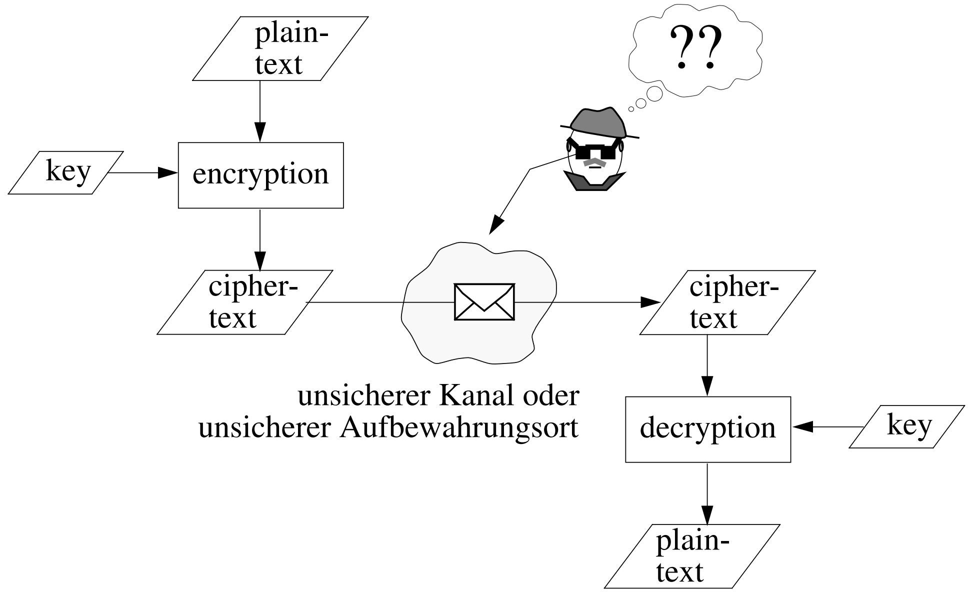
The MD5 algorithm is designed to be quite fast on 32-bit machines. In addition, the MD5 algorithm does not require any large substitution tables; the algorithm can be coded quite compactly.

The MD5 algorithm is an extension of the MD4 message-digest algorithm. MD5 is slightly slower than MD4, but is more “conservative” in design. MD5 was designed because it was felt that MD4 was perhaps being adopted for use more quickly than justified by the existing critical review; because MD4 was designed to be exceptionally fast, it is “at the edge” in terms of risking successful cryptanalytic attack. MD5 backs off a bit, giving up a little in speed for a much greater likelihood of ultimate security. It incorporates some suggestions made by various reviewers, and contains additional optimizations. The MD5 algorithm is being placed in the public domain for review and possible adoption as a standard.

...

The level of security discussed in this memo is considered to be sufficient for implementing very high security hybrid digital-signature schemes based on MD5 and a public-key cryptosystem.

Kryptosysteme



- Schreibweisen
  - *Verschlüsseln* mit Schlüssel K_1: Schlüsseltext = { Klartext }$_{K1}$
  - *Entschlüsseln* mit Schlüssel K_2: Klartext = { Schlüsseltext }$_{K2}$
- Eigenschaften von Kryptosysteme
  - { { Klartext } $_{K1}$ } $_{K2}$ = { { Klartext } $_{K2}$ } $_{K1}$ für bel. Schlüssel $K1$, $K2$
  - auch wenn { { Klartext } $_{K1}$ } $_{K2}$ ≠ Klartext
- *Symmetrische* Kryptosysteme: K1 = K2
- *Asymmetrische* 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, sind sowieso verdächtig!”

- Verschlüsselungsfunktion prinzipiell umkehrbar
  - ohne Kenntnis der Schlüssel jedoch höchstens mit unverhältnismässige 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 derzeit bekannte asymmetrische Verfahren

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

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”)

One-Time Pads: Übungen

- Wie beweist man, dass One-Time Pads garantiert sicher sind?
- Wie knackt man One-Time Pads bei endlichem, periodischem Schlüssel?
- Wie entlarvt man Schlüssel, die einen sinnvollen Text darstellen?
- Wie knackt man Nachrichten, die alle mit dem gleichen One-Time Pad verschlüsselt wurden?

- *Weitere Übungen auf ausgeteiltem Aufgabenblatt!*

crypt

SYNOPSIS

crypt [password]

DESCRIPTION

crypt encrypts and decrypts the contents of a file...
crypt encrypts and decrypts with the same key...

crypt implements a one-rotor machine designed along the lines of the German Enigma, but with a 256-element rotor. Methods of attack on such machines are widely known, thus crypt provides minimal security.

RESTRICTIONS

This program is not available on software shipped outside the U.S.

- Source-Code von UNIX enthielt früher einfach

```
#ifndef  EXPORT
```

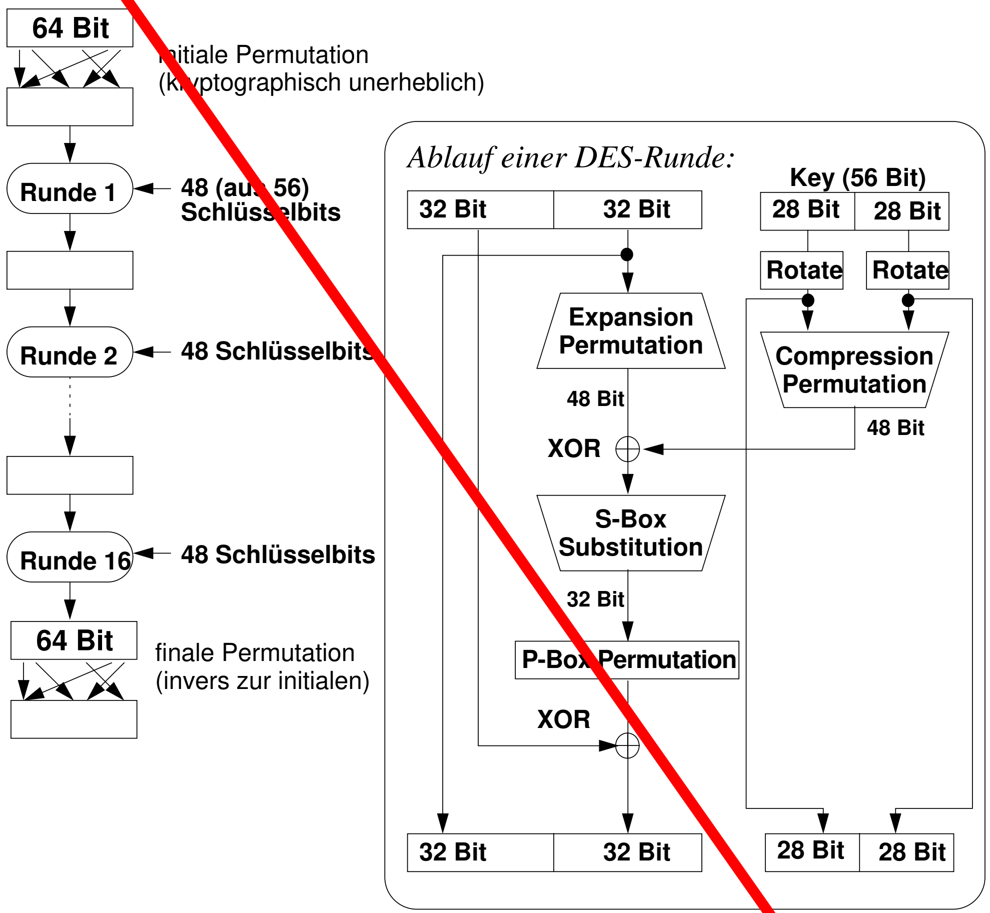
--> Präprozessor-Anweisung zur bedingten Übersetzung

DES (Data Encryption Standard)

- Symmetrisches Kryptosystem mit 56 Bit Schlüssellänge
  - plus 8 Paritätsbits zur Vermeidung undekodierbarer “falscher” Kodierungen
- Transformiert blockweise (64-Bit Blöcke)
- Algorithmus ist bekannt; zentrale Entwurfsprinzipien unterliegen der Geheimhaltung
  - mehrstufige Transposition, Substitution, exklusiv-oder
  - US-Exportrestriktionen für DES-Soft- und Hardware
- 1973: Ausschreibung durch US-Standardisierungsbehörde
  - Vorschlag der IBM auf Grundlage des LUCIFER-Algorithmus
  - Begutachtung durch US National Security Agency (NSA), daraufhin:
  - Reduzierung der Schlüssellänge von 112 Bit auf 56 Bit
  - „willkürliche“ Änderungen interner Details
- 1976: öffentliche Workshops zur Evaluierung
  - lebhafte Diskussionen (vor allem: Einbau einer Trapdoor durch NSA?)
  - erstmals Offenlegung wesentlichen Knowhows der NSA, dadurch erheblicher Schub für die öffentliche Kryptographieforschung
- Ebenfalls 1976: Einführung als offizieller US-Standard
  - mit regelmässiger Re-evaluierung und Zertifizierung nur auf Zeit
  - Nachfolgealgorithmus (insbes. 56 Bit-Schlüssel nicht mehr sehr sicher)
- Ursprünglich nur für Hardware-Realisierung konzipiert:
  - spezielle DES-Chips (1993: 32-MHz-Chip mit 200 MByte/s)
  - DES-Software erst seit 1993 zertifizierbar (i80486, 66 MHz: ca. 170 kByte/s)

DES: Prinzip

Prinzip des Verfahrens:



- Es bleibt (uns) ziemlich unklar, wieso diese verwickelt Bitmanipulation ein sicheres Verfahren darstellt...

was sind da eigentlich die Ansprüche?

DES: Bemerkungen zum Verfahren

- Schlüsseltransformation
  - Rotation je nach Runde um 1 oder 2 Bit
- Expansion und Permutation der rechten Klartexthälfte
  - dient vor allem der schnelleren Ausbreitung des Einflusses jedes Bits
- S-Box-Substitution
  - besteht aus 8 Boxen mit je 6 Eingängen und 4 Ausgängen
  - bildet eigentliches Herzstück des DES-Verfahrens
    --> alle übrigen Transformationen “leicht” analysierbar
  - monatelange Suche nach geeigneten Kandidaten bei IBM, aber von NSA stark modifiziert (Misstrauen gegen IBM? Trapdoor? Stärkung?)
  - genaue Entwurfskriterien bis heute geheim, aber bisherige Analysen zeigen: andere S-Boxen sind eher schlechter, weniger Runden ermöglichen einfache Kryptoanalyse (indirekte Rehabilitation der NSA!)

- Früherer Online-Manual-Eintrag unter UNIX: “man des”

NAME
des - encrypt or decrypt data using Data Encryption Standard

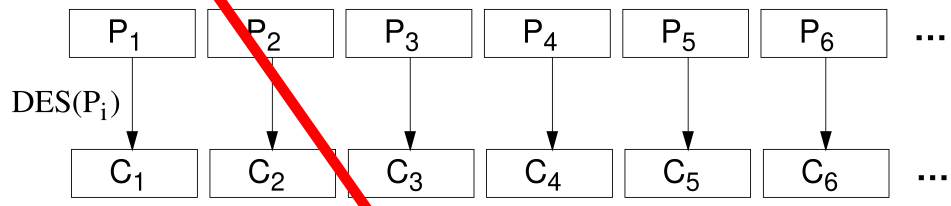
DESCRIPTION
des encrypts and decrypts data using the NBS Data Encryption Standard algorithm. One of -e (for encrypt) or -d (for decrypt) must be specified.

The des command is provided to promote secure exchange of data in a standard fashion.

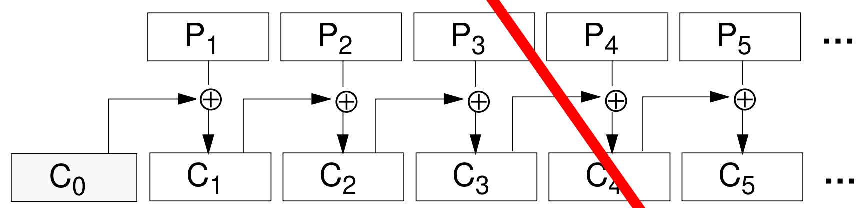
RESTRICTIONS
Software encryption is disabled for programs shipped outside of the U.S. The program will still be able to encrypt files if one can obtain an encryption chip, legally or otherwise.

DES: Betriebsarten und Varianten

- DES verschlüsselt blockweise (je 64 Bit); was tun bei langen Klartexten?
- 1. Methode: Block-für-Block-Verschlüsselung
  - Blockchiffren: “Electronic Codebook Mode”(ECB)



  - Selbstsynchronisation bei Übertragungsfehlern des Schlüsseltextes
  - aber: gleichartige 8-Byte-Gruppen werden immer gleich verschlüsselt!
- 2. Methode: Stromchiffren, z.B. “Cipher Block Chaining”



  - Verschlüsselung: $C_i = DES(P_i$ **XOR** $C_{i-1})$;
  - Entschlüsselung: $P_i = DES^{-1}(C_i)$ **XOR** C_{i-1}
  - Es gibt eine Reihe weiterer Varianten von Chaining-Verfahren

---

- Triple-DES --> Verlängerung des Schlüssels (Faktor 2)

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

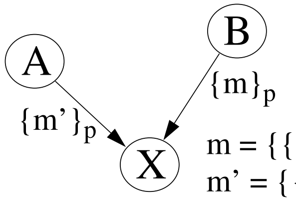
- Für jeden Prozess X existiert ein Paar (p,s)

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

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

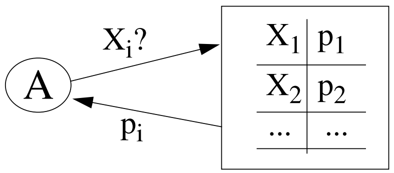
- Jeder Prozess, der an X sendet, kennt p

- Nur X selbst kennt s



$m = \{\{m\}_p\}_s$
$m' = \{\{m'\}_p\}_s$

- *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 aus $\{m\}_p$ (oder $\{m\}_s$) 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 private 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?)

Symmetrische und asymmetrische Kryptosysteme in der Praxis

- DES in Software auf einer 175-MHz DEC Alpha-Workstation benötigt 4 µs für eine Anwendung

- Ausprobieren aller Schlüssel --> ca. 4500 Jahre
  - aber: viele Prozesoren parallel?
  - es gibt spezielle sehr schnelle VLSI-Chips (Raten von bis zu 1 Gb/s)
  - wenn Teile des Schlüssels bekannt sind, geht es viel schneller

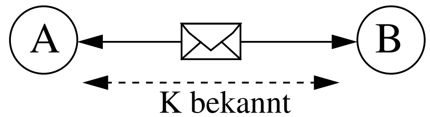
- Es gibt Alternativen zu DES
  - IDEA(International Data Encryption Algorithm):
    - ETH Zürich (X. Lai, J. Massey) und Ascom
    - Schlüssellänge 128 Bit
    - keine Export- / Anwendungsrestriktionen
    - Prinzip analog zu DES
    - es gibt Chip mit einer Verschlüsselungsrate von 177 Mb/s
    - wird z.B. als Verschlüsselungsmethode in PGP benutzt

- Public-Key-Verfahren sind wesentlich langsamer
  - z.B. RSA-Chip mit Verschlüsselungsrate von 64Kb/s
  - DES ist gut 1000 Mal schneller

- RSA benötigt eine wesentlich grössere Schlüssellänge zur Erzielung vergleichbarer Brute-Force-Resistenz als DES

- Generell: Vorsicht bei automatischer Schlüsselgenerierung (Pseudozufallszahlen können erratbar sein!)

Authentifizierung mit symmetrischen Schlüsseln



Sei K der zwischen A und B vereinbarte (und geheimzuhaltende!) 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")

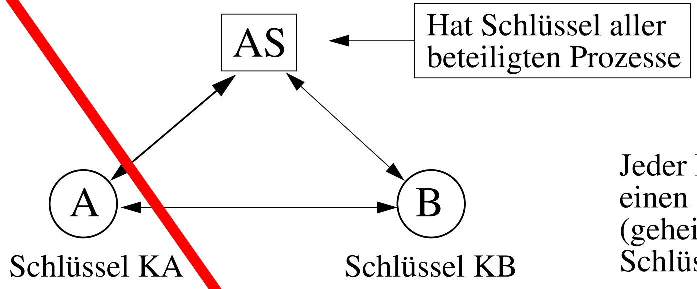
A: n' := $\{n\}_K$

A-->B: n'

B: überprüfe, ob $\{n\}_K$ = n'

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

Authentifizierung mit Authentifizierungsserver



Jeder Prozess hat einen einzigen (geheimen) Schlüssel

A-->B: "Ich bin A"
B-->A: n
A: n' := $\{n\}_{KA}$
A-->B: n'

Wie oben (2. Verfahren)

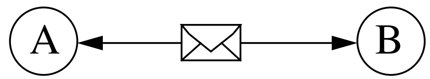
B: berechne n" := $\{A,n'\}_{KB}$
B-->AS: n"

A hat eine von mir ausgedachte Zahl mit n' kodiert. Wie lautet diese Zahl?

AS: ermittle {A,n'} aus n" mittels KB;
berechne m := $\{\underbrace{\{n'\}_{KA}}_{n}\}_{KB}$
AS-->B: m
B: überprüfe, ob m = $\{n\}_{KB}$

Bemerkung: Authentifizierungsserver geniesst Vertrauen aller Prozesse und ist gleichzeitig die zentrale *Schwachstelle*: Angreifer könnte in Besitz aller keys kommen!

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"

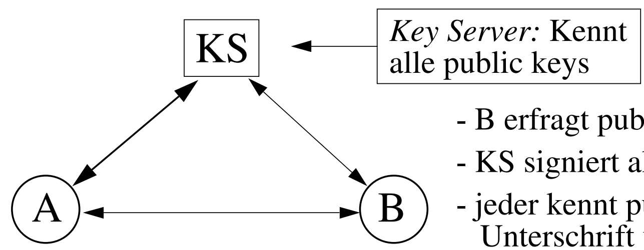
$m' = \{ m, n \}_{sA}$ —— *m, m'* ——> $[m, n] = \{ m' \}_{pA}$? Falls ja:

digitale Unterschrift

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

⇒ „Dies muss von Alice stammen!"

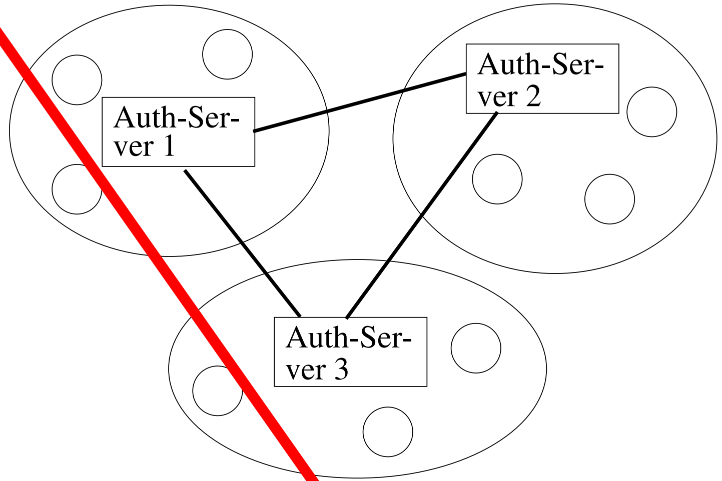
- geschützt gegen Replays (wieso?)
- Vorsicht: "Man in the middle"-Angriff möglich (wie?)
- Nachteil: B muss viele public keys speichern; alternativ:



- 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

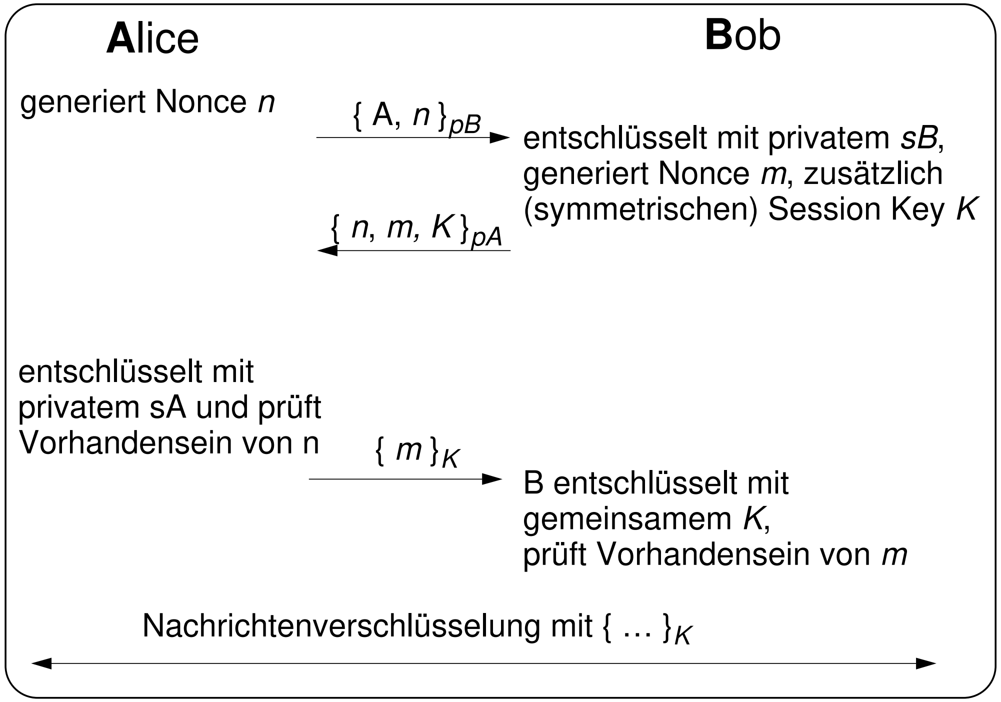
Dezentrale Authentifizierungsserver



- Authentifizierungsdienst ist auf einzelne Zuständigkeitsbereiche verteilt
  - effizienter
  - einfacher zu verwalten
- Zuständigkeitsbereiche werden durch andere Dienste geregelt (z.B. Name-Service, Namenskonventionen...)
- Server kommunizieren untereinander in sicherer und authentischer Weise
- Lokaler Server garantiert Authentizität für seine Prozesse auch bzgl. entfernter Prozesse
  - wenn sich die Server gegenseitig anerkennen / zertifizieren
  - Problem, wenn diese unterschiedliche "policies" bzgl. Sicherheit, Authentifizierung... haben
- Ggf. Replikation: Ausfallsicherheit und erhöhter Schutz vor Angriffen

Gegenseitige Authentifizierung mit Schlüsselvereinbarung

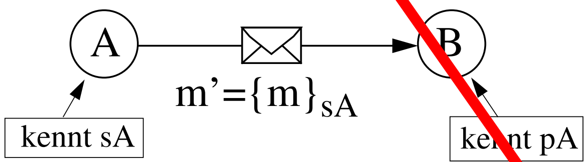
- Im Prinzip möglich wie oben beschrieben nacheinander in beide Richtungen
- Gleich beides zusammen erledigen ist aber effizienter!
- Hier zusätzlich: Vereinbarung eines symmetrischen "session keys" K, der nach der Authentifizierung zur effizienten Verschlüsselung benutzt wird
- Voraussetzung: A und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



Digitale Unterschiften

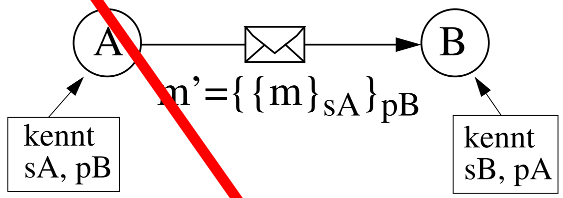
- Zweck: Schliessen von “elektronischen Verträgen”, z.B.:
  - Unterzeichnen von Dokumenten zum Nachweis der Urheberschaft
  - Ausstellen elektronischer Schecks
- Gewünschte Eigenschaften (wie gewöhnliche Unterschrift):
  - *Authentizität* der Unterschrift und *Nichtabstreitbarkeit*: niemand sonst kann so unterzeichnen und das Unterzeichnete stammt zweifelsfrei vom Unterzeichner
  - *Fälschungssicherheit* des Unterschriebenen
  - *Nichtwiederverwendbarkeit*: kein “Ausschneiden und auf anderes Dokument kleben”

- Mit asymmetrischer Verschlüsselung (“public key”):



  - bei Rechtsstreitigkeiten: B legt $\{m\}_{sA}$ vor und zeigt, dass $\{\{m\}_{sA}\}_{pA}$ wirklich m ergibt
  - nur A konnte $\{m\}_{sA}$ erzeugen, das sich tatsächlich entschlüsseln lässt
- Probleme
  - Replays (z.B. Kopierbarkeit bei Schecks) --> Sequenznummern, nonce...
  - Verschlüsselung langsam --> unterzeichne nur message digest
  - Zuordnung [A, pA] muss auf ewig öffentlich bekannt bleiben
  - um pA verlässlich in Erfahrung zu bringen, braucht es oft einen verlässlichen Schlüsselserver

Geheime, authentische Nachrichten mit asymmetrischer Verschlüsselung



--> $m=\{\{m'\}_{sB}\}_{pA}$

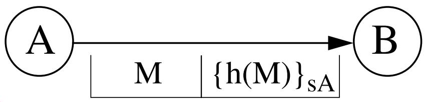
- Nur B kann die äussere Hülle entfernen
- Nur A kann die Nachricht mit sA verschlüsselt haben

Beachte: Die Nachricht ist zwar geheim (da mit pB verschlüsselt), sie könnte jedoch abgehört und später wiederholt eingespeist werden (“replay”)

--> Weitere Massnahmen nötig, um “vollständige” Sicherheit zu gewährleisten!

Nachrichten-Authentizität mit Fingerprints

- Integritätsschutz von M mit *asymmetrischen* Schlüsseln:



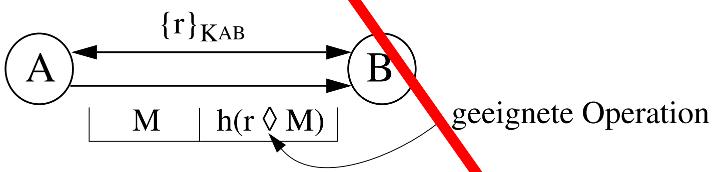
sA = secret key von A

- Fingerprint {h(M)} ist von A signiert
- Signieren von {h(M)} ist effizienter als Signieren von M
- jemand, der M unterwegs zu M' ändert, kann kein $\{h(M')\}_{sA}$ erzeugen
- Ersetzen von M durch M'' mit $\{h(M'')\}_{sA} = \{h(M)\}_{sA}$ ist praktisch unmöglich

---

- Integritätsschutz mit *symmetrischem* Schlüssel K_{AB}:

- A und B vereinbaren mittels K_{AB} eine geheime grosse *Zufallszahl* r



- Empfänger kann den Fingerprint h(r ◊ M) überprüfen, da er r kennt
- ein Angreifer, der M zu M' ändert, müsste auch wieder ein korrektes h(r ◊ M') erzeugen - das sollte sehr schwierig sein!
- *Denkübungen:* r ◊ M ist vermutlich effizienter als das digitale Unterschreiben des ersten Verfahrens, aber ist es (z.B. für xor oder Konkatenation) auch sicher? Wie oft sollte r neu vereinbart werden?

- Beachte: Die gesamte Nachricht (inkl. Fingerprint) sollte *verschlüsselt* und gegen *Replays* gesichert werden

"Digitale Einschreiben"

Problem: Wie kann A beweisen, dass A etwas an B gesendet hat?

- solche Aspekte wichtig im Zusammenhang mit "electronic commerce"

1) *Quittung* verlangen:

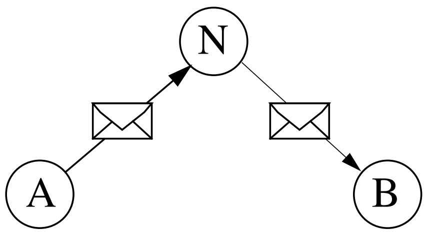
B sendet die gleiche Nachricht zurück, versehen mit B's digitaler Unterschrift (und verschlüsselt)

- aber wenn B behauptet, dass dies gefälscht sei, da er aus seinem secret key (also seiner "Unterschrift") nie ein Geheimnis machte?

---

2) *Notar* N *als Zeugen* einschalten:

- A sendet geheime, authentische Nachricht an N
- N registriert diese und sendet sie verpackt in eine eigene, geheime Nachricht an B weiter



- Notar muss vertrauenswürdig sein
- Abstreiten des Empfangs von B ist sinnlos (zumindest ist A seiner Verpflichtung nachgekommen)

Mit Notarsrolle kann auch der Besitzstandwechsel von digitalen Unikaten garantiert werden

- z.B. "Tickets" für lizenzierte Software

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.

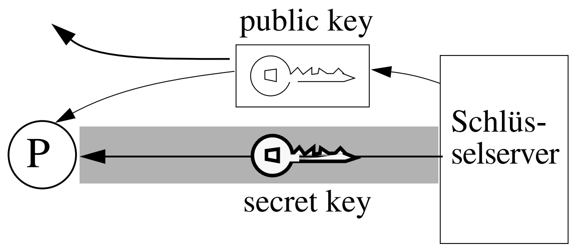
- lokale Uhrzeit
- globale Zeitapproximation aus Zeitservice (z.B. NTP-Protokoll)
- Empfängerzeit vorher erfragen

- 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

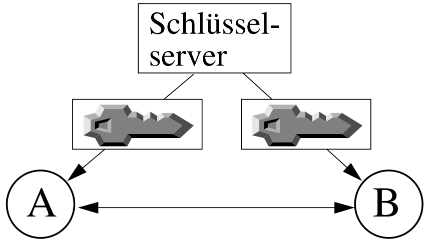
- Zur Vergabe eines Paares von public-, secret-keys:



- 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")

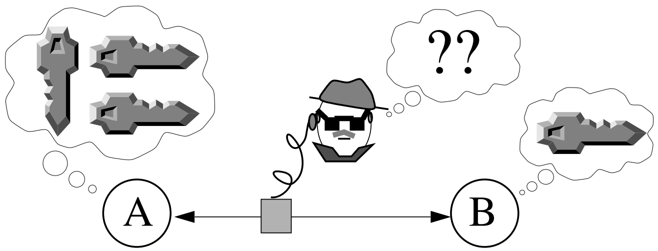
--> erhöhte Sicherheit gegen Angreifer



- 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

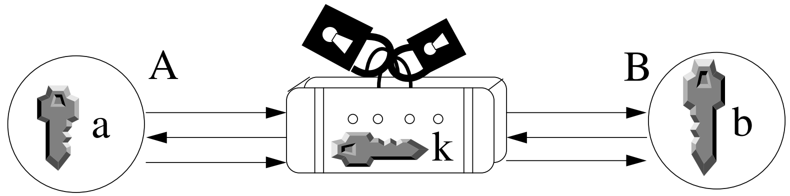


- Problem: A und B wollen sich über einen unsicheren Kanal auf einen gemeinsamen Schlüssel einigen, ohne einen Schlüsselserver zu verwenden

- 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

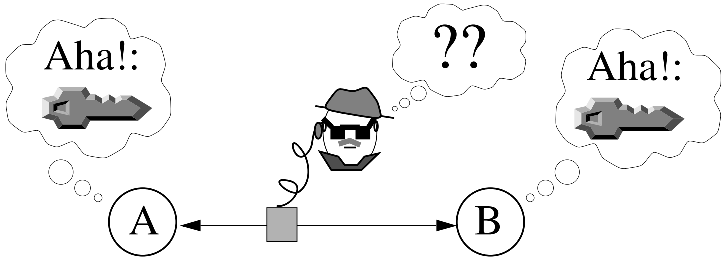
Beachte: k wird nie offen transportiert!

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

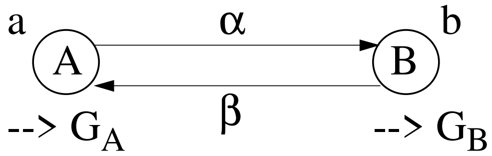
(1 < c < p; i.a. ist p eine grosse Primzahl)

- 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):

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

$$\left.\begin{array}{l} a = 3 \text{ --> } \alpha = 6 \\ b = 4 \text{ --> } \beta = 2 \end{array}\right\} \begin{array}{l} \text{--> } G_B = 6^4 \bmod 7 = 1 \\ \text{--> } G_A = 2^3 \bmod 7 = 1 \end{array}$$

$G_A = G_B$

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

$$\boxed{(c^b \bmod p)^a \bmod p \;=\; (c^a \bmod p)^b \bmod p}$$

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

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

Bemerkungen:

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

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. IT22, 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. ...

---> "Patent-Server" http://www.patents.ibm.com/

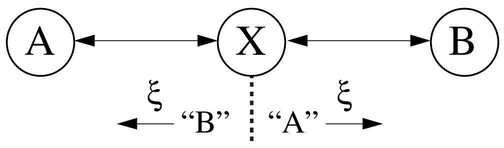
Sweet Little Secret G

- 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!
  - $\alpha = c^a \bmod p$ --> 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 --> langwierig
  - α und β sind *unabhängig* voneinander! (Wieso ist das ein Argument?)

- Wie ist es aber bei *aktiven Angreifern*?

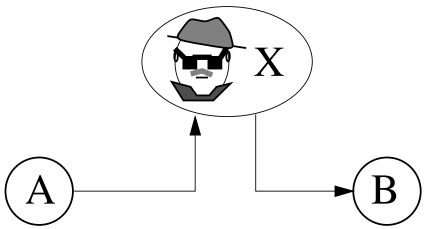


- "man in the middle"
- ein ξ für ein β bzw. α vormachen!

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

Aktive Angriffe durch Eindringen und Schlüsselfälschung

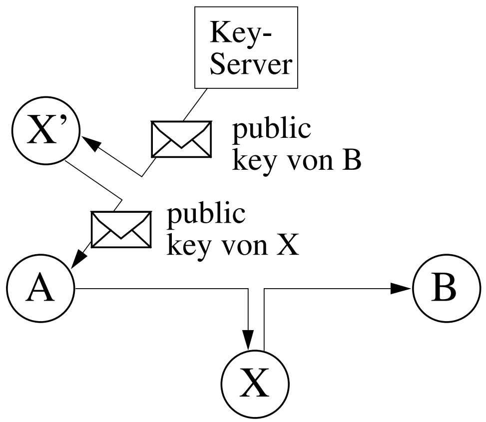
Szenario 1:



- X verhält sich gegenüber A wie B, gegenüber B wie A (--> X arbeitet “transparent”)
- z.B. eigene Schlüssel für die Teilstrecken vereinbaren

- Challenge-Response-Tests: X reicht Challenges einfach an von ihm vorgetäuschten Partner weiter und miemt mit abgefangener Antwort die angenommene Identität

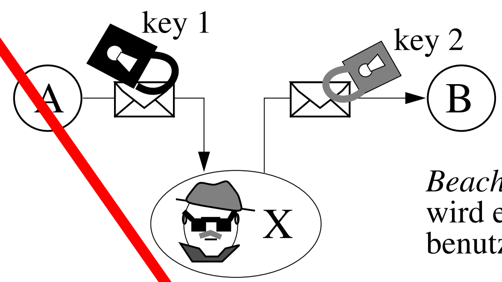
Szenario 2:



- kompromittierter Key-Server; Verschwörung...
- X kann alle von A mit dem falschen Schlüssel verschlüsselten Nachrichten an B entziffern
- X verschlüsselt danach die Nachricht mit dem richtigen Schlüssel für B

- digitale Unterschrift des Key-Servers nützt nichts, wenn A den Prozess X’ für den Key-Server hält und dessen Unterschrift akzeptiert!
- nützt die allgemeine Bekanntgabe des public keys des Key-Servers?
- ist es überhaupt möglich, X in diesen Szenarien zu erkennen?

Erkennen von Eindringlingen



Beachte: Auf der Strecke AX wird ein anderer Schlüssel benutzt als auf der Strecke XB!

1) B stellt eine Anfrage, die nur A beantworten kann
2) A generiert die Antwort und verschlüsselt diese
3) A sendet nur die Hälfte davon an “B”
   - z.B. nur die geraden Bits
   - B erwartet die Hälfte der Antwort in weniger als t Zeiteinheiten
4) Ohne die andere Hälfte kann X diese nicht entschlüsseln und neu verschlüsseln
   - oder könnte X erzwingen, dass key 1 = key 2 ist?

5) Erst nach t Zeiteinheiten sendet A die andere Hälfte
   - B setzt Schlüsseltexthälften zusammen und überprüft Antwort

---

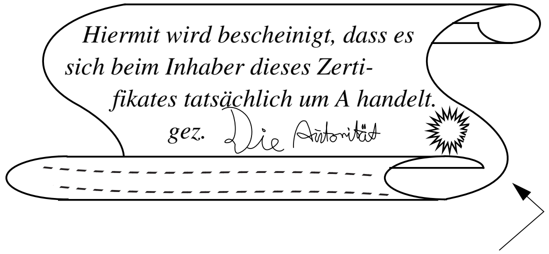
--> Gibt X die halbe Nachricht unverändert weiter, kann B diese nicht entschlüsseln --> *Fälschung erkannt*

--> Behält X die halbe Nachricht bis zum Eintreffen der anderen Hälfte (und speichert die andere Hälfte dann t Zeiteinheiten zwischen), dann arbeitet X nicht mehr zeittransparent --> *Eindringling erkannt*

Frage: Wird in 1) nicht schon ein gemeinsames Geheimnis vorausgesetzt? Können (im Kontext des Diffie-Hellmann-Verfahrens) A und B nicht dieses benutzen, um einen von X nicht ermittelbaren gemeinsamen Schlüssel zu finden? Oder genügt in 1) eine schwächere Eigenschaft (“originelle” Antwort; Fähigkeit, die nur A hat...)?

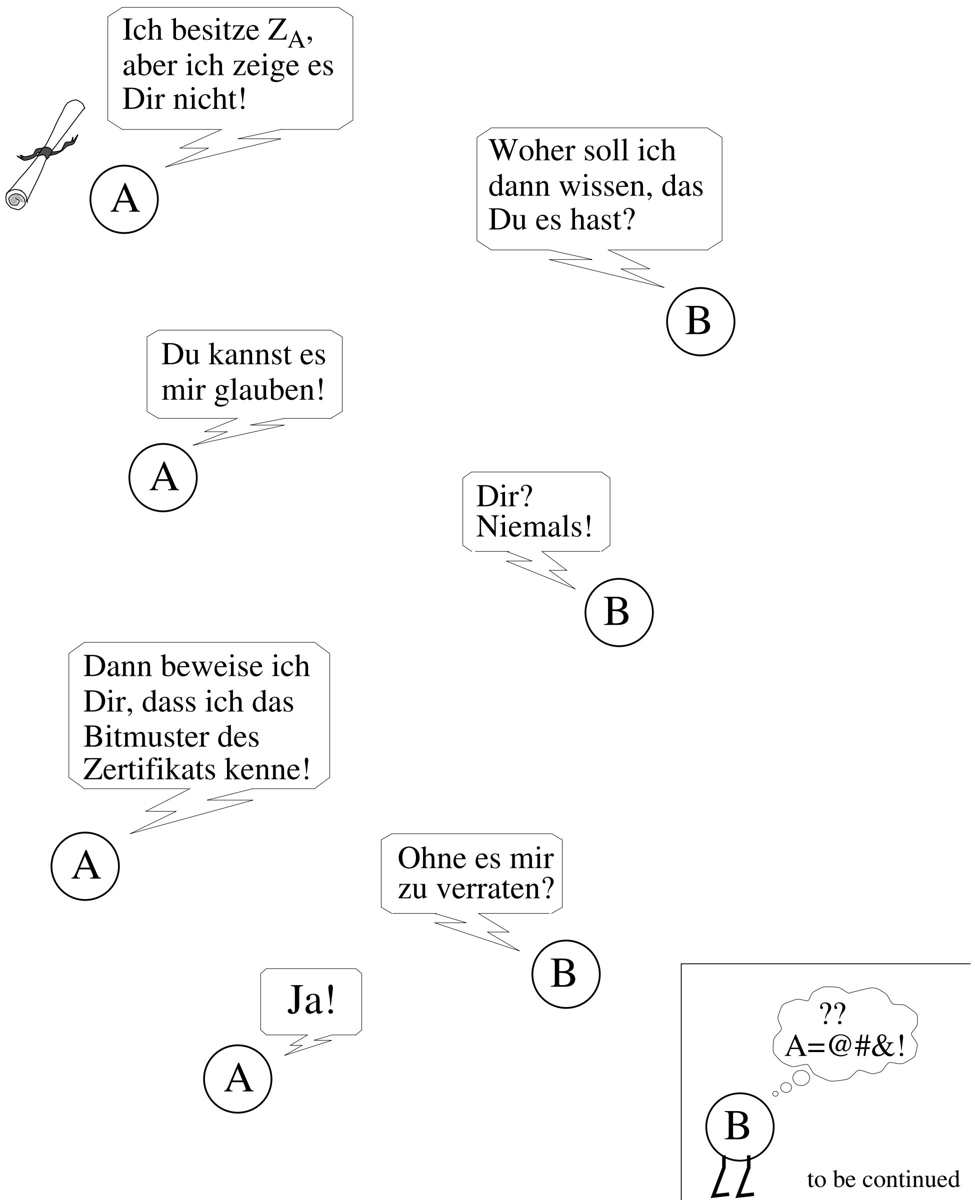
Authentifizierung mit Zertifikaten ?

- Die Idee:



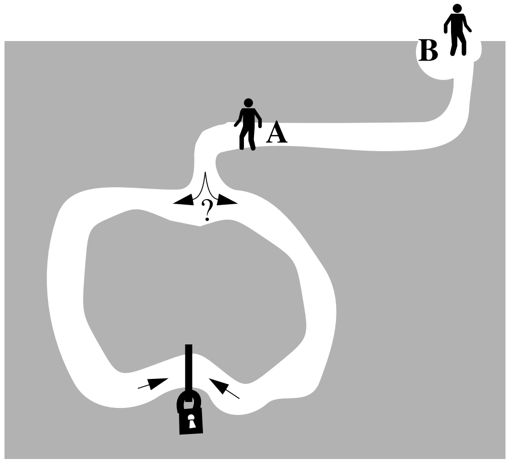
- A lässt sich einmalig von einer Autorität ein *Zertifikat* Z_A mitgeben (sollte von der Autorität signiert sein)
  - Autorität gilt als vertrauenswürdig und hat A ggf. persönlich in Augenschein genommen (oder einem fremden Zertifikat vertraut)
- Wenn B an A's Identität zweifelt, weist A B auf sein Zertifikat Z_A hin
  - 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 ?



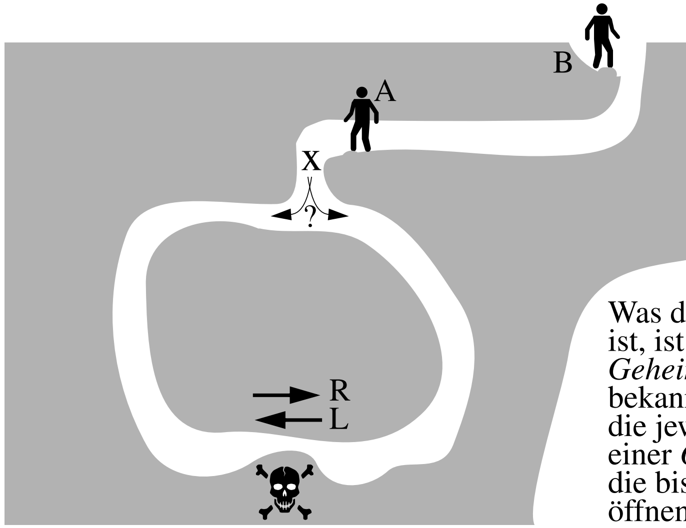
Geheime Zertifikate !

- 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
- Ein “Höhlengleichnis” dazu (in Kurzform):



- A beweist B, dass er das Geheimnis des Türöffners kennt, ohne es offenzulegen:
- A begibt sich unbeobachtet in linken oder rechten Seitengang
- B folgt bis zur Gabelung; bittet A, aus zufällig gewähltem Seitengang zu erscheinen, links oder rechts
- A hat 50%-Chance, erfolgreich zu lügen

Ein Höhlengleichnis



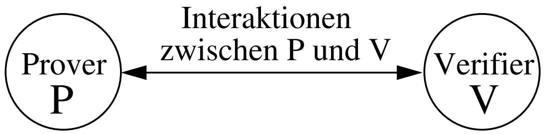
Der Höhlendämon lässt nur diejenigen die Engstelle lebendig passieren, die dort in der richtigen Richtung (L oder R) vorbeigehen.

Was die richtige Richtung ist, ist ein streng gehütetes *Geheimnis*; es ist lediglich bekannt, dass der Dämon die jeweilige Entscheidung einer *Orakelbox* entnimmt, die bisher noch niemand öffnen konnte.

- A sagt zu B: “Ich kenne das Geheimnis. Das beweise ich Dir, ohne das Geheimnis zu verraten!”
  - A begibt sich in die Höhle bis zur Engstelle; erst danach folgt B bis zur Stelle x (B sieht nicht, welche Richtung A dort eingeschlagen hat)
  - B ruft A *entweder*
    - “komm links heraus!” *oder*
    - “komm rechts heraus!” zu
  - A tut dies, indem A ggf. die Engstelle (in der richtigen Richtung) passiert
  - A und B verlassen zusammen die Höhle
- Nachdem A das ganze n Mal überlebt hat, glaubt B, dass A das Geheimnis (= Funktion der Orakelbox) kennt!
  - Die Irrtumswahrscheinlichkeit beträgt nur 2^{-n}
- B hat in diesem “interaktiven Beweis” das Geheimnis nicht erfahren! ==> “Zero knowledge proof”

Zero-Knowledge-Proofs

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



Hat Kenntnis einer bestimmten Information

Überprüft, ob P diese Kenntnis wirklich hat

- 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

---

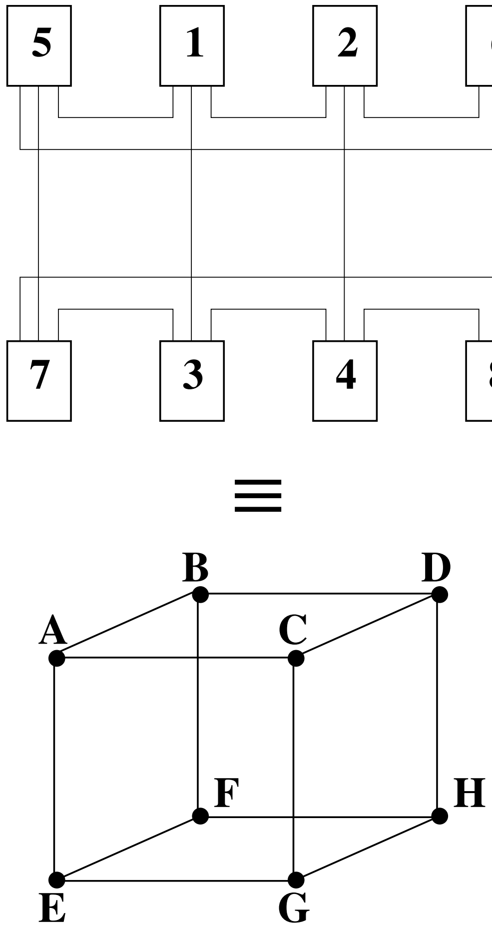
Idee:

Problem X —(Transformiere mit Einwegfunktion)→ Problem X' —(Verrate Lösung von X')→

(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 Knotenzuordnung 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

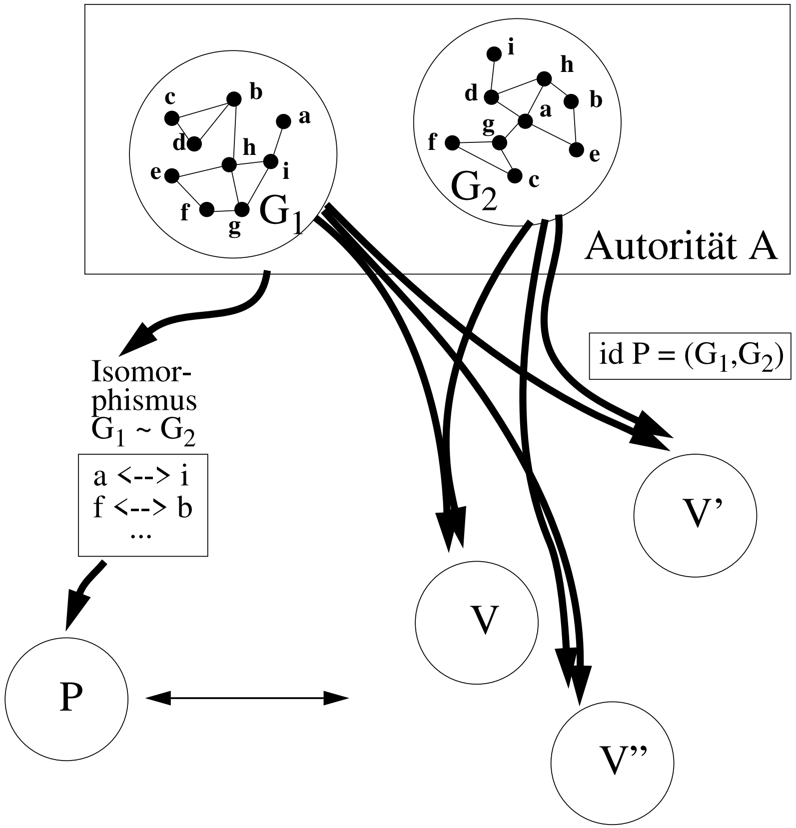
- Zur Minimierung der Interaktionen lassen sich die "Runden" *parallelisieren*: P sendet *mehrere* "isomorphe Zeugen" an V, und V sendet einen Bitvektor zurück, der die Einzelnachweise auswählt

- 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)

Identitätsbeweise mit Graphisomorphie?



- So wären Identitätsbeweise *im Prinzip* möglich:
  - A veröffentlicht zwei isomorphe Graphen G_1, G_2 als "Identität" von P (entspricht einer Art public key)
  - A teilt nur P den Isomorphismus $G_1 \sim G_2$ mit (entspr. secret key)
  - P kann einem V nun seine Identität durch Zero-knowledge-Proof von $G_1 \sim G_2$ beweisen
- Nachteil: Graphen sind sehr gross!

Zero-Knowledge-Identitätsbeweise

Hier: Leicht vereinfachtes Protokoll von Fiat und Shamir (1986)

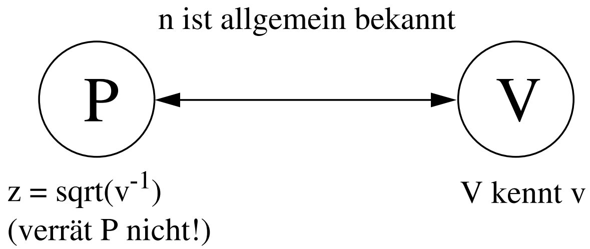
- Bemerkung: Die k-te Wurzel in einem Restklassenring mod n zu bestimmen ist sehr schwierig, wenn man nicht die Zerlegung n = pq kennt ("trap door"); die Berechnung der Zerlegung pq aus n ist genauso schwer

- Modulus n sei eine mehrere 100 Bit lange Zahl; nur eine Zertifizierungsautorität X kennt die Primzahlen p, q

- Jeder Prozess P bekommt von X eine Zahl v sowie ein Zertifikat z. Die Zahlen n und v sind allgemein bekannt

- Dabei wird v so gewählt, dass $x^2 = v \bmod n$ eine Lösung hat und v^{-1} existiert; das (geheime!) z ist die kleinste Quadratwurzel aus v^{-1}

  Die Bestimmung der Quadratwurzel ist dabei für X einfach, da X die Zerlegung von n in p und q kennt



Das Protokoll von Fiat und Shamir

P möchte seine Identität gegenüber V beweisen

$P \longleftrightarrow V$

1) $\xrightarrow{x}$

(mit $x = r^2 \bmod n$ und $r \neq 0$ eine Zufallszahl $< n$)

2) $\xleftarrow{b}$

(mit $b \in \{0,1\}$ zufällig)

3) $\xrightarrow{y}$

(mit $y = r$ falls $b = 0$, $y = r\,z \bmod n$ sonst)

4) - Falls $b = 0$ war: V überprüft, ob $x = y^2 \pmod n$ gilt.
==> P kennt sqrt(x) = r (selbstverständlich, wenn P "echt" ist!)

- Falls $b = 1$ war: V überprüft, ob $x = y^2\,v \pmod n$ gilt.
(Wenn P tatsächlich $y = r\,z \bmod n$ in Schritt 3 gesendet hat, ist das der Fall: $y^2\,v = (r\,z)^2\,v = r^2\,z^2\,v = r^2\,v^{-1}\,v = r^2 = x$.)
==> P kennt $\mathrm{sqrt}(x\,v^{-1}) = y$

5) - Schritte 1 - 4 des Protokolls werden mehrfach wiederholt.

- Beachte: Alle Operationen sind effizient durchführbar (es wird z.B. nirgendwo die Quadratwurzel " sqrt" explizit angewendet)
- Das Protokoll soll angeblich ein vielfaches effizienter sein als eine digitale Unterschrift mit dem RSA-Verfahren

Die Situation des Angreifers

Ein Angreifer A kennt z nicht: z wird nie gesendet, kann also nicht direkt abgehört werden. Auch falls $x = r^2 \pmod n$ und $y = r\,z \pmod n$ abgehört wird, kann daraus nicht r (und damit z) ermittelt werden.

Kann sich aber A *ohne Kenntnis von z* für P ausgeben?
A muss zunächst ein x, dann ein *dazu passendes* y senden.

(a) A setzt darauf, dass $b = 0$ von V gewählt wird:
A wählt ein beliebiges r und sendet zunächst $x = r^2 \pmod n$), sowie später $y = r$.

(b) A setzt darauf, dass $b = 1$ von V gewählt wird:
A wählt im Vorfeld ein beliebiges y (!), berechnet daraus $x = y^2\,v \pmod n$.
A sendet zunächst dieses x, sowie später das gewählte y.

Falls A die Reaktion von B (d.h. b=0 oder 1) richtig vorhergesagt hat, wird V den Beweis für diese Runde akzeptieren!

Was aber, wenn sich A bzgl. b verspekuliert hat?
(a) A müsste statt $y = r$ nun $y = \mathrm{sqrt}(x\,v^{-1})$ senden (mit $x = r^2 \pmod n$)).
(b) A müsste statt dem gewählten y nun sqrt(x) senden (mit $x = y^2\,v \pmod n$)).

Die Bestimmung von sqrt ist aber (praktisch) nicht machbar!
==> A hat pro Runde nur eine 50% Wahrscheinlichkeit, unentdeckt zu bleiben!

Denkübungen: (1) Man überlege sich die Stichhaltigkeit der Argumente.
(2) Muss v allgemein (insbesonder V) bekannt sein, oder könnte P diesen Wert zusammen mit x an V übermitteln?
(3) A höre P so lange ab, bis P ein r^2 zum zweiten Mal sendet. Kann A aus den zugehörigen y-Werten etwas entnehmen?

Ein Beispiel

- Sei $n = 35$ (mit Primfaktoren 5 und 7)

ist unrealistisch klein!

- $x^2 = v \pmod{35}$ hat für folgende v eine Lösung:

1: $x^2 = 1 \pmod{35}$ wird gelöst durch x=1, 6, 29, 34
4: $x^2 = 4 \pmod{35}$ wird gelöst durch x=2, 12, 23, 33
9: $x^2 = 9 \pmod{35}$ wird gelöst durch x=3, 17, 18, 32
11: $x^2 = 11 \pmod{35}$ wird gelöst durch x=9, 16, 19, 26
14: $x^2 = 14 \pmod{35}$ wird gelöst durch x=7, 28
15: $x^2 = 15 \pmod{35}$ wird gelöst durch x=15, 20
16: $x^2 = 16 \pmod{35}$ wird gelöst durch x=4, 11, 24, 31
21: $x^2 = 21 \pmod{35}$ wird gelöst durch x=14, 21
25: $x^2 = 25 \pmod{35}$ wird gelöst durch x=5, 30
29: $x^2 = 29 \pmod{35}$ wird gelöst durch x=8, 13, 22, 27
30: $x^2 = 30 \pmod{35}$ wird gelöst durch x=10, 25

- Zu v = 14, 15, 21, 25, 30 existiert kein Inverses mod 35 (sind nicht relativ prim zu 35). Für die anderen v gilt:

v	v^{-1}	$\mathrm{sqrt}(v^{-1})$
1	1	1
4	9	3
9	4	2
11	16	4
16	11	9
29	29	8

Es werde **v = 9** und **z = 2** gewählt.

1) P wählt eine Zufallszahl r = 16, und sendet $x = 16^2 \pmod{35} = \underline{11}$ an V.

2) V sendet b = 1 zurück.

3) P sendet $y = r\,z = 16 \cdot 2 = 32$ an V.

4) V verifiziert $y^2\, v \pmod{n} =$ $32^2\, 9 \pmod{35} = \underline{11 = x}$

Geheimnisverrat?

- Die Autoren Feige, Fiat, Shamir haben im Juli 1986 ihren Algorithmus in den USA zum Patent angemeldet

- Ein halbes Jahr später teilte das Patentamt mit, dass die Veröffentlichung die nationale Sicherheit gefährde

- Die Autoren wurden aufgefordert, alle Nicht-Amerikaner zu nennen, denen der Algorithmus anvertraut wurde

---

- Allerdings: Die Autoren haben in der zweiten Jahreshälfte ‘86 auf mehreren Konferenzen in Europa, Israel und den USA die Ergebnisse vorgestellt!

- Ausserdem sind die Autoren keine Amerikaner, und die gesamte Forschungsarbeit wurde in Israel durchgeführt

---

- Nachdem dies in der Öffentlichkeit zu Heiterkeit und Entrüstung geführt hatte, wurde die Geheimhaltungsanordnung zurückgenommen

Identitätsnachweise

- Beim Verfahren von Fiat und Shamir sind typischerweise 20 und mehr Runden nötig, um “sicher” zu sein.

- Dies ist viel, wenn Einzelnachrichten aufwendig sind.
  - jede einzelne Nachricht löst im Rechner ggf. eine Transaktion aus
  - z.B. bei Kreditkartenverifikation über Telefonmodems

- Für Smart-Cards und ähnliche Anwendungen wurden Varianten des Protokolls entwickelt, die mit einer *einzigen “Runde”* (3 Nachrichten) auskommen, um eine Irrtumswahrscheinlichkeit von 2^{-m} zu erreichen.
  - wobei m innerhalb gewisser Grenzen frei gewählt werden kann
  - z.B. Verfahren von Guillou und Quisquater (Unterschied zu Fiat und Shamir u.a.: k-te Wurzel ($k \geq 2$) aus v und Zufallszahl statt Zufallsbit b)

- Auch Zero-knowledge-Identitätsprotokolle helfen nicht gegen jeglichen *Missbrauch*, z.B.:
  - Beantragen eines Zertifikats für eine *falsche Identität* bei der Autorität
  - Beantragen *verschiedener* Zertifikate (“multiple identity”)
  - *“Verleihen”* von Zertifikaten
  - *Kopieren, Vertauschen* von Zertifikaten etc.

==> Man beweist nicht seine *Identität*, sondern dass man ein *Zertifikat* kennt!
  - Genetischer Fingerabdruck als Zertifikat der Identität?
  - Was eigentlich ist die “Identität”?

Übungsbeispiel zu Kryptographie (1)

(Von Reinhard Schwarz)

Münzwurf über Telefon

Prinzip: *„Münzen in der Schatzkiste“*

- **Alice:** (unbeaufsichtigt)
  - klebt je eine Münze mit „Kopf“ und „Zahl“ auf den Boden zweier Schatzkisten
  - verschliesst beide Schatzkisten und legt sie Bob zur Auswahl vor
- **Bob:** (ggf. unbeaufsichtigt)
  - wählt zufällig eine der verschlossenen Kisten
- **Alice und Bob:** (unter gegenseitiger Kontrolle)
  - öffnen gemeinsam die von Bob gewählte Kiste und lesen die Münzoberseite ab

Probleme:

- Wie simuliert man „verschlossene Schatzkiste“?
- Wie simuliert man „gegenseitige Kontrolle“ in verteiltem System?

Münzwurf über Telefon

Gegeben: kommutatives Public-Key-System

- Alice und Bob wählen jeweils Paare [*sA*, *pA*], [*sB*, *pB*]: Beide Schlüsselpaare geheim!

Alice an Bob:	sendet { „Kopf" $\}_{sA}$ und { „Zahl" $\}_{sA}$ an Bob.
Bob:	wählt willkürlich ein Chiffrat (z.B. {„Zahl" $\}_{sA}$); verschlüsselt es zu { {„Zahl" $\}_{sA}$ $\}_{sB}$
Bob an Alice:	antwortet mit { {„Zahl" $\}_{sA}$ $\}_{sB}$
Alice:	bildet { { {„Zahl" $\}_{sA}$ $\}_{sB}$ $\}_{pA}$ = { { {„Zahl" $\}_{sA}$ $\}_{pA}$ $\}_{sB}$ = {„Zahl" $\}_{sB}$
Alice an Bob:	sendet halb entschlüsselte Nachricht {„Zahl" $\}_{sB}$ zurück.
Bob:	entschlüsselt { { {„Zahl" $\}_{sB}$ $\}_{pB}$ = **„Zahl"**
Bob an Alice:	teilt seinen (bis dahin geheimen) Schlüssel *pB* mit
Alice:	entschlüsselt Nachricht aus Schritt 5 als { { {„Zahl" $\}_{sB}$ $\}_{pB}$ = **„Zahl"**
Alice an Bob:	teilt ihren (bis dahin geheimen) Schlüssel *pA* mit

Komplexere Protokolle

Denkübungen zum Protokoll „Münzwurf über Telefon"

- **Optimierungsvorschläge**
  - Warum nicht einfacher? Etwa so…

Alice an Bob:	bildet { „Kopf" $\}_{sA}$ und { „Zahl" $\}_{sA}$ und sendet beides an Bob.
Bob an Alice:	sendet „Ich wähle die zweite Nachricht" zurück
Alice an Bob:	antwortet mit *pA*
Bob:	entschlüsselt { { „Zahl" $\}_{sA}$ $\}_{pA}$ = „Zahl"

  - Weglassen letzter Schritt?
  - Vertauschen der beiden letzten Schritte? Vertauschen der beiden Schritte davor?

- **Verwendung eines *Public-Key*-Systems**
  - Warum hält Alice *sA* zunächst geheim?
  - Warum hält Bob *sB* zunächst geheim?
  - Wann dürfen *sA* und *sB* offengelegt werden?

Übungsbeispiel zu Kryptographie (4)

Komplexere Protokolle

Denkübungen zum Protokoll „Münzwurf über Telefon" (Fortsetzung)

- **Voraussetzung bisher: Verfügbarkeit eines kommutativen Public-Key-Systems**
  – harte Forderung: Existenz solcher Systeme? Rechenaufwand?
- **Alternative 1**
  – Verwende ein nicht-kommutatives System!
  – Löse die dadurch entstehenden Probleme auf anderem Wege!
  – Beitrag der Kommutativität hinsichtlich Geheimhaltung und Festlegung (Commitment) auf einen bestimmten Nachrichteninhalt?
- **Alternative 2**
  – Verwende statt Verschlüsselung eine **Einwegfunktion**
  – Was ist zu beachten, um Ausgang des Münzwurfs nachträglich belegen zu können?
- **Einer der Mitspieler erfährt den Ausgang des Spiels zwangsläufig als erster:**
  – Was tun, wenn er bei einer Niederlage das Spiel abbricht, ohne es aufzulösen?
  – *„Wer aufhört verliert!"* – Ist das eine Lösung? (juristisch: Unschuldsvermutung!)

Kerberos "Security Service"

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

erstes grosses Client-Server-Campusnetz, ca. 25.000 Benutzer, 1.200 Computer

- Wird u.a. im DCE der OSF eingesetzt

Distributed Computing Environment

Open Software Foundation

- Basiert auf Needham-Schroeder-Protokoll mit symmetrischen Schlüsseln (i.a. DES)
- Public domain; es gibt aber kommerzielle Varianten

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 nur verschlüsselt und 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 personengebunden, nur in Verbindung mit Authentitätsnachweis gültig

- Gültigkeit von Tickets / Sitzungsschlüsseln zeitlich befristet

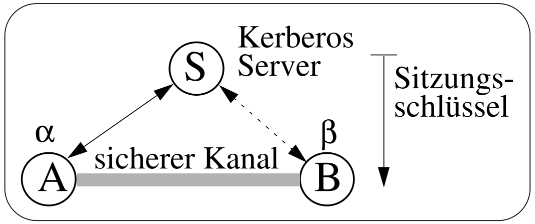
- Drei Sicherheitsstufen (dreiköpfiger Höllenhund!)
    (1) Authentifizierung nur bei Einrichtung eines Kommunikationskanals
    (2) Authentifizierung bei jeder Nachricht zwischen A und B
    (3) zusätzlich Verschlüsselung der Nachrichten

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); Version 5 im Prinzip nur leicht unterschiedlich

session key

Zeit-stempel

Gültigkeitsdauer des session key

A --> S: A, B

S --> A: $\{B, K, Ts, \Delta, \{A, K, Ts, \Delta\}_\beta\}_\alpha$

Geheime Schlüssel von A bzw. B

A --> B: $\{A, K, Ts, \Delta\}_\beta, \{A, Ta\}_K$

B sieht: A kennt K

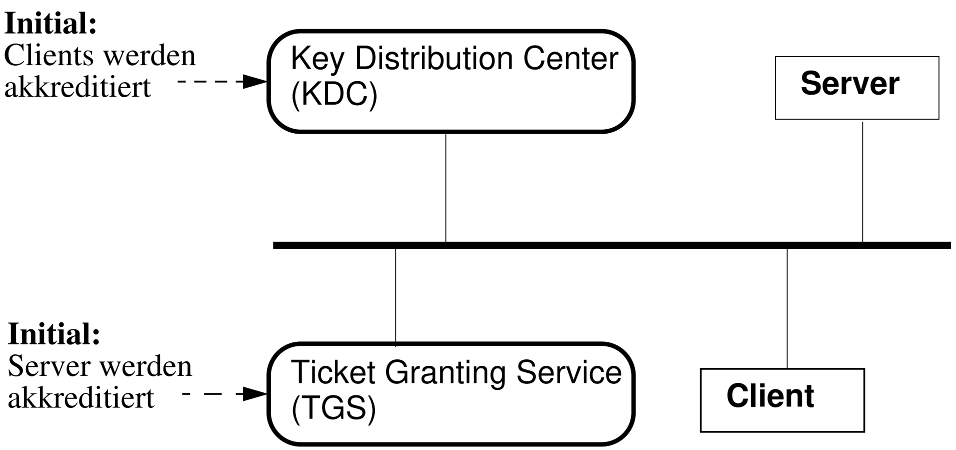
B --> A: $\{Ta + 1\}_K$

Hiermit authentifiz. sich B gegenüber A (kein anderer konnte Ta entschlüsseln)

- 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



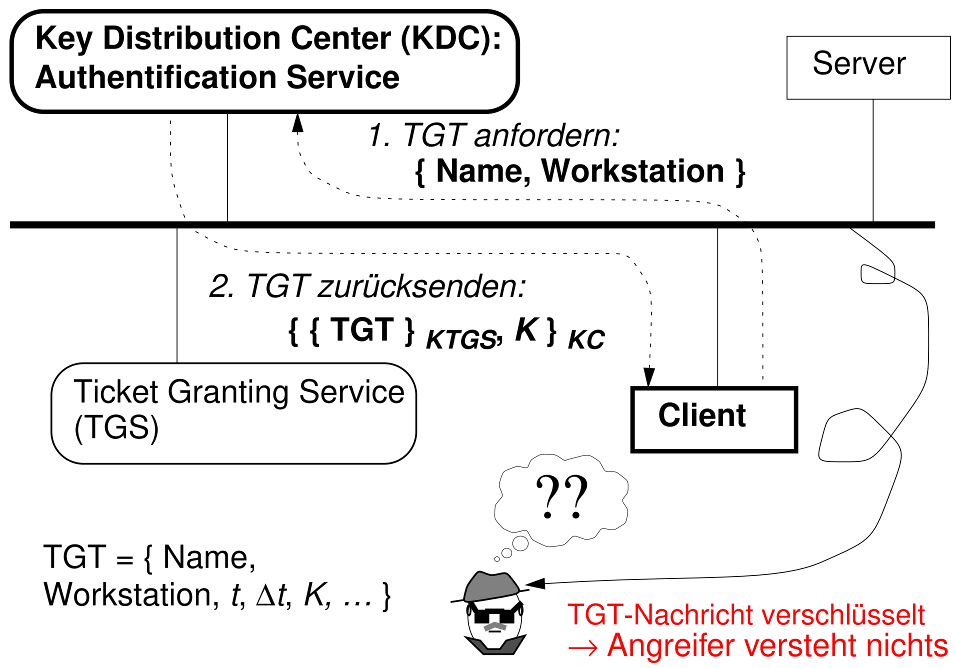
- Benutzer und deren Passwörter (= Schlüssel) werden dem KDC bekannt gemacht

- 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

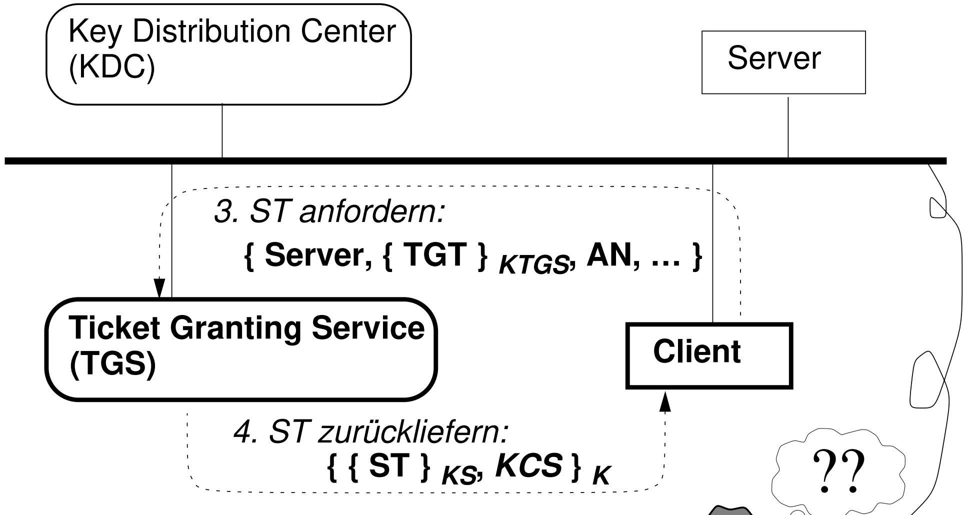
- Client erwirbt zunächst ein Ticket Granting Ticket (TGT)



- Client an KDC: sendet { Name, Workstation } im Klartext
- KDC: wählt K; erstellt TGT = { Name, Workstation, 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 Workstation nicht und wird sofort wieder gelöscht
--> TGT ist verschlüsselt, nur für Zeitspanne Δt gültig, geht nur an Client

Kerberos: Service Ticket erwerben



AN = { t, Name, Workstation, … }$_K$

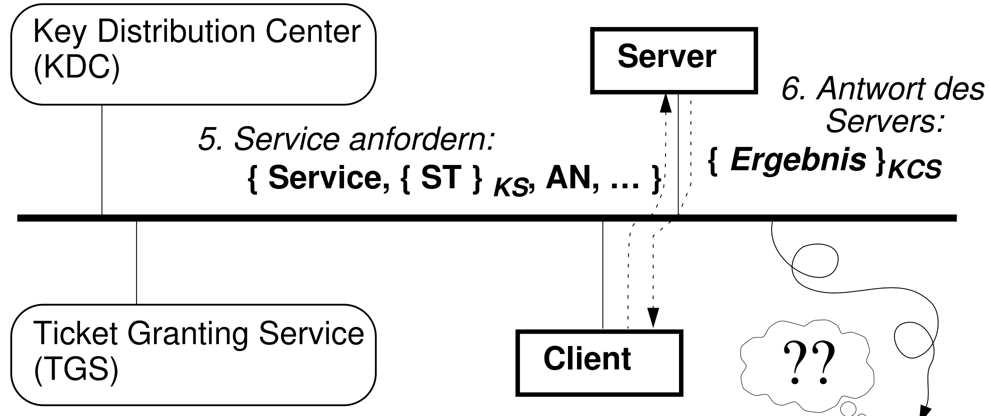
ST = { Name, Workstation, t, Δt, KCS, … }

- Client: erstellt Authentizitätsnachweis AN = { t, Name, Workstation, … }$_K$
- Client an TGS: sendet { 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, Workstation, t, Δt, KCS, … }
- TGS an Client: sendet { { 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, und nur an bezeichneter Workstation

Kerberos: Nutzung des Service



AN = { t, Name, Workstation, … }$_{KCS}$

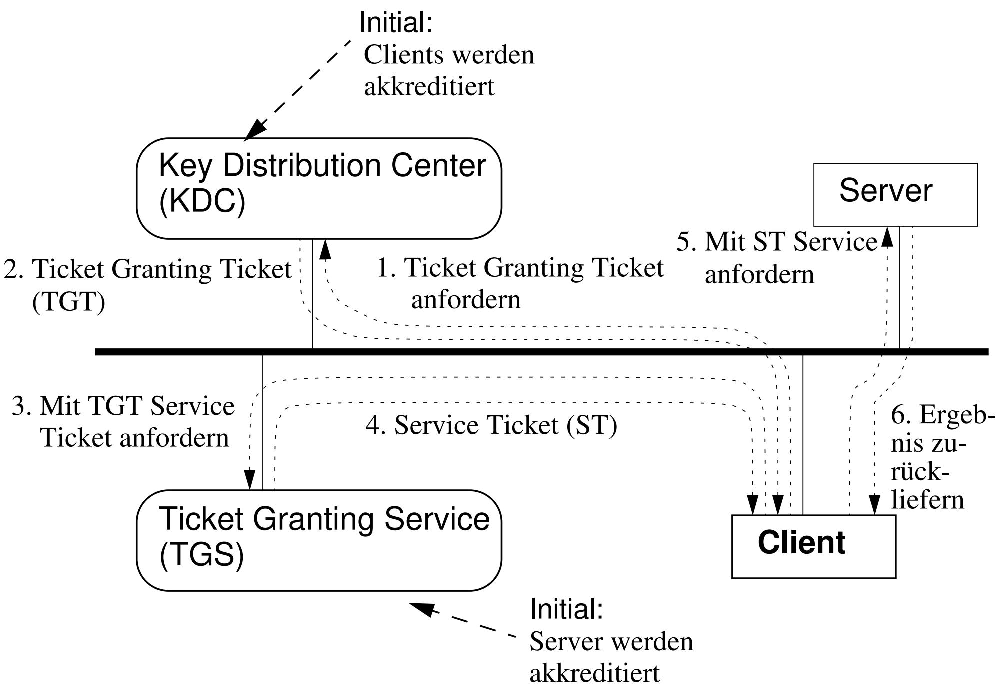
ST = { Name, Workstation, t, Δt, KCS, … }

- Client: erstellt Authentizitätsnachweis AN = { t, Name, Workstation, … }$_{KCS}$
- 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!)
  - “Read-only”-Netzwerkzugang
- 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
- 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
  - nicht ohne weiteres aus den USA exportierbar (Verwendung von DES)

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) funktioniert 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?