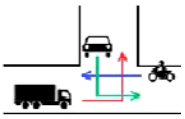
Wechselseitiger Ausschluss

Wechselseitiger Ausschluss
(Mutual Exclusion, „Mutex“)

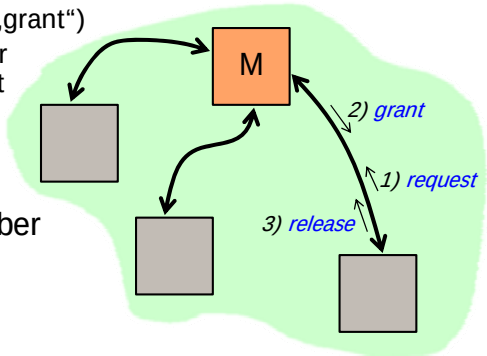
- Koordination, wenn viele wollen, aber nur einer darf
- „Streit“ um exklusives Betriebsmittel, z.B.:
  - konkrete Ressource wie gemeinsamer Datenbus
  - abstrakte Ressource wie etwa ein „Termin“ in einem (verteilten) Terminkalendersystem
  - „kritischer Abschnitt“ in einem nebenläufigen Programm



- Es gibt klassische Lösungen bei shared memory
  - z.B. Semaphore und Monitore (→ Betriebssystemtheorie)
  - sind in unserem Kontext aber nicht interessant

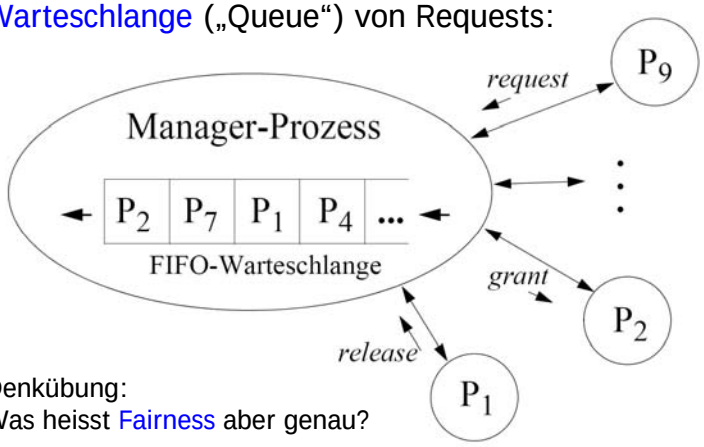
Zentraler Manager?

- Hier: Nachrichtenbasiertes System konkurrierender Prozesse
- Idee: Manager, der die Ressource (in fairer Weise!) zuordnet
  - ein Prozess bewirbt sich um die Ressource mit „request“
  - wartet dann auf Erlaubnis („grant“)
  - teilt schliesslich Freigabe der Ressource dem Manager mit „release“ mit
- Vergleichsweise einfach und wenige Nachrichten, aber
  - potentieller Engpass
  - single point of failure



Globale Warteschlange garantiert Fairness

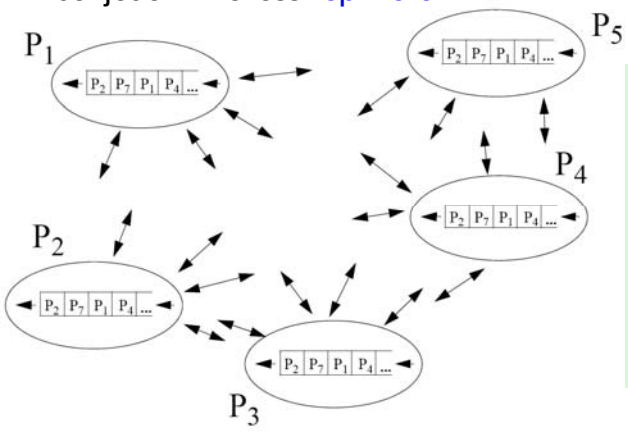
- Der Manager-Prozess hält eine (zeitlich geordnete) Warteschlange („Queue“) von Requests:



- Denkübung:
  Was heisst Fairness aber genau?

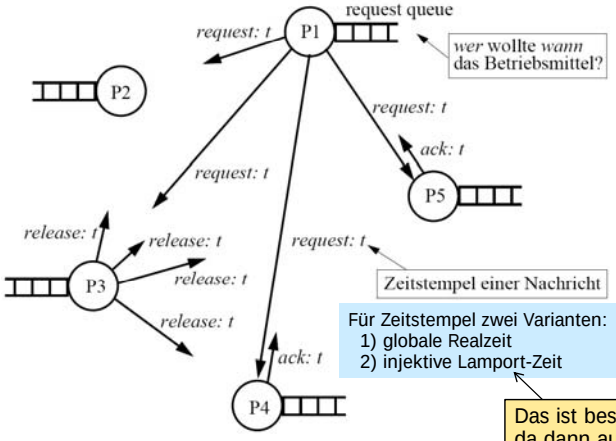
Replizierte Warteschlange?

- Idee für dezentrale Lösung: globale Warteschlange bei jedem Prozess replizieren



- Alle Prozesse sollen die gleiche Sicht der „virtuell globalen“ Warte-schlange haben
- Konsistenz wird mit (vielen) Nachrichten und logischer Zeit erreicht (→ nächste slides)

Synchronisation der Warte-schlangen mit Zeitstempeln



Für Zeitstempel zwei Varianten:
1) globale Realzeit
2) injektive Lamport-Zeit

Das ist besonders interessant, da dann auf synchronisierte Uhren verzichtet werden kann

- Voraussetzung: FIFO-Kommunikation
- Alle Nachrichten tragen (eindeutige!) Zeitstempel
- Request- und Release-Nachrichten immer an alle senden (FIFO-Broadcast)
- Requests werden bestätigt („ack“)

Der Algorithmus (Lamport 1978)

1) Bewerbung um Betriebsmittel: Request mit Zeitstempel und Absender an alle senden und in eigene Queue einfügen
2) Bei Empfang eines Request: Request in eigene Queue einfügen, ack versenden
3) Bei Freigabe des Betriebsmittels: Aus eigener Queue entfernen und Release an alle versenden
4) Bei Empfang eines Release: Zugehörigen Request aus eigener Queue entfernen
5) Ein Prozess darf das Betriebsmittel nutzen, wenn:
   a) der eigene Request der früheste in seiner Queue ist
   b) und er bereits von jedem anderen Prozess (irgendeine) spätere Nachricht bekommen hat

(Frühester Request ist global eindeutig ⇒ die beiden Bedingungen garantieren, dass kein früherer Request mehr kommt (wieso?))

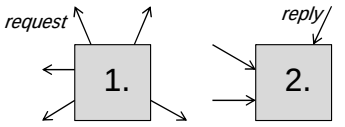
Denkübungen:
- Wo geht die Uhrenbedingung der Lamport-Zeit ein?
- Wieso ist FIFO notwendig?

Wieso sind garantiert:
1) Safety (zu jedem Zeitpunkt höchstens einer),
2) Fairness (jeder Request wird „schliesslich“ erfüllt)?

3(n-1) Nachrichten pro Bewerbung (n = Zahl der Prozesse)

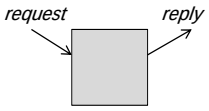
Ein anderer verteilter Mutex-Algorithmus (Ricart / Agrawala, 1981)

- Nur 2(n-1) Nachrichten (Reply übernimmt Rolle von Release und ack)



1. Sende Request (mit Zeitstempel!) an alle n-1 anderen
2. Dann auf n-1 Replies warten, danach Betriebsmittel nutzen

- Bei Eintreffen einer Request-Nachricht:



  - wenn nicht selbst beworben oder der Sender „ältere Rechte“ (bzgl. logischer Zeit) hat, dann Reply sofort schicken
  - ansonsten Reply erst später (im Sinne von Release) schicken, nach Erfüllen des eigenen Requests (d.h. exklusivem Zugriff)

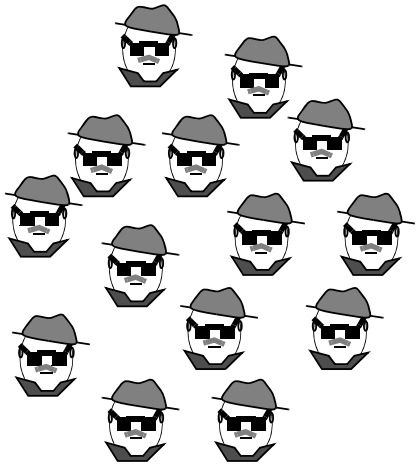
  Nur älteste Bewerbung setzt sich überall durch!

Denkübungen: Safety? Fairness? Deadlockfreiheit? FIFO-Kanäle notwendig?

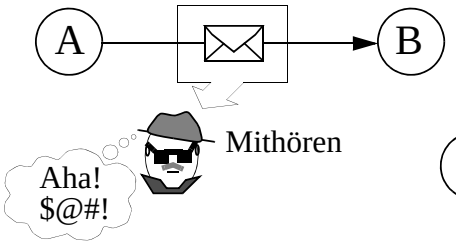
Resümee (8a)

- Wechselseitiger Ausschluss (mit logischer Zeit)
  - replizierte Warteschlangen von Lamport (request, reply, ack)
  - anderes Verfahren: verzögerte Replies (mit 2(n-1) Nachrichten)
  - Korrektheitsargumente? (Safety, Fairness,...)

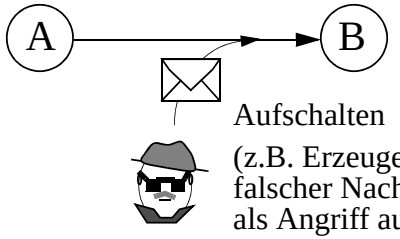
Sicherheit



Sicherheit in verteilten Systemen

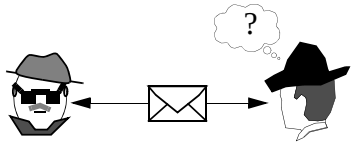


Mithören

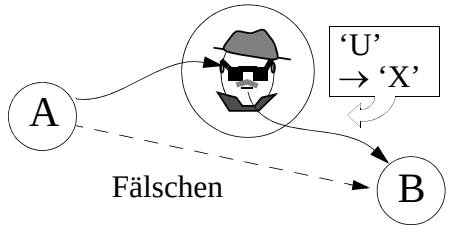


Aufschalten

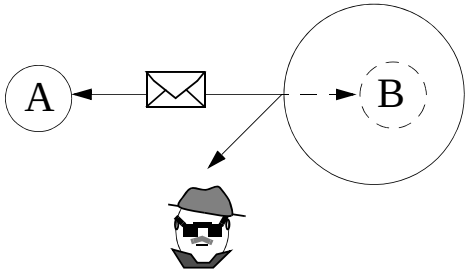
(z.B. Erzeugen falscher Nachrichten als Angriff auf die Glaubwürdigkeit)



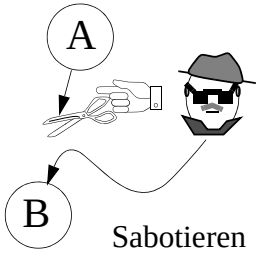
Wechselseitiges Misstrauen



Fälschen



Vorspiegeln falscher Identität



Sabotieren

Sicherheit: Anforderungen

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

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

Sicherheit: Verteilungsaspekte

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

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

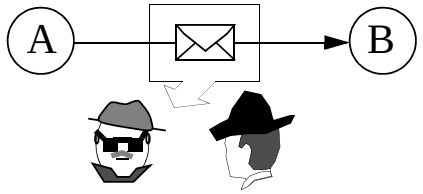
Typische Techniken und "Sicherheitsdienste":

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

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

Angriffsformen

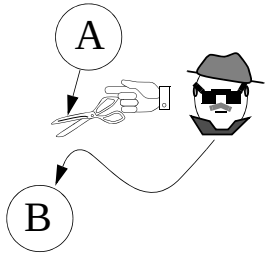
- *Passive Angriffe*: Beobachten der Kommunikation
  - Inhalt von Nachrichten in Erfahrung bringen
  - Kommunikationsverhalten analysieren (“wer mit wem wie oft?”)



→ 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. durch Überfluten mit Nachrichten

Authentizität

…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 rauen 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*
  - ist er wirklich der, der er vorgibt zu sein?
  - darf ich als Server daher ihm (?) den Zugriff gewähren?
- Authentizität eines *Dienstes*
  - 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 ausgeben?
- Authentizität *gespeicherter Daten*
  - ist dies wirklich der Vertragstext, den wir gemeinsam elektronisch hinterlegt haben?
  - hat der Autor Casimir von Hinkelstein wirklich *das* geschrieben?
  - ist das Foto nicht eine Fälschung?
  - ist dieser elektronische Schlüssel wirklich echt?

Hilfsmittel zur Authentifizierung

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

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

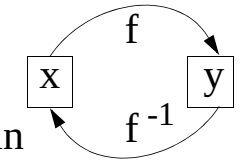
- Peer-Authentifizierung mit *Passwort*
  - typischerweise zur Authentifizierung eines Benutzers ("Client") zum Schutz des Dienstes vor unbefugter Benutzung (Autorisierung)
  - Kenntnis des Passworts gilt als Identitätsbeweis (ist das gerechtfertigt?)

- 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
  - 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



zeitaufwändig (→ praktisch nicht durchführbar)

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

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

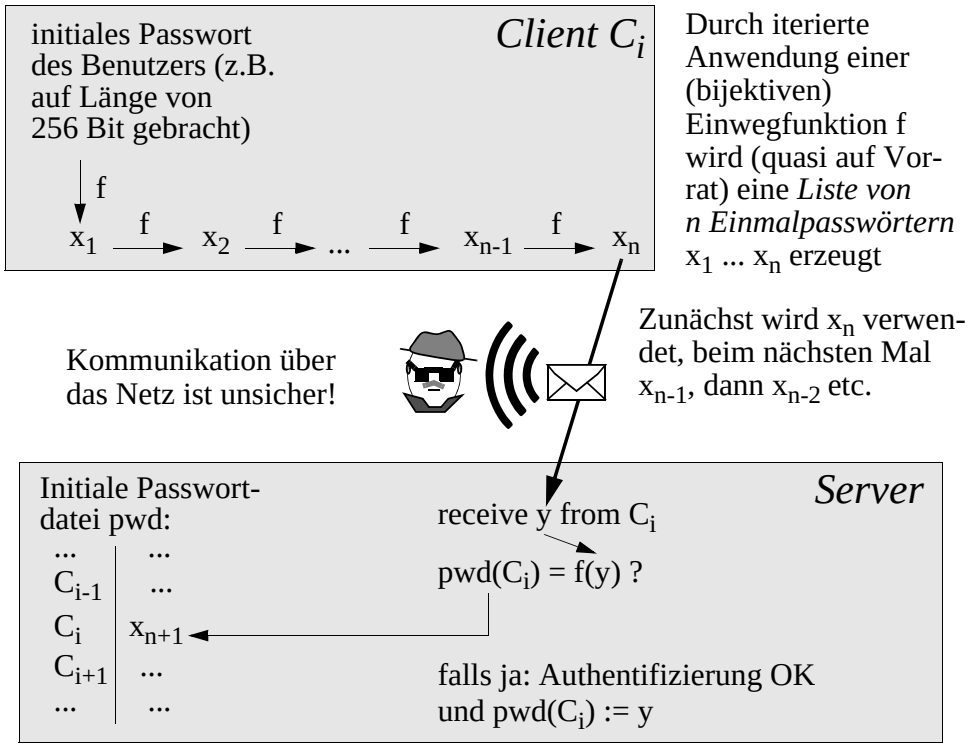
- Einwegfunktionen erscheinen zunächst ziemlich sinnlos: Ein zu $y = f(x)$ verschlüsselter Text x kann nie wieder entschlüsselt werden!

  ⇒ 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.Allg. 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.Allg. schwierig. (Aber: k-te Wurzel einfach, wenn Primzerlegung von $m = pq$ bekannt → trap-door!)

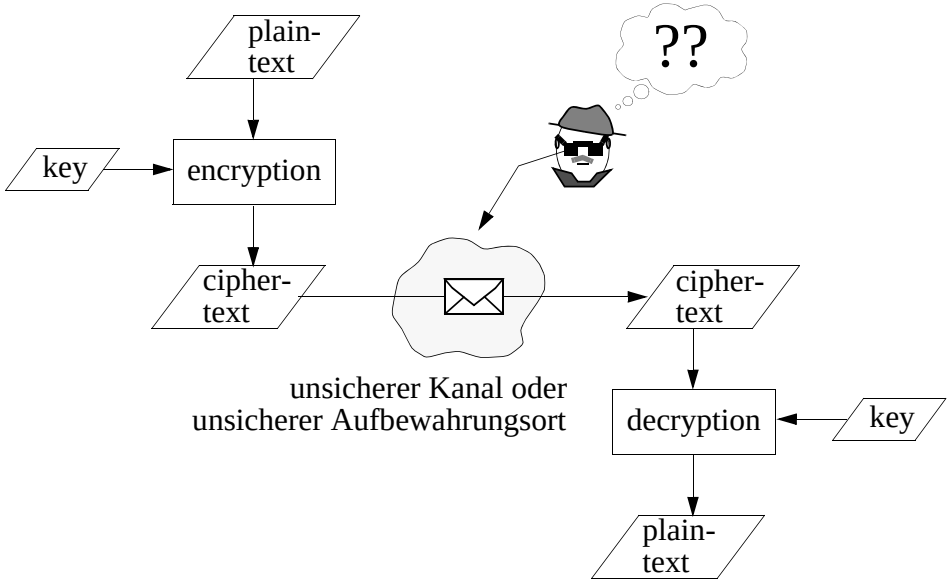
Einmalpasswörter mit Einwegfunktionen

- 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 nur das *vergangene* Passwort vermerkt
- Einwegfunktion f muss nicht geheimgehalten werden
- Realisiert z.B. im S/KEY-Verfahren (RFC 1760)

Kryptosysteme



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

Kryptosysteme (2)

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

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

---

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

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

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

One-Time Pads

- “Perfektes” (symmetrisches) Kryptosystem
  - Denkübung: unter welchen Voraussetzungen?

- Prinzip: Wähle zufällige Sequenz von Schlüsselbits
  - *Verschlüsselung:* Schlüsseltext = Klartext XOR Schlüsselbitsequenz
  - *Entschlüsselung:* Klartext = Schlüsseltext XOR Schlüsselbitsequenz
  - Begründung: ((a XOR b) XOR b) = a (für alle Bitbelegungen von a, b)

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 (hoher Bedarf an frischen Schlüsselbits, dadurch aufwändiger Schlüsselaustausch)
  - Synchronisationsproblem bei Übertragungsstörungen (wenn Empfang ausser Takt gerät ist Folgetext verloren)
  - nur für hohe Sicherheitsanforderungen gebräuchlich (z.B. “rotes Telefon”)

Pseudo-Zufallszahlen?

Security Loophole Found in Microsoft Windows

University of Haifa, 12 Nov 2007

A group of researchers in Israel notified Microsoft that they have discovered a security loophole in the Windows 2000 operating system.

The researchers say they have found a way to decipher how Windows' random number generator works, compute previous and future encryption keys used by a computer, and monitor private communication. The security loophole jeopardizes emails, passwords, and credit card numbers entered into a computer. "This is not a theoretical discovery," says Dr. Benny Pinkas from the Department of Computer Science at the University of Haifa, who headed the research initiative. "Anyone who exploits this security loophole can definitely access this information on other computers."

The researchers say the newer versions of Windows may also be vulnerable if Microsoft uses similar random number generator programs.

Asymmetrische Kryptosysteme

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

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

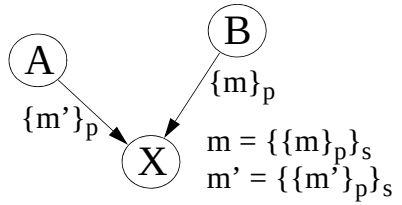
- Für jeden Prozess X existiert ein Paar (p,s)

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

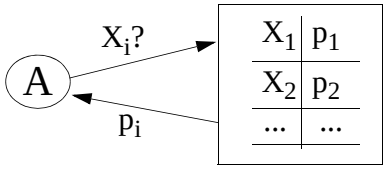
  s = *secret key* (oder "private" key) ← zum *Entschlüsseln* von mit p verschlüsselten Nachrichten

- Jeder Prozess, der an X sendet, kennt p

- Nur X selbst kennt s



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



  - Server muss vertrauenswürdig sein
  - Kommunikation zum Server darf nicht manipuliert sein

Asymmetrische Kryptosysteme (2)

- Sinnvolle *Forderungen*:

  1) m lässt sich nicht allein aus $\{m\}_p$ ermitteln

  2) s lässt sich aus p oder einer verschlüsselten, bekannten Nachricht nicht (mit vertretbarem Aufwand) ableiten

  3) $m = \{\{m\}_p\}_s$

  4) evtl. 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
  - das Kryptosystem muss demgegenüber robust sein

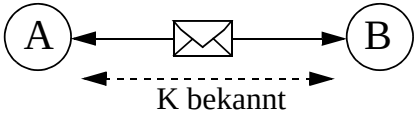
- Vorteil gegenüber symmetrischen Verfahren: vereinfachter Schlüsselaustausch
  - jeder darf den übermittelten public key p mithören
  - secret key s braucht grundsätzlich nie anderen mitgeteilt zu werden
  - bei n Teilnehmern genügen 2n Schlüssel (statt $O(n^2)$ bei sym. Schlüsseln)

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

sA bzw. pA secret bzw. public key von A

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

Authentifizierung mit symmetrischen Schlüsseln



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

Problem: B soll die Authentizität von A feststellen.

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

Bemerkung: Oft ist eine *gegenseitige* Authentifizierung nötig

1. Verfahren:

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

A→B: m', m

Damit B den richtigen Schlüssel (für A) wählt

B: überprüfe, ob $\{m\}_K$ = m'

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

2. Verfahren:

A→B: "Ich bin A"

B→A: n

Einmalkennung ("nonce")

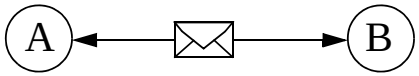
A: n' := $\{n\}_K$

A→B: n'

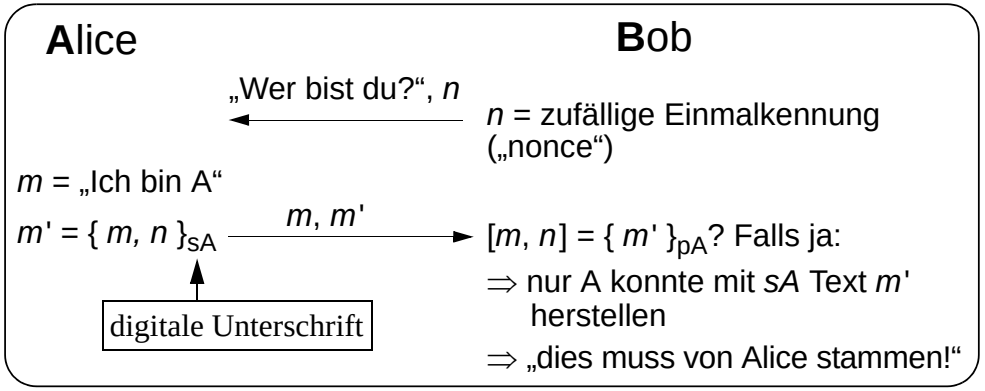
B: überprüfe, ob $\{n\}_K$ = n'

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

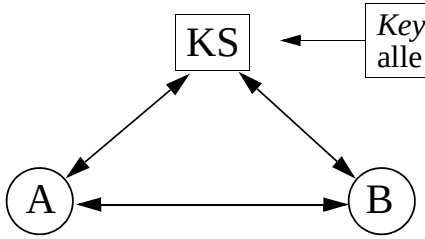
Authentifizierung mit asymmetrischen Schlüsseln



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



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



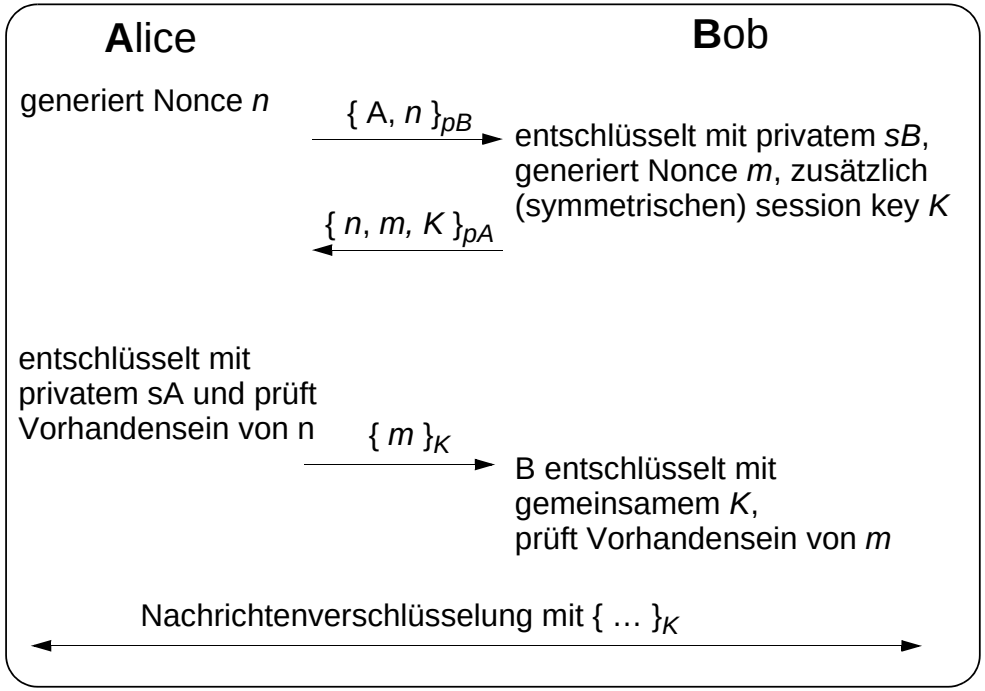
Key Server: Kennt alle public keys

- 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 evtl. repliziert oder verteilt

Gegenseitige Authentifizierung mit Schlüsselvereinbarung

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



Replays

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

---

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

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

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

3) Mitverschlüsseln der *Absendezeit*

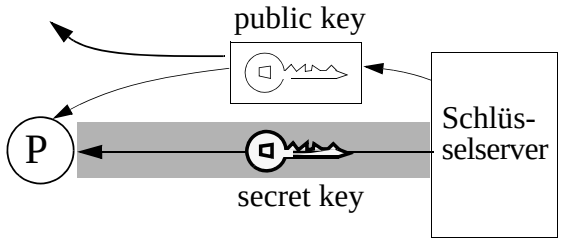
   - Empfänger akzeptiert Nachricht nur, wenn seine Zeit max. Δt abweicht.

   - 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 jeder Nachricht oder nach einem 'reject')
   - Angreifer darf Zeitservice nicht manipuliern können!

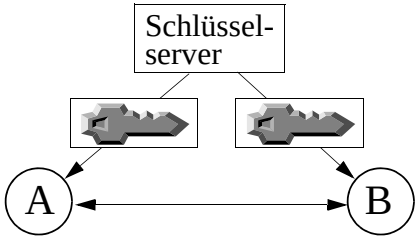
Schlüsselvergabe

- Zur Vergabe eines Paares von public / secret keys:



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

- Zur Generierung von temporären symmetrischen Schlüsseln ("session key")



Session keys werden sicher und authentisch mit einem Public-Key-Verfahren an zwei Kommunikationspartner übertragen

- Schlüsselserver kann session keys nach Übertragung bei sich löschen
- aufwändiges Public-Key-Verfahren nur ein Mal pro "Session", tatsächliche Nachrichtenverschlüsselung dann effizient per symm. Schlüssel