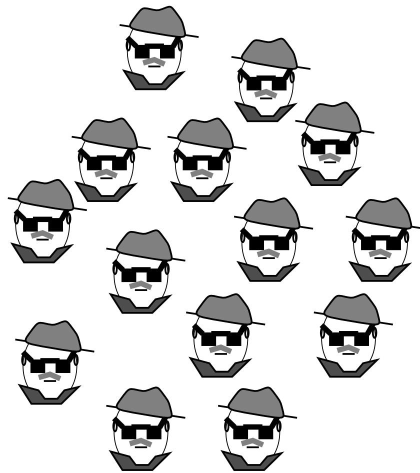
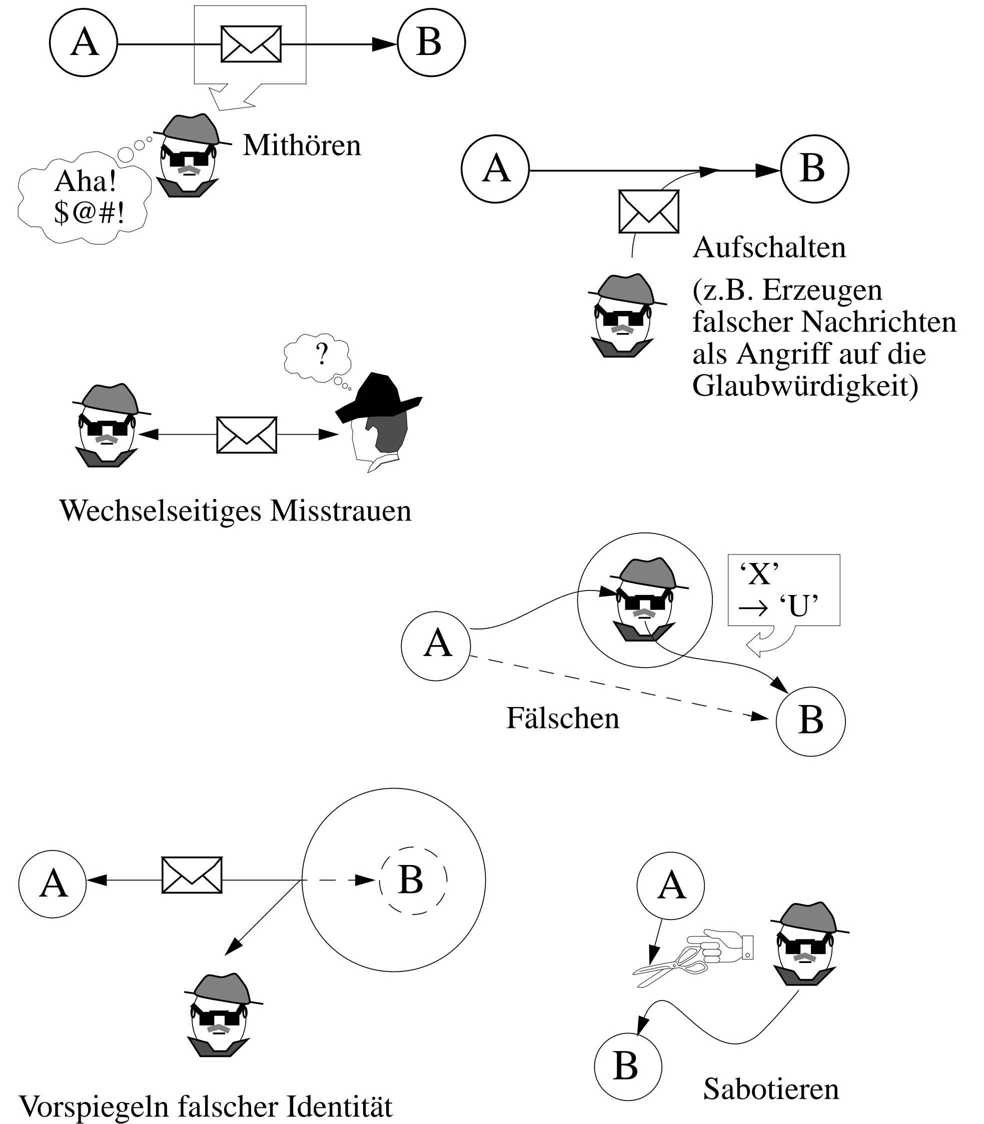
Sicherheit



Sicherheit in verteilten Systemen



Sicherheit: Anforderungen

- Autorisierung / Zugriffsschutz
  - Einschränkung der Nutzung auf den Kreis der Berechtigten
- Vertraulichkeit
  - Daten / Nachrichteninhalt gegen Lesen Unberechtigter schütztzen
  - 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. 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!

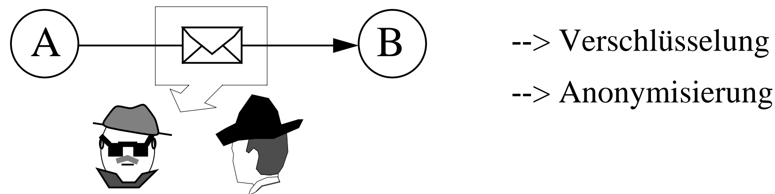
Typische Techniken und "Sicherheitsdienste":

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

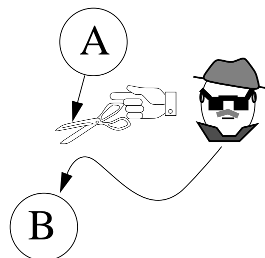
} Hierfür Kryptosysteme und Protokolle als "Security Service", z.B. *Kerberos*

Angriffsformen

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



- *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)

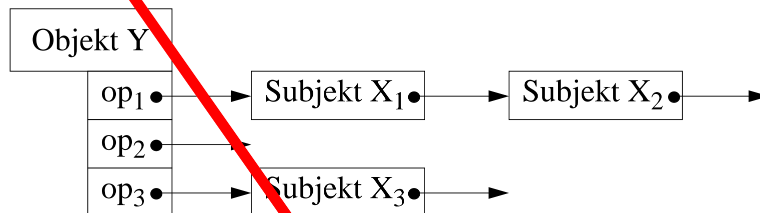
Objekte (Spalten), *Subjekte* (Zeilen)

	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
  - “Türsteherprinzip”: vgl. z.B. UNIX-Dateischutzattribute



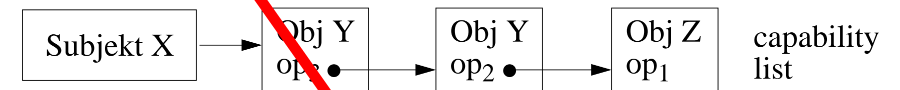
  - Subjekt X_1 und Subjekt X_2 dürfen op_1 auf Objekt Y ausführen; Subjekt X_3 darf Operation op_3 auf Objekt Y ausführen

- 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”



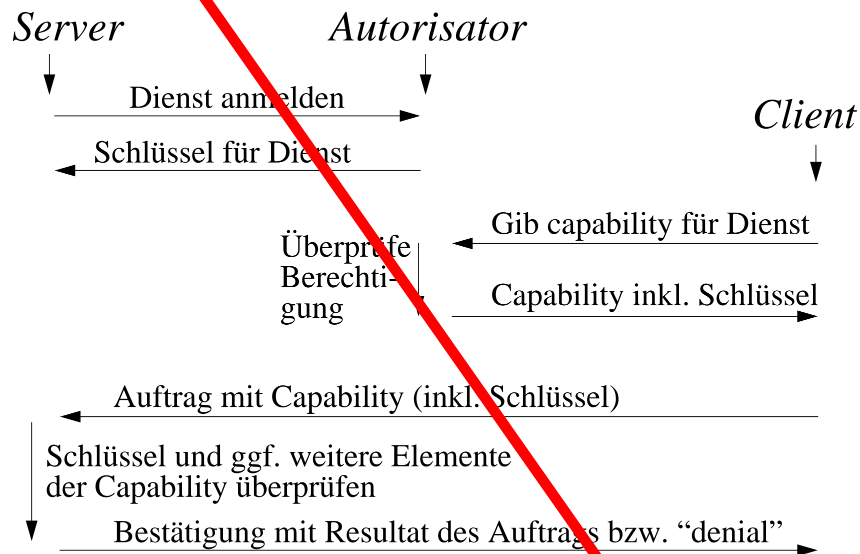
  - Subjekt X darf auf Objekt Y die Operationen op_2, op_3 ausführen und auf Objekt Z die Operation op_1 ausführen

- 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

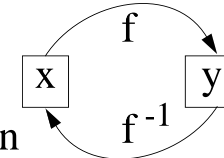
  - Abhörgefahr (kein Passwortaustausch im Klartext; Speicherung des Passworts nur in codierter Form, so dass Invertierung prakt. unmöglich
  - Replay-Attacke (Gegenmassnahme: Einmalpasswörter)

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

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

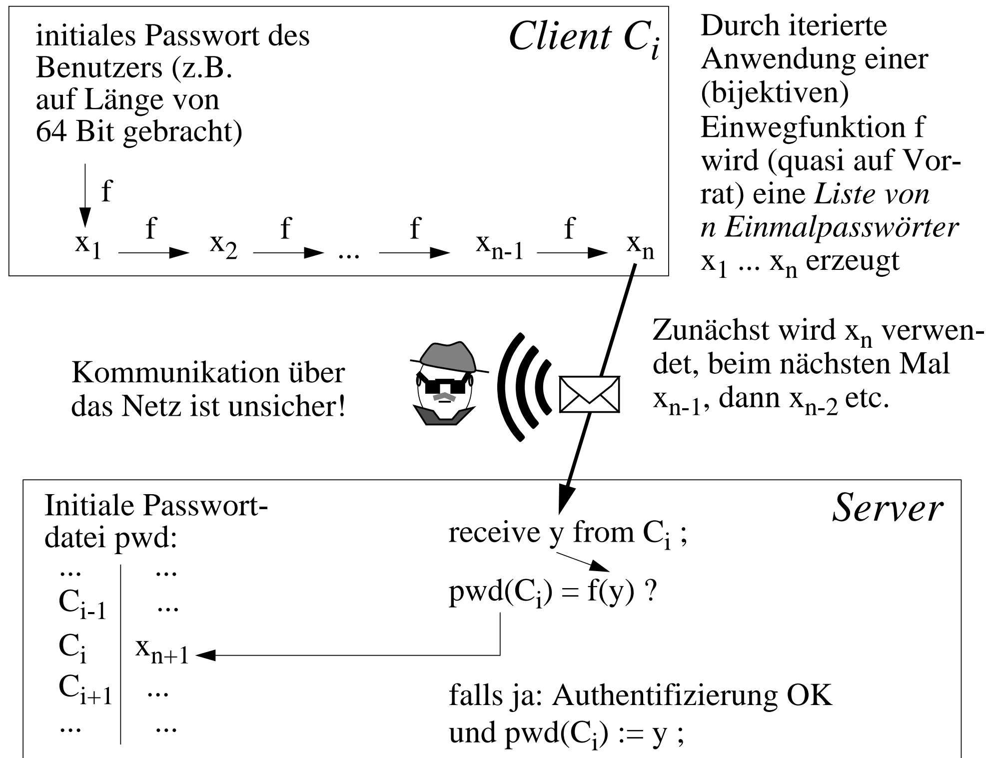
- Einwegfunktionen erscheinen zunächst ziemlich sinnlos: Ein zu $y = f(x)$ verschlüsselter Text x kann nie wieder entschlüsselt werden!

  ==> 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

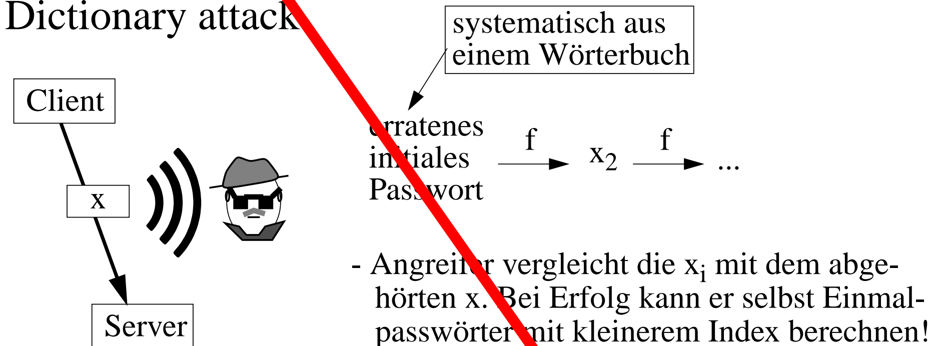
keyinfo takes an optional user name and displays the user’ s current sequence number and seed found in the S/Key database. The command can be useful when generating a list of passwords for use on a field trip.

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

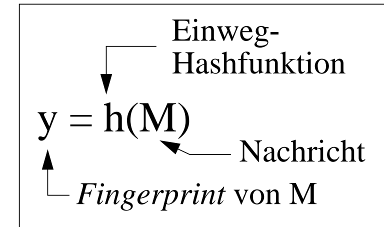
- Der Angreifer spielt Server und fragt den Client nach dem i-ten Passwort x_i , wobei i kleiner als der Index j des vom echten Server erwarteten nächsten Passworts x_j ist. Aus x_i berechnet er dann durch (ggf. mehrfache) Anwendung von f das nächste Passwort x_j .

- 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



- *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

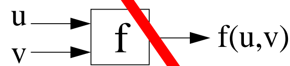
3) Digitale Unterschrift nur auf den Fingerprint anwenden (effizienter!)

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:

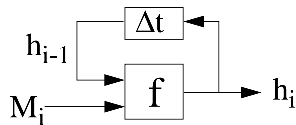
Benutze eine zweistellige Einwegfunktion f



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

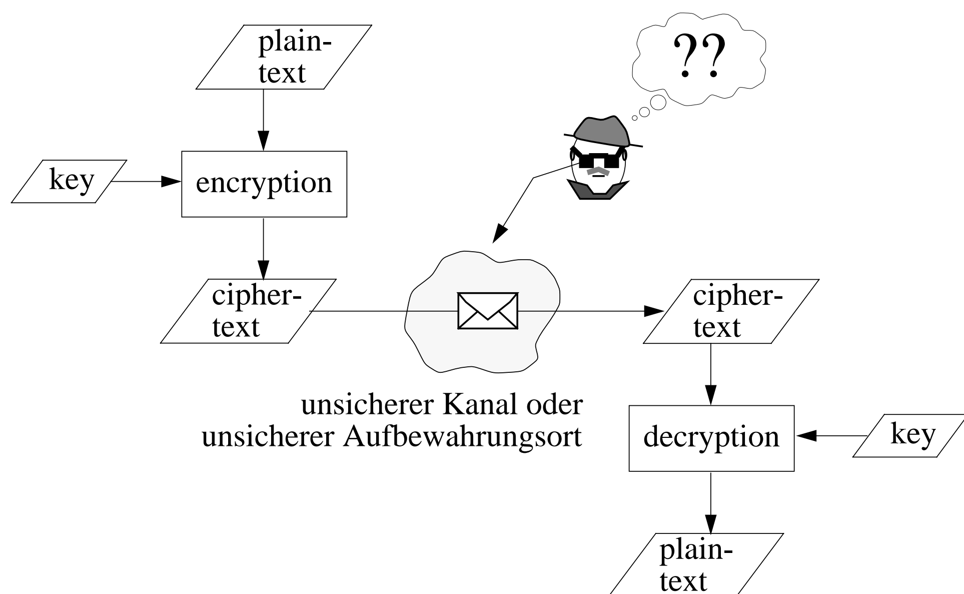
This document describes the MD5 message-digest algorithm. The algorithm takes as input a message of arbitrary length and produces as output a 128-bit "fingerprint" or "message digest" of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message 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 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 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
```

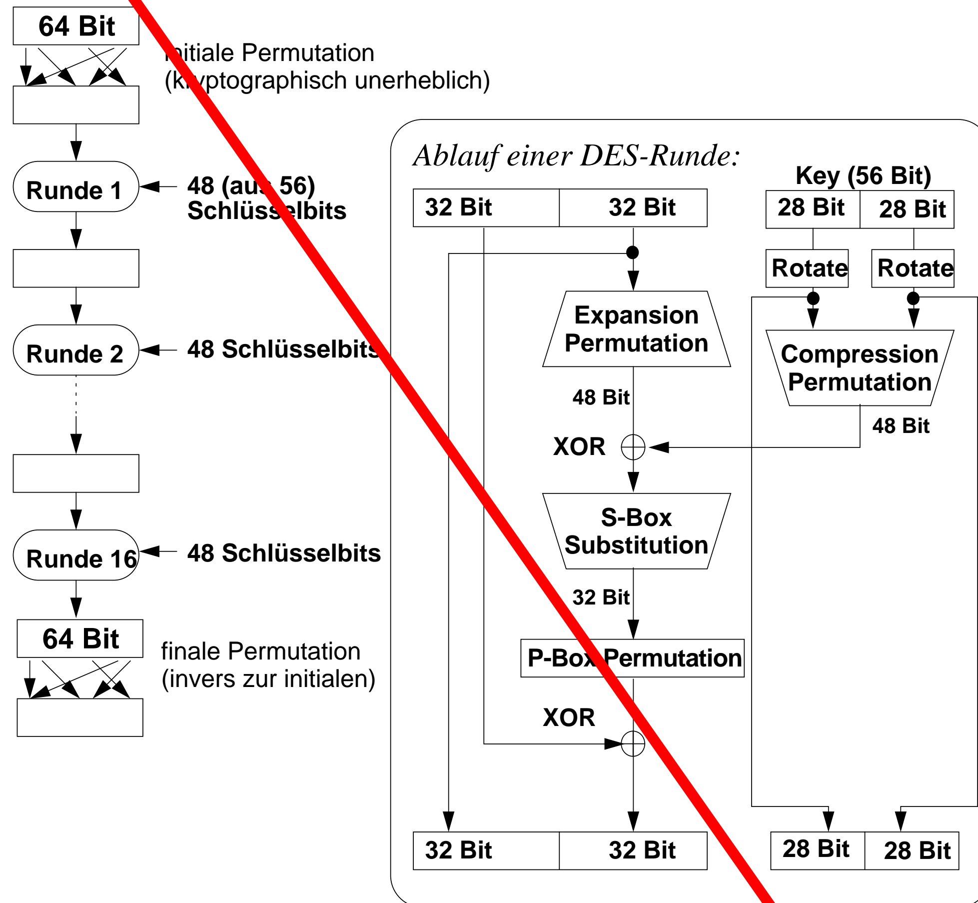
--> Präprozessor-Anweisung zur bedingten Übersetzung

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.

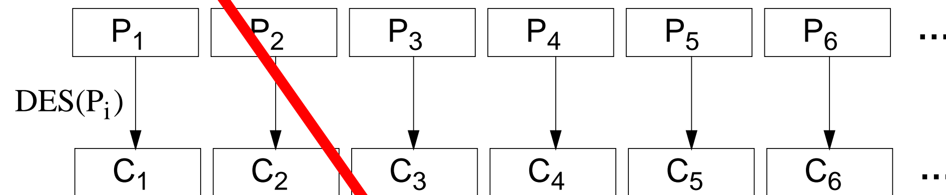
The des command is provided to promote secure exchange of data in a standard fashion.

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

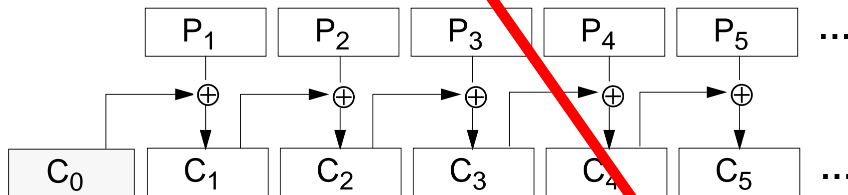
- DES verschlüsselt blockweise (je 64 Bit); was tun bei langen Klartexten?

- 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: $C_i = DES(P_i \textbf{ XOR } C_{i-1})$;
  - Entschlüsselung: $P_i = DES^{-1}(C_i) \textbf{ XOR } C_{i-1}$
  - 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

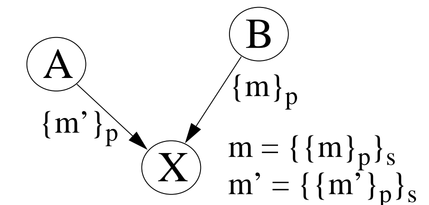
- Für jeden Prozess X existiert ein Paar (p,s)

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

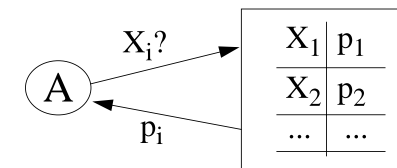
  s = *secret key* ← zum *Entschlüsseln* von mit p verschlüsselten Nachrichten
  (oder “private” key)

- Jeder Prozess, der an X sendet, kennt p

- Nur X selbst kennt s



- *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

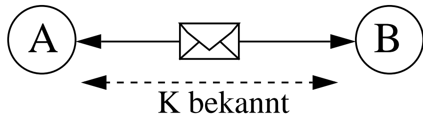
- Es gibt Alternativen zu DES
  - IDEA(International Data Encryption Algorithm):
    - ETH Zürich (X. Lai, J. Massey) und Ascom
    - Schlüssellänge 128 Bit
    - keine Export- / Anwendungsrestriktionen
    - Prinzip analog zu DES
    - es gibt Chip mit einer Verschlüsselungsrate von 177 Mb/s
    - wird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



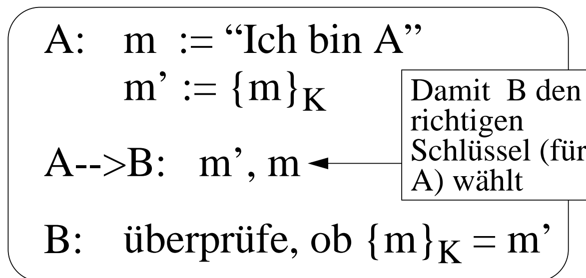
Sei K der zwischen A und B vereinbarte (und geheim-zuhaltende!) Schlüssel

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"

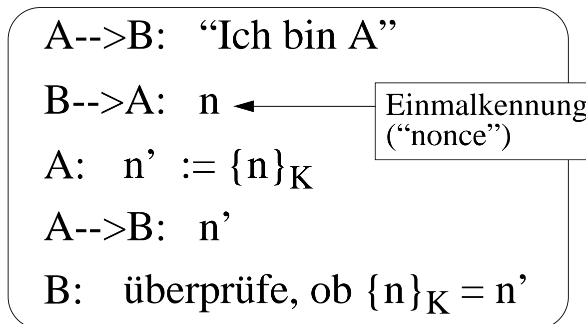
Bemerkung: Oft ist eine *gegenseitige* Authentifizierung nötig

1. Verfahren:



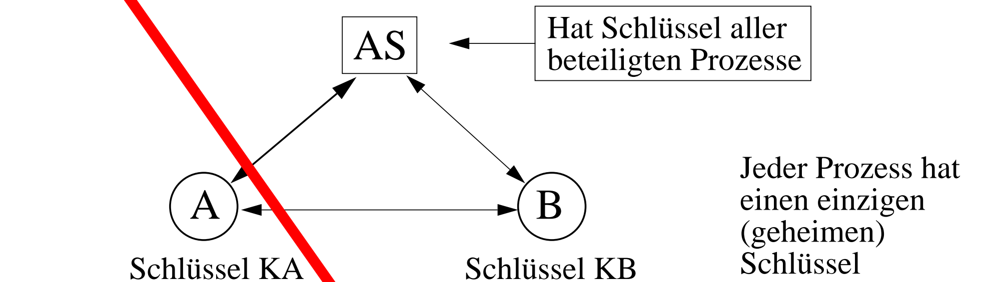
- *Idee*: Überprüfe die Fähigkeit, Nachrichten mit einem geheimen Schlüssel zu kodieren.
- *Nachteil*: Möglichkeit von replays durch Abhören

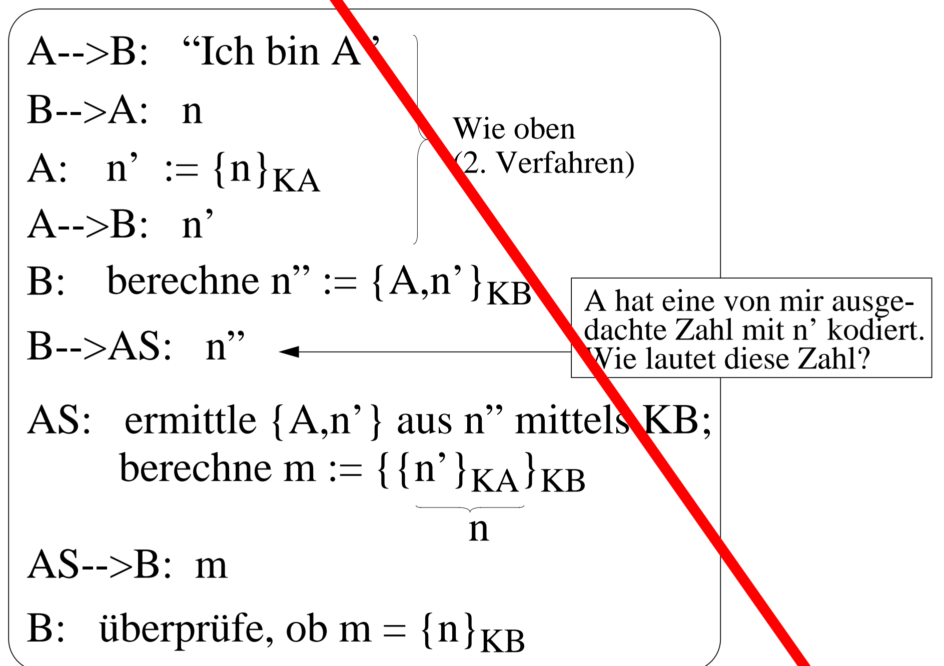
2. Verfahren:



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

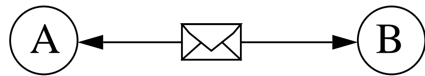
Authentifizierung mit Authentifizierungsserver



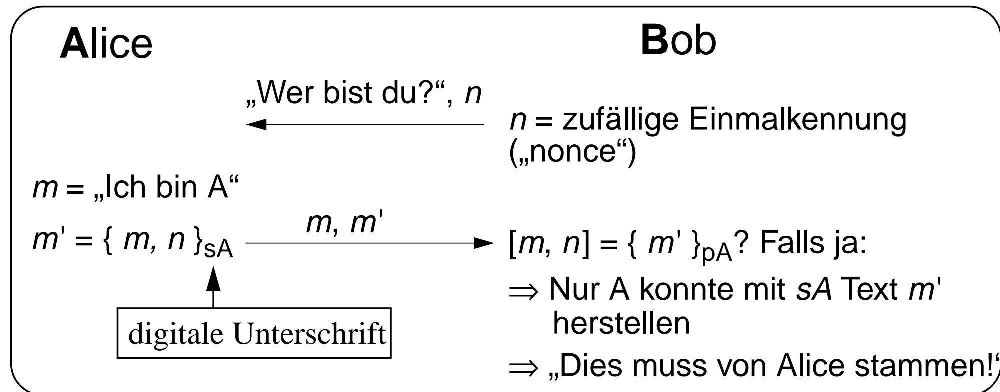


Bemerkung: Authentifizierungsserver geniesst Vertrauen aller Prozesse und ist gleichzeitig die zentrale *Schwachstelle*: Angreifer könnte in Besitz aller keys kommen!

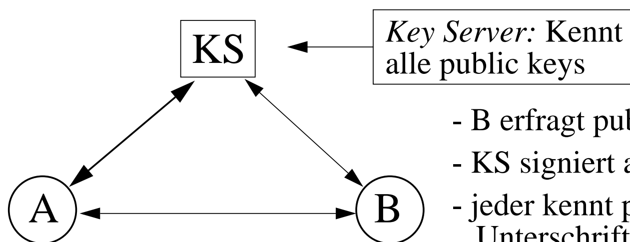
Authentifizierung mit asymmetrischen Schlüsseln



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



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

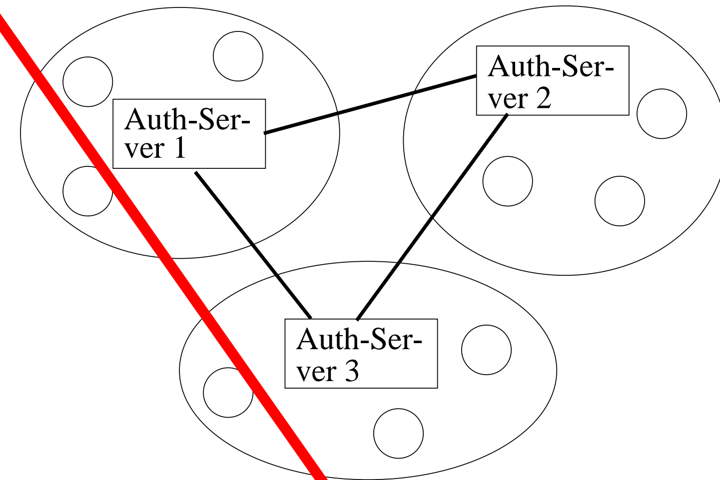


Key Server: Kennt alle public keys

- 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

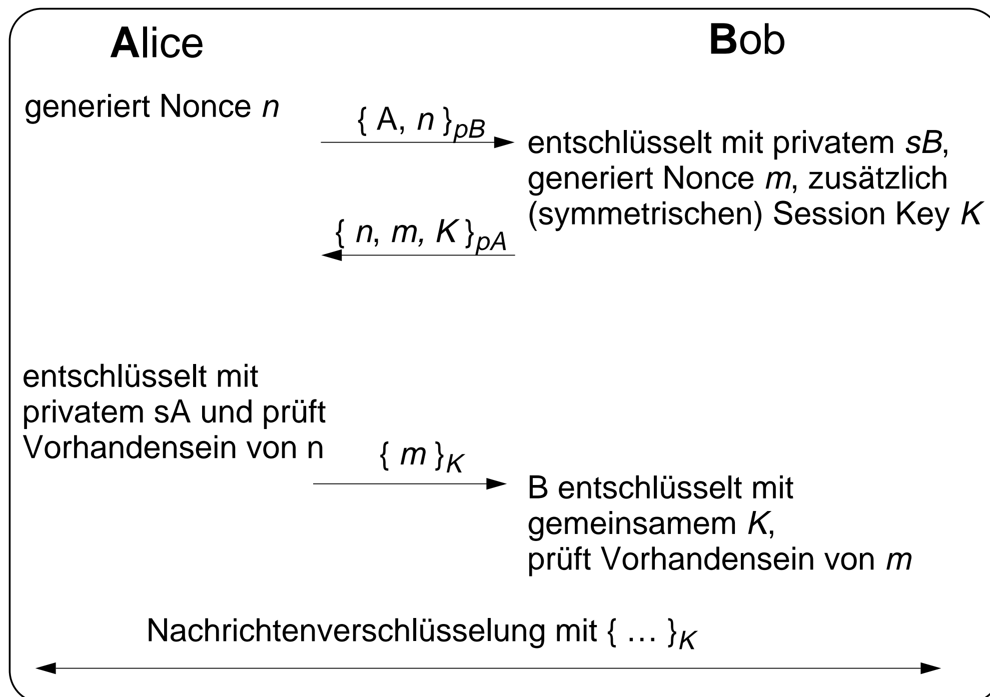
Dezentrale Authentifizierungsserver



- 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

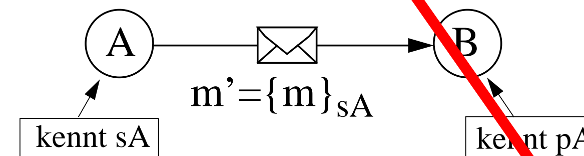
- Im Prinzip möglich wie oben beschrieben nacheinander in beide Richtungen
- Gleich beides zusammen erledigen ist aber effizienter!
- Hier zusätzlich: Vereinbarung eines symmetrischen “session keys” K, der nach der Authentifizierung zur effizienten Verschlüsselung benutzt wird
- Voraussetzung: A und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



Digitale Unterschiften

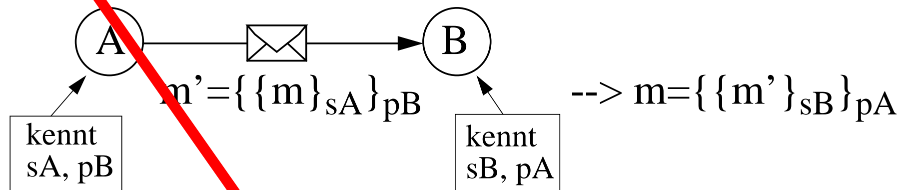
- Zweck: Schliessen von “elektronischen Verträgen”, z.B.:
  - Unterzeichnen von Dokumenten zum Nachweis der Urheberschaft
  - Ausstellen elektronischer Schecks
- Gewünschte Eigenschaften (wie gewöhnliche Unterschrift):
  - *Authentizität* der Unterschrift und *Nichtabstreitbarkeit*: niemand sonst kann 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



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

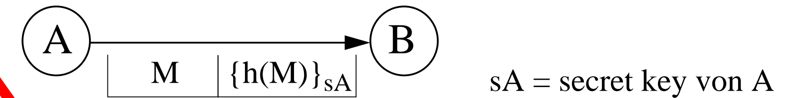
- Nur B kann die äussere Hülle entfernen
- Nur A kann die Nachricht mit sA verschlüsselt haben

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:

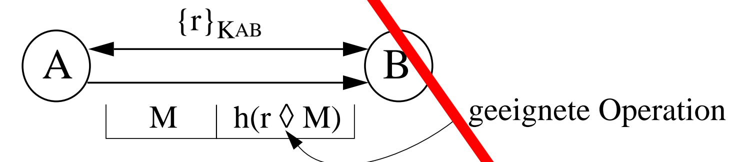


sA = secret key von A

- 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:

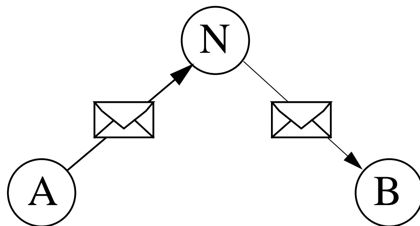
B sendet die gleiche Nachricht zurück, versehen mit B's digitaler Unterschrift (und verschlüsselt)

- 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.

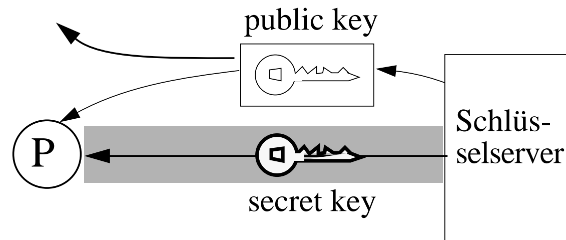
Absendezeit:
- lokale Uhrzeit
- globale Zeitapproximation aus Zeitservice (z.B. NTP-Protokoll)
- Empfängerzeit vorher erfragen

- 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

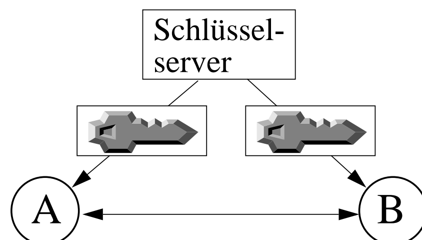
- Zur Vergabe eines Paares von public-, secret-keys:



- 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”)

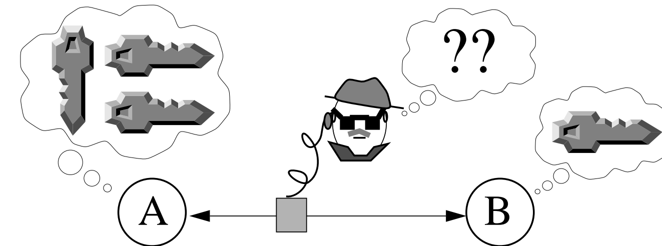
--> erhöhte Sicherheit gegen Angreifer



- 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

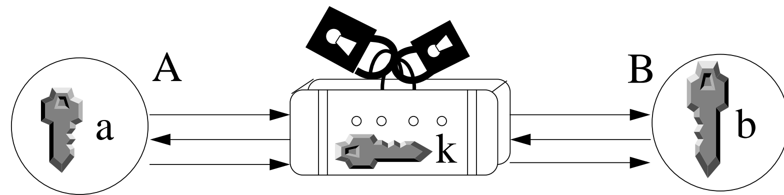


- Problem: A und B wollen sich über einen unsicheren Kanal auf einen gemeinsamen Schlüssel einigen, ohne einen Schlüsselserver zu verwenden

- 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: $\{k\}_a$ — a und b sind "lokal erfunden"
4. B verschlüsselt dies mit seinem Schlüssel b
5. B-->A: $\{\{k\}_a\}_b$
6. A entschlüsselt mit seinem Schlüssel a:
   $\{\{\{k\}_a\}_b\}_{\bar{a}} = \{\{\{k\}_a\}_{\bar{a}}\}_b = \{k\}_b$

   Forderung! (zur ersten Gleichung)

   Bezeichne $\bar{x}$ den zu x inversen Schlüssel (oft: $\bar{x}$=x)
7. A-->B: $\{k\}_b$
8. B entschlüsselt mit seinem Schlüssel: $\{\{k\}_b\}_{\bar{b}} = k$ — gemeinsames Geheimnis

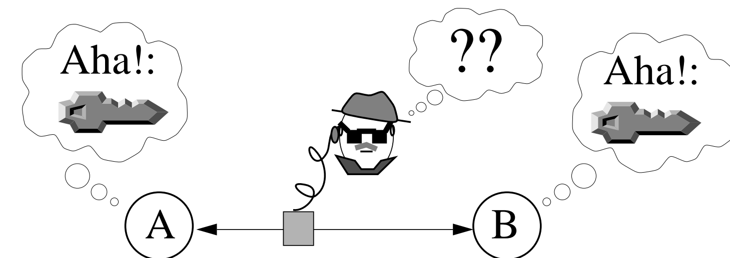
Beachte: k wird nie offen transportiert!

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