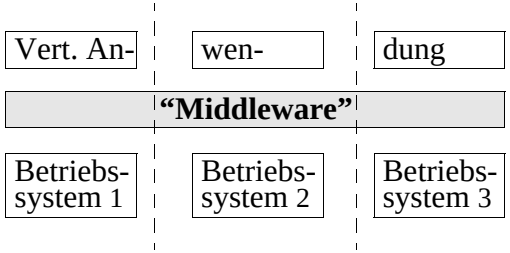
Middleware

Middleware

- Zweck: Durch eine geeignete Softwareinfrastruktur die Realisierung verteilter Anwendungen vereinfachen
  - Herausfaktorisieren gemeinsame Aspekte typischer verteilter Anwendungen



- Aufgabe von Middleware:
  - Verteiltheit (für die Anwendung) möglichst “unsichtbar” machen (z.B. globaler Namensraum, globale Zugreifbarkeit, Ortstransparenz)
  - zumindest aber die Verteiltheit einfach handhabbar machen
- Soll insbesondere Kommunikation und Kooperation zwischen Anwendungsprogrammen unterstützen
  - Verbergen von Heterogenität von Rechnern und Betriebsystemen (z.B. durch einheitliche Datenformate)
  - einheitliche „Umgangsformen“: Schnittstellen, Protokolle
- Bietet gewisse Basismechanismen und -dienste für verteiltes Programmieren an, z.B.
  - Verzeichnis- und Suchdienste (name service, lookup service,...)
  - Weiterleiten von events

Übersicht: “Historische” Entwicklung

1. RPC-Bibliotheken
   - Unterstützung des Client-Server-Paradigmas
   - Schnittstellen-Beschreibungssprache, Datenformatkonversion, Stubgeneratoren
   - Sicherheitskonzepte wie Authentifizierung, Autorisierung, Verschlüsselung

2. Client-Server-Verteilungsplattformen (z.B. DCE)
   - Verzeichnisdienst, globaler Namensraum, globales Dateisystem
   - Programmierhilfen: Synchronisation, Multithreading,…

3. Objektbasierte Verteilungsplattformen (z.B. CORBA)
   - Kooperation zwischen verteilten Objekten
   - objektorientierte Schnittstellenbeschreibungssprache
   - “Object Request Broker”

4. Web-Services
   - Dienstorientierung aufbauend auf WWW als Plattform (SOAP, XML)

5. Infrastruktur für spontane Kooperation (z.B. Jini)
   - unterstützt Dienstorientierung, Mobilität, Dynamik

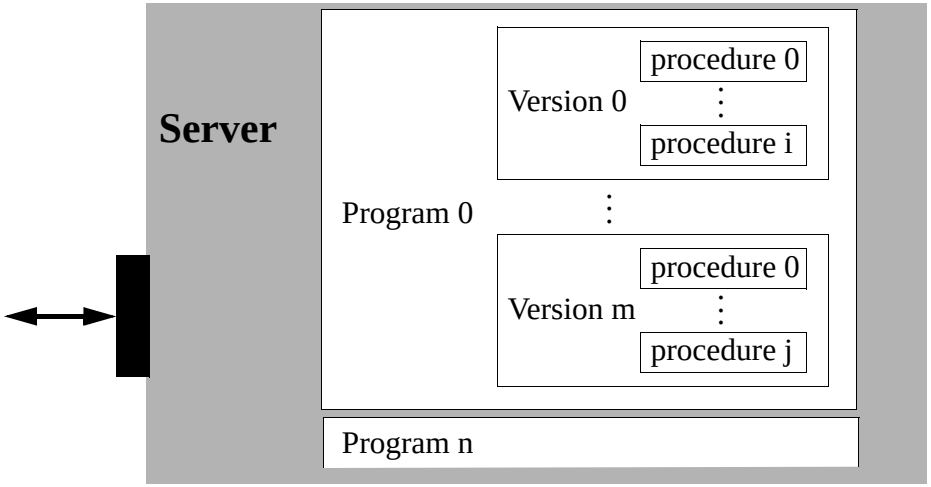
Beachte: Der Begriff “Middleware” ist im Laufe der Zeit zunehmend verwässert worden

- oft nicht nur gebraucht im technischen Sinne als Verteilungsplattform und Kommunikations- und Diensteinfrastruktur
- sondern auch für fast alles, was nicht direkt Anwendung oder Betriebssystem ist, also z.B. auch Datenbanken etc.

RPC unter UNIX, Linux etc.

- Ursprünglich entwickelt von der Firma Sun (“Sun-RPC”)
  - einige Aspekte wurden Standard im Internet

- Eine entfernte Prozedur wird identifiziert durch ein Tripel (prognum, versnum, procnum)



- Jede Prozedur eines Dienstes realisiert eine Teilfunktionalität (z.B. open, read, write,... bei einem Dateiserver)

- Prozedur Nummer 0 ist vereinbarungsgemäss für die “Nullprozedur” reserviert
  - keine Argumente, kein Resultat, sofortige Rückkehr (“Ping-Test”)

- Mit der Nullprozedur kann ein Client feststellen, ob ein Dienst in einer bestimmten Version existiert:
  - falls Aufruf von Version 4 des Dienstes XYZ nicht klappt, dann versuche, Version 3 aufzurufen...

“Sun-RPC”: Service-Registrierung

int *rpc_reg*(prognum, versnum, procnum, procname, inproc, outproc)

Register procedure *procname* with the RPC service package. If a request arrives for program *prognum*, version *versnum*, and procedure *procnum*, *procname* is called with a pointer to its parameters; procname must be a procedure that returns a pointer to its static result; *inproc* is used to XDR-decode the parameters while *outproc* is used to XDR-encode the results.

- Welche Programmnummer bekommt ein Service?

  → Einige Programmnummern für *Standarddienste* sind vom System bereits fest konfiguriert:

Servicename	*Prg-Nr.*	*Kommentar*
portmapper	100000	portmap
rstatd	100001	rup
rusersd	100002	rusers
nfs	100003	nfsprog
ypserv	100004	ypprog
mountd	100005	mount
...	...	

Zuordnung mittels *getrpcbyname*() und *getrpcbynumber*() möglich

  → Ansonsten freie Nummer wählen:

- Mit *pmap_set*(prognum, versnum, protocol, port) oder *rpcb_set* bekommt man den Returncode FALSE, falls prognum bereits (dynamisch) vergeben; ansonsten wird dem Service die Portnummer ‘port’ zugeordnet

TCP oder UDP

“Sun-RPC”: Service-Aufruf

int *rpc_call*(host, prognum, versnum, procnum, inproc, in, outproc, out)

Call the remote procedure associated with *prognum*, *versnum*, and *procnum* on the machine *host*. The parameter *in* is the address of the procedure’s argument, and *out* is the address of where to place the result; *inproc* is an XDR function used to encode the procedure’s parameters, and *outproc* is an XDR function used to decode the procedure’s results.

Warning: You do not have control of timeouts or authentication using this routine.

- Es gibt auch eine Broadcast-Variante:

rpc_broadcast(prognum, versnum, procnum, inproc, in, outproc, out, eachresult)

Like *rpc_call*(), except the call message is broadcast... Each time it receives a response, this routine calls *eachresult*(). If *eachresult*() returns 0, *rpc_broadcas*t() waits for more replies.

“Sun-RPC”: Secure RPC

- Client und Server vereinbaren einen DES-Session-Key K (als *“shared secret”* nach dem Diffie-Hellman-Prinzip)
  - wird zum Verschlüsseln der Nachrichten genutzt
- Mit jeder Request-Nachricht wird ein mit K kodierter Zeitstempel als Verifier mitgesandt
- Die erste Request-Nachricht enthält ausserdem verschlüsselt eine Zeitfenstergrösse W als zeitliches Toleranzintervall sowie (ebenfalls verschlüsselt) W-1
  - “zufälliges” Generieren einer ersten Nachricht ist nahezu unmöglich
  - replay (bei kleinem W) ist ebenfalls erfolglos
  - W ist verschlüsselt, um Angreifern keine Informationüber die Fenstergrösse und auch kein Klartext-Schlüsseltext-Paar zu geben
- Server überprüft jeweils, ob:
  (a) Zeitstempel grösser als letzter Zeitstempel
  (b) Zeitstempel innerhalb des Zeitfensters
- Die Antwort des Servers enthält (verschlüsselt) den letzten erhaltenen Zeitstempel-1 (→ Authentifizierung!)
  - gelegentliche Uhrenresynchronisation nötig (RPC-Aufruf kann hierzu optional die Adresse eines “remote time services” enthalten)

CORBA

- **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
  - erste brauchbare (d.h. interoperable) Version: 1997 (CORBA 2.0)

eine *Architektur*, kein Produkt!

- Propagiert durch die OMG (Object Management Group)
  - herstellerübergreifendes Konsortium
  - Ziel: Definition und Entwicklung einer Architektur für kooperierende objektorientierte Softwarebausteine und Services in verteilten heterogenen Systemen (→ *“Middleware”*)

Beachte: *Objektorientierung* selbst ist eigentlich ein “altes” Konzept

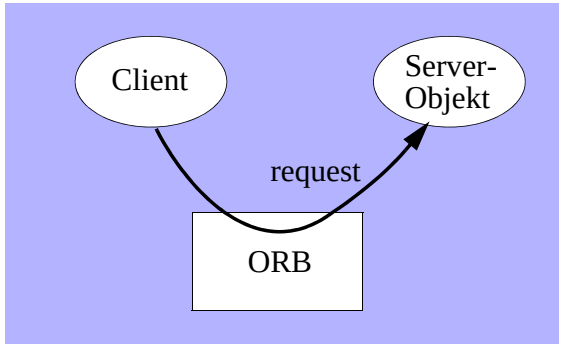
- Mitte der 1960er-Jahre (Programmiersprache “Simula”)
- damals bereits fast alle Aspekte der Objektorientierung (Klassenhierarchien, virtuelle Klassen, Polymorphismus,...)

Man lese zu CORBA auch: Michi Henning: *The rise and fall of CORBA.* Commun. ACM, Vol. 51, No. 8 (August 2008), pp. 52-57

Mehr zur CORBA allgemein: Oliver Haase: *Kommunikation in verteilten Anwendungen (2. Auflage).* R. Oldenbourg Verlag, 2008, Kapitel 7

CORBA - Übersicht

- *Objektmodell*
- *IDL* (**I**nterface **D**escription **L**anguage) mit entsprechenden Generatoren und Compilern
- *ORB* (**O**bject **R**equest **B**roker) als Vermittlungsinfrastruktur

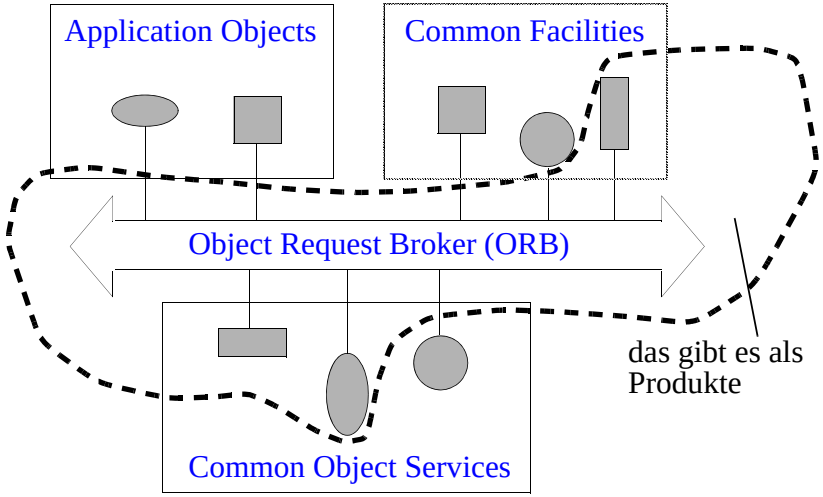


- *Konventionen* bezüglich Schnittstellen, Protokollen etc.
- *Systemfunktionen* in Form von Object Services

- CORBA-Implementierungen sind also *Infrastrukturen* und unterstützen die *Ausführung* vert. objektorientierter Anwendungen in heterogenen Systemen
- *Entwurfs- und Spezifikationsaspekte* solcher Systeme werden dagegen mit anderen Konzepten unterstützt, z.B. *UML* ("Unified Modeling Language"), mit der u.a. Diagrammnotationen standardisiert werden

Object Management Architecture

- OMA ist eine *Referenzarchitektur*, welche die wesentlichen Bestandteile einer Plattform für verteilte objektorientierte Applikationen definiert:



- *Application Objects*: Objekte der eigentlichen Anwendung
  - gehören damit nicht zur CORBA-Infrastruktur
- *ORB*: Vermittlung zwischen verschiedenen Objekten; Weiterleitung von Methodenaufrufen etc.
  → Ortstransparenz, Kommunikation,...
- *Object Services*: Schnittstelle zu standardisierten wichtigen Diensten
- *Common Facilities*: allgemein nützliche Funktionalität
  - nicht Teil aller CORBA-Implementierungen, oft sind nur wenige Funktionen davon realisiert

Object Services (1)

- “Common Object Services” als Basisdienste für eine systemweite Infrastruktur
  - mit objektorientierter Schnittstelle
  - nicht alle Dienste wurden aber vollständig spezifiziert bzw. realisiert

---

1) Ereignismeldung
   - Weiterleitung asynchroner Ereignisse an Ereigniskonsumenten
   - Einrichten von “event channels” mit Operationen wie push, pull,...

2) Persistenz
   - Dauerhaftes Speichern von Objekten auf externen Medien

3) Naming
   - Erzeugung von Namensräumen
   - Abbildung von Namen auf Objektreferenzen
   - Lokalisierung von Objekten

4) Trading
   - Matching von Services zu einer Service-Beschreibung eines Clients

5) Time
   - Uhrensynchronisation etc.

6) Security

Object Services (2)

7) Concurrency control
   - Semaphore, Locks,...

8) Transaktionen
   - 2-Phasen-Commit etc.

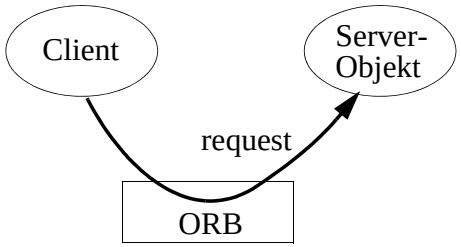
9) Replikation
   - Sicherstellung der Konsistenz replizierter Objekte in einer verteilten Umgebung

10) Externalization
   - Export von Objekten in sequentielle Dateien

- und noch einige andere (nicht weiter relevante) Services

Kommunikation zwischen Objekten

- Im Wesentlichen Client/Server-Prinzip



- Methodenaufruf durch requests unterschiedl. Semantik
  - *synchron* (mit Rückgabewerten; analog zu RPC)
  - *verzögert synchron* (Aufrufer wartet nicht auf das Ergebnis, sondern holt es sich später ab)
  - *one way* (asynchron: Aufrufer wartet nicht; keine Ergebnisrückgabe)

- Für den transparenten Transport eines Methodenaufrufs vom Client zum Server ist der ORB zuständig

Interface Description Language (IDL)

- Sprache zur Definition von Schnittstellen (Parameter, Attribute, Superklasse bzgl. Vererbung, Exceptions,...)

- Sprachneutral, aber lexikalisch an C++ angelehnt

- Bsp: **`oneway void move (in long x, in long y)`**

Eingabe-, Rückgabe- oder Ein- und Rückgabeparameter

- Grundsätzlicher Aufbau:

  [oneway] <op_type_spec> <identifier> ($param_1$,....,$param_L$)
  [raises($except_1$,...,$except_N$)]

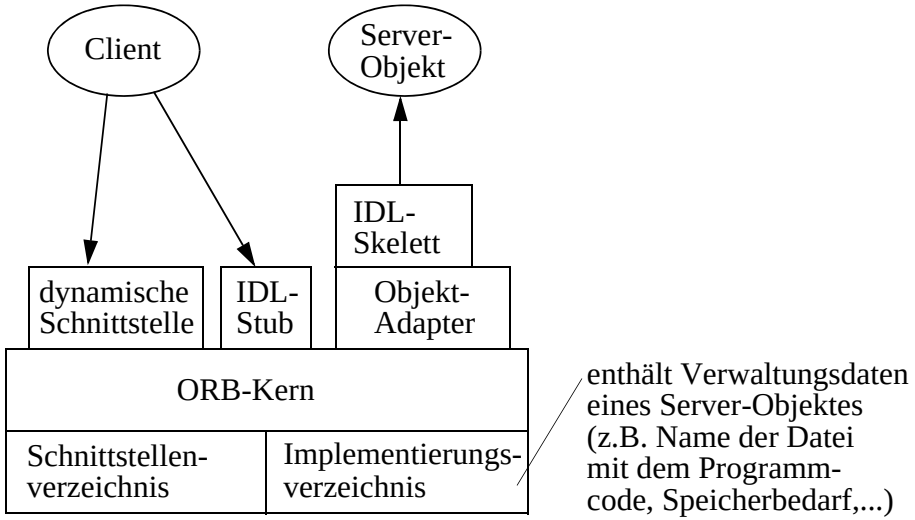
erlaubt Angabe möglicher Exceptions, die bei Durchführung der Methode auftreten können

- Aus einer Schnittstellenspezifikation in IDL erzeugt ein Compiler einen IDL-Stub für Clients und ein IDL-Skelett für Server

---

- Mehr zur CORBA-IDL siehe z.B.: Oliver Haase: *Kommunikation in verteilten Anwendungen (2. Auflage)*. R. Oldenbourg Verlag, 2008, Kapitel 7.2

ORB



- ORB bietet Clients zwei Arten von Schnittstellen für den Methodenaufruf an
  - *statische Schnittstelle* (Erzeugung von Stubs aus der IDL-Beschreibung mit Compiler analog zu RPCs)
  - *dynamische Schnittstelle* (Client kann zur Laufzeit das Schnittstellenverzeichnis abfragen und einen geeigneten Methodenaufruf zusammenstellen)

- Objektadapter: Steuert anwendungsunabhängige Funktionen des Server-Objekts
  - u.A. Aktivierung des Server-Objektes bei Eintreffen eines requests, Authentifizierung von requests, Zuordung von Objektreferenzen zu Objektinstanzen etc.
  - zuständig ausserdem für Registrierung von Services
  - es gibt einen standardisierten Basic Object Adapter (BOA), der für viele Anwendungen ausreichende Grundfunktionalität bereitstellt

Server-Objekte

- Server-Objekte können ein aus der IDL-Spezifikation generiertes Objekt-Skelett nutzen

- Objekt muss sich beim lokalen Objekt-Adapter anmelden und dabei eine "server policy" angeben:
  - *Shared Server:* kann zusammen mit mehreren anderen aktiven Server-Objekten von einem einzigen Prozess verwaltet werden
  - *Unshared Server*
  - *Server per Method:* Start eines eigenen Prozesses bei Methodenaufruf
  - *Persistent Server:* ein Shared Server, der von CORBA initial bereits gestartet wurde

- Objekt muss sich ferner beim Implementierungs-verzeichnis anmelden
  - damit es bei einem Methodenaufruf durch Clients gefunden wird

CORBA - weitere Entwicklungen

Ab ca. 2000 entstand der Wunsch nach einer wesentlichen Erweiterung der CORBA-Funktionalität. Gründe:

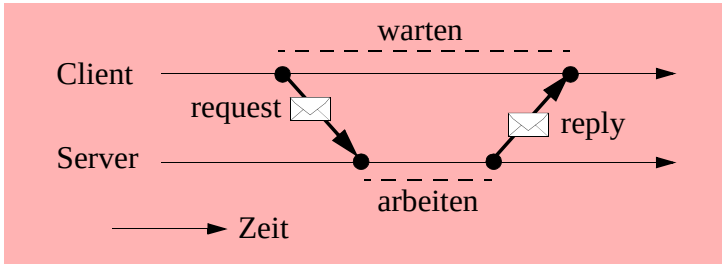
- neue Anforderungen durch E-Commerce-Anwendungen
- Ausbreitung des WWW (und später: Web-Services, SOAP,...)
- Aufkommen von Java (und später: EJB, Jini,...)
- Aufkommen mobiler Geräte

Dem sollte durch Weiterentwicklung ("CORBA 3.0") der Spezifikation Rechnung getragen werden:

- Messaging Service und asynchroner Methodenaufruf
- Objects by Value
- Persistente Objekte
  - "Abspeichern" von Objekten
- Komponenten-Modell
- Java-Unterstützung
  - Generieren von IDL aus Java bzw. Java-RMI ("reverse mapping")
- Firewall-Unterstützung
  - klassische Firewalltechnik (z.B. Services identifiziert mit Portnummern) versagt teilweise; Callbacks erscheinen als Aufruf von aussen...
- Minimum CORBA
  - Unterstützung von embedded systems (i.w. Weglassen von Dynamik)
- Realzeit-Unterstützung
- Fault Tolerant CORBA
  - durch redundante Einheiten

Messaging Service

- Asynchrones Kommunikationsparadigma
- Motivation:
  - mobile Geräte sind oft nicht online
  - bei sehr grossen verteilten Systemen sind fast immer einige Geräte bzw. Services nicht erreichbar (Netzprobleme etc.)
- CORBA basierte bisher auf einer engen ("synchronen") Kopplung von Client und Server



- Asynchron:
  - Entkopplung von Sender / Empfänger
  - Sender blockiert nicht solange, bis Nachricht angekommen ist
  - Nachricht kann von Hilfsinstanzen zwischengespeichert werden, bis Kommunikationspartner erreichbar ist ("store & forward")
  - Antwort kann evtl. von anderem Client als ursprünglichem Sender entgegengenommen werden

Asynchronous Method Invocation

- Bisherige Möglichkeiten eines Methodenaufrufs in CORBA:
  - *synchron* (insbes. bei Rückgabewerten; analog zu RPC)
  - *verzögert synchron* (Aufrufer wartet nicht auf das Ergebnis, sondern holt es sich später ab)
  - *one way* (Aufrufer wartet nicht) mit "best effort"-Semantik ("fire and forget")

gedacht war an UDP-Implementierung; Semantik (z.B. Fehlermeldung bei Misslingen?) implementierungsabhängig

---

- Neu: Asynchronous Method Invocation (AMI)
  - bisher eher umständlich und fehleranfällig durchThreads zu simulieren
- Zwei Aufruftechniken bei AMI:
  - (1) Callback
    - Client gibt dem Aufruf eine Objektreferenz für die Antwort mit
    - Callback-Objekt kann sich im Prinzip irgendwo befinden
    - Kommunikations-Exceptions werden im Callback-Objekt ausgelöst
  - (2) Polling
    - Client erhält sofort ein Objekt zurück, das er für Polling oder zum Warten auf Antwort nutzen kann

Time-independent Invocation

- Aufruf von Objekten, die nicht aktiv sind oder zeitweise nicht erreichbar sind
- Aufruf-Nachrichten werden von zwischengeschalteten "Router Agents" verwaltet
  - Store and Forward-Prinzip
  - Router Agent beim Client ermöglicht disconnected operations
  - Router Agent beim Server kann dessen Eingangsqueue verwalten
- Sogen. Quality of Service steuerbar (als "Policy")
  - z.B. max. Round Trip-Zeit: dadurch brauchen Router Agents die Nachrichten nicht beliebig lange aufbewahren
  - oder z.B. Aufrufreihenfolge: Soll Router seine gespeicherten Aufträge zeitlich geordnet oder nach Prioritäten oder... ausliefern?

CORBA-Probleme

- Die Weiterentwicklung von CORBA geriet ins Stocken
  - zu weitreichende Anforderungen → komplex / ineffizient
  - kommerzielle Implementierungen zögerlich
  - fehlende Unterstützung durch Microsoft (eigene Architektur)
  - OMG versuchte, es jedem Recht zu machen (widersprüchliche Interessen, barocke Konstrukte durch Kompromisse,...)
  - aufkommende Konkurrenzsysteme, die z.T. besser an die neuen Anforderungen angepasst sind

---

Konkurrenzsysteme bzw. alternative Ansätze u.a.:

- Microsoft: DCOM bzw. .Net
- Java-Technologie
  - z.B. Enterprise Java Beans (EJB), RMI
- Web-Services (und verwandte Systeme)
  - XML, WSDL, SOAP,...
  - Integrationsplattformen (z.B. WebSphere von IBM)

DCOM

Component Object Model

- DCOM: verteilte Version von COM (Microsoft)
- Teilweise analoge Zielsetzung zu CORBA
  - Integration, Interoperabilität, Komponenten-Modell
  - transparenter Zugriff auf entfernte Objekte
  - statische und dynamische Aufrufe
  - IDL heisst hier MIDL ('M' für Microsoft)
  - Aufgaben von ORB und Object Adapter werden von einem "Service Control Manager" (SCM) wahrgenommen
- DCOM aber proprietär auf MS-Technologie ausgerichtet
  - Kapselung binärer Bibliotheken
- Keine Vererbung, stattdessen *Delegation* und *Aggregation*
  - Einbettung von Objekten in andere mit Weiterreichung von Schnittstellen nach aussen
  - *Aggregation:* automatisches Durchreichen der kompletten inneren Schnittstelle nach aussen
  - *Delegation:* Durchreichen eines Teils der inneren Schnittstelle nach aussen; äussere Schnittstelle ruft innere Methoden explizit auf
- Abgelöst durch .NET

.NET-Framework

- Microsoft-Softwareplattform; integriert in Windows
  - Laufzeitumgebung, Klassenbibliotheken (API), Services
  - für gemischtsprachige Programmierung in sogenannten .NET-Sprachen (u.a. C#, C++, Visual Basic)

- Quellcode in .NET-Sprache wird kompiliert in Common Intermediate Language (CIL)
  - entspricht etwa Java Bytecode

- Virtuelle .NET-Maschine (Common Language Runtime, CLR) führt CIL-Code aus
  - entspricht Java Virtual Machine
  - CLR enthält Just-in-Time (JIT) Compiler

- .NET-Remoting: entfernter Methodenaufruf
  - Clients verwenden lokale Proxies, die dieselbe Schnittstelle wie das entfernte Serverobjekt anbieten und Methodenaufrufe weiterleiten

- Verschiedene *Server-Aktivierungsmodi* möglich:
  - *Singleton:* ein einziges Serverobjekt für alle Methodenaufrufe
  - *SingleCall:* ein dediziertes Serverobjekt für jeden Methodenaufruf
  - *klientenaktiviertes Objekt:* erzeugt neues Serverobjekt, das für alle Aufrufe desselben Klienten genutzt wird (dadurch kann klientenspezifische Zustandsinformation über Aufrufe hinweg gehalten werden)

- Alternativ zu .NET-Remoting bietet .NET auch *Socket-Kommunikation* und XML-basierte *Web-Services* an

Objektserialisierung

- Konvertierung von Objekten in Byteströme (und Rückkonvertierung in identische Kopie des Ausgangsobjekts)
  - Abspeichern von Objekten auf externen Medien
  - Kommunikation von Objekten

- Java RMI (**R**emote **M**ethode **I**nvocation)
  - Schnittstelle “Serializable” implementieren
  - Serialisierung durch Schreiben auf “ObjectOutputStream”
  - Instanzvariablen, die nicht serialisiert werden sollen, werden mit “transient” gekennzeichnet
  - Deserialisierung durch Lesen von “ObjectInputStream”

- .NET: zwei mögliche Serialisierungsformate
  - *Binärformat:* platzsparend, effizient, nicht menschlich lesbar, gut für homogene .NET-Anwendungen
  - *SOAP-Format:* XML-Kodierung, interoperable mit anderen SOAP-Komponenten (auch Nicht-.NET)
  - zu serialisierende Klasse wird mit Attribut “Serializable” versehen
  - nicht zu serialisierende Komponente wird mit Attribut “NonSerialized” markiert $\rightarrow$ entspricht bei Java “transient”

Web Services

- Definition des W3C (World Wide Web Consortium):

  A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

- Also: Ein entfernter Dienst, der über *SOAP-Nachrichten* aufgerufen werden kann; in der Web Service Description Language *(WSDL) beschrieben* ist; und im Universal Description Discovery and Integration *(UDDI)-Registry* verzeichnet ist.

- *WSDL:* XML-basierte Sprache zur Spezikation der Schnittstellen von Web Services
  - Rolle vergleichbar mit CORBA IDL

- *SOAP* (Simple Object Access Protocol): Austausch von XML-Nachrichten über HTTP (oder HTTPS) zum Zweck des Remote Procedure Calls

- *UDDI:* XML-basierter Verzeichnisdienst für Web Services
  - UDDI ist selbst ein Web Service → Anfragen über SOAP-Nachrichten
  - Ergebnisse sind WSDL-Dokumente

Web Services (2)

- Im Wesentlichen also RPC über Internet / WWW
  - das Web aufgefasst und genutzt als Software-Layer

- Eigenschaften
  - unabhängig von existierenden Plattformen (Sprachen, Middleware)
  - sehr lockere Koppelung von Client und Server
  - ubiquitär nutzbar, (im Prinzip) weltweit zugreifbar

- Web-Browser als kanonischer Client nutzbar
  - fungiert als Interface für Nutzer bei Web-Service-Applikationen

- Problembereiche
  - Overhead für einen Aufruf ist relativ gross
  - http war als reines Dokumentenaustauschprotokoll für menschl. Nutzer konzipiert worden - nicht zur Kommunikation zwischen Computern
  - die Beschreibung von global anwendbaren Diensten für E-Commerce etc. stellt andere Anforderungen als die (rein syntaktische) Interface-Beschreibung klassischer Prozeduren in Programmiersprachen
  - UDDI-Service global (“universell”) zu etablieren, ist eine technische und kommerzielle Herausforderung (teilweise Suchmaschinen-funktionalität!)

- Erweiterung der Web-Service-Idee: Service-orientierte Architekturen (SOA)

Mehr zu Web-Services in anderen Vorlesungen (→ Prof. G. Alonso)

Jini

(Teil der Vorlesung
„Verteilte Systeme“)

Jini

- Infrastructure (“middleware”) for dynamic, cooperative, spontaneously networked systems
  - facilitates implementation of distributed applications

F. Ma. 2

Jini

- Infrastructure ("middleware") for dynamic, cooperative, spontaneously networked systems
  - facilitates implementation of distributed applications

- framework of APIs with useful functions / services
- helper services (discovery, lookup,…)
- suite of standard protocols and conventions

F. Ma. 3

Jini

- Infrastructure ("middleware") for dynamic, cooperative, spontaneously networked systems
  - facilitates implementation of distributed applications

- services, devices, ... find each other automatically ("plug and play")
- dynamically added / removed components
- changing communication relationships
- mobility

F. Ma. 4

Jini

- Infrastructure ("middleware") for dynamic, cooperative, spontaneously networked systems
  - facilitates implementation of distributed applications
- Based on Java
  - uses RMI (Remote Method Invocation)
  - code shipping
  - requires JVM / bytecode everywhere
- Service-oriented
  - (almost) everything is considered a service
  - Jini system is a federation of services
  - service access through mobile proxy objects

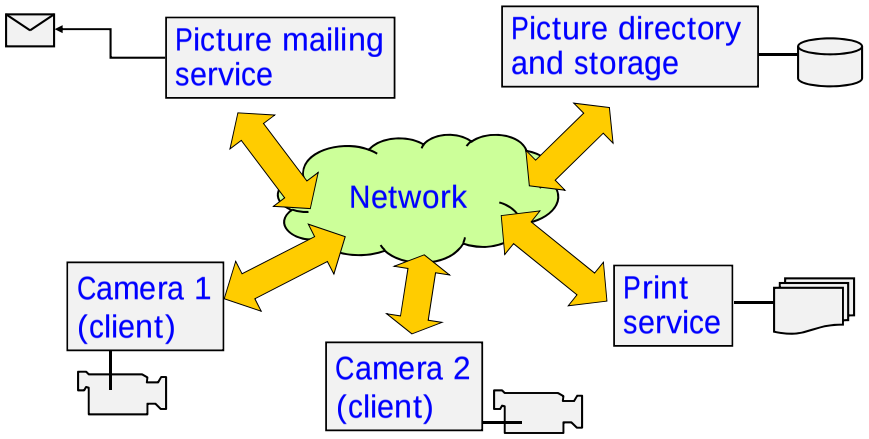
F. Ma. 5

Service Paradigm

- (Almost) everything relevant is a service
- Jini's run-time infrastructure offers mechanisms for adding, removing, finding, and using services
- Services are defined by interfaces and provide their functionality via their interfaces
  - services are characterized by their type and their attributes (e.g. "600 dpi", "version 21.1")
- Services (and service users) may "spontaneously" form a so-called "federation"

F. Ma. 6

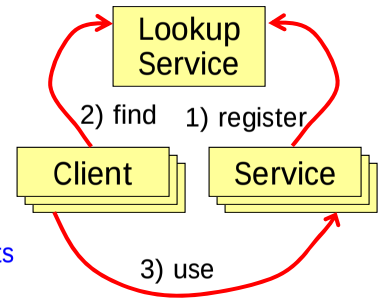
Example of a Jini Federation



F. Ma. 7

Jini: Global Architecture

- Lookup Service (LUS)
  - main registry entity and brokerage service for services and clients
  - maintains information about available services
- Services
  - specified by Java interfaces
  - register together with proxy objects and attributes at the LUS
- Clients
  - know the Java interfaces of the services, but not their implementation
  - find services via the LUS
  - use services via proxy objects



F. Ma. 8

Network Centric

- Jini is based on the network paradigm
  - network = hardware and software infrastructure
- View is “network to which devices are connected to”, not “devices that get networked”
  - network always exists, devices and services are transient
- Jini supports dynamic networks and adaptive systems
  - adding and removing components or communication relations should only minimally affect other components

F. Ma. 9

Spontaneous Networking

- Objects in an open, distributed, dynamic world find each other and form a transitory community
  - cooperation, service usage, ...
- Typical scenario: client wakes up (device is switched on, plugged in, ...) and asks for services in its vicinity
- Finding each other and establishing a connection should be fast, easy, and automatic

F. Ma. 10

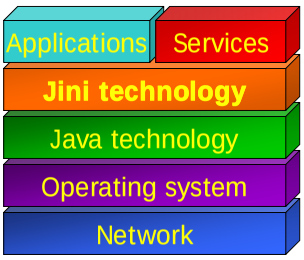
Some Fallacies of Common Distributed Computing Systems

- The “classical” idealistic view...
  - the network is reliable
  - latency is zero
  - bandwidth is infinite
  - the network is secure
  - the topology is stable
  - there is a single administrator
- ...isn’t true in fact
  - Jini addresses some of these issues (or at least it does not hide or ignore them)

F. Ma. 13

Bird’s-Eye View on Jini as a Middleware Infrastructure

- Jini consists of a number of APIs
- Is an extension to the Java platform dealing with distributed computing
- Is an abstraction layer between the application and the underlying infra-structure (network, OS)



F. Ma. 14

Jini's Use of Java

- Jini requires JVM (as bytecode interpreter)
  - homogeneity in a heterogeneous world
  - but is this a realistic assumption?

run protocols for discovery and join; have a JVM

- Devices that are not "Jini-enabled" or that do not have a JVM can be managed by a software proxy (somewhere in the net)

F. Ma. 15

Main Components of the Jini Infrastructure

- Lookup service (LUS)
  - as repository / naming service / trader
- Protocols
  - discovery & join, lookup of services
  - based on TCP/UDP/IP
- Proxy objects
  - transferred from service to clients (via LUS)
  - represent the service locally at the client

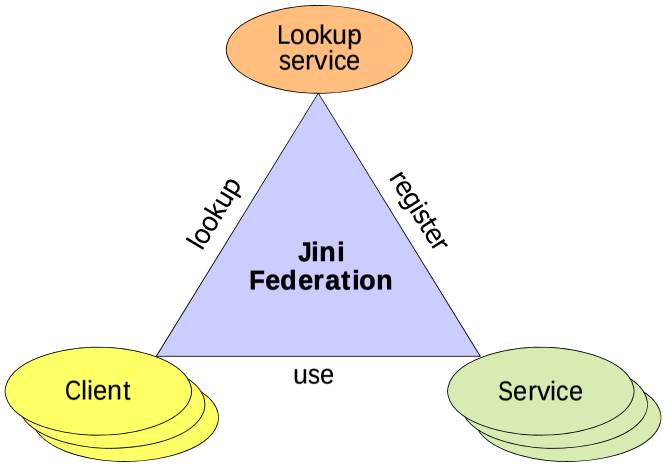
F. Ma. 16

Lookup Service (LUS)

- Central component of every Jini federation
- Repository of services
- Similar to naming services (e.g., RMI registry) of other middleware architectures
- Functions as a “help-desk” for services and clients
  - registration of services (services advertise themselves)
  - distribution of services (clients lookup and find services)
- Has mechanisms to bring together services and clients

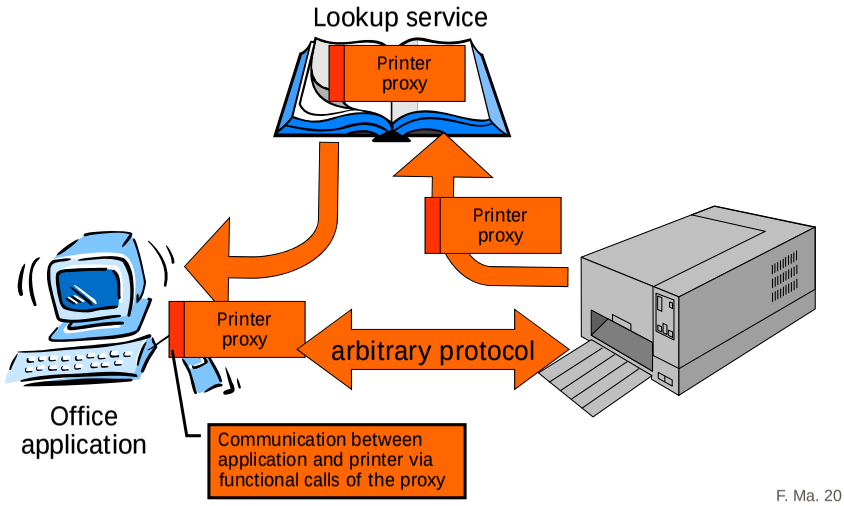
F. Ma. 18

Lookup Service



F. Ma. 19

Example



F. Ma. 20

Lookup Service

- Uses Java RMI for communication
  - objects („proxies“) can migrate over the network
- Not only name/address of a service is stored (as in traditional naming services), but also:
  - set of attributes
    - e.g.: printer(color: true, dpi: 600, ...)
  - proxies, which may be complex classes
    - e.g. user interfaces
- Further possibilities:
  - responsibility can be distributed to a number of (logically separated) lookup services
  - increase robustness by running redundant lookup services

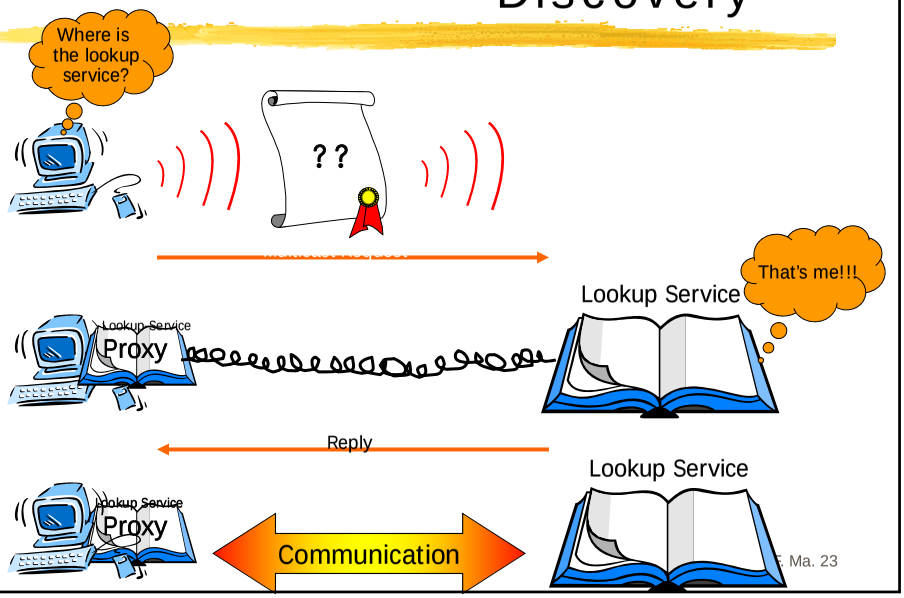
F. Ma. 21

Discovery: Finding a LUS

- Goal: Find a lookup service (without knowing anything about the network) to
  - advertise (register) a service, or
  - find (look up) an existing service
- Discovery protocol:
  - multicast to well-known address/port
  - lookup service replies with a serialized object (its proxy)
    - communication with LUS then via this proxy

F. Ma. 22

Discovery



F. Ma. 23

Multicast Discovery Protocol

- Search for lookup services
  - no information about the host network needed
- Discovery request uses multicast UDP packets
  - multicast address for discovery (224.0.1.85)
  - default port number of lookup services (4160)
  - recommended time-to-live is 15
  - usually does not cross subnet boundaries
- Discovery reply is establishment of a TCP connection
  - port for reply is included in multicast request packet

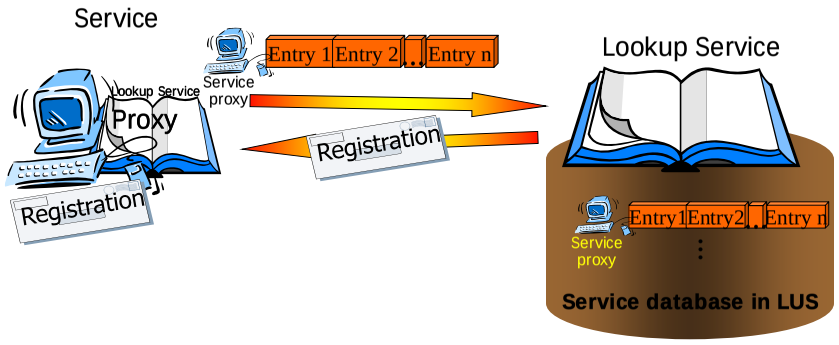
F. Ma. 24

Join: Registering a Service

- Assumption: Service provider already has a proxy of the lookup service (→ discovery)
- It uses this proxy to register its service
- Gives to the lookup service
  - its service proxy
  - attributes that further describe the service
- Service provider can now be found and used in this Jini federation

F. Ma. 25

Join



F. Ma. 26

Join: More Features

- To join, a service supplies:
  - its proxy
  - its ServiceID (if previously assigned; “universally unique identifier”)
  - set of attributes
- Service waits a random amount of time after start-up
  - prevents packet storms after restarting a network segment
- Registration with a lookup service is bound to a lease
  - service has to renew its lease periodically

F. Ma. 27

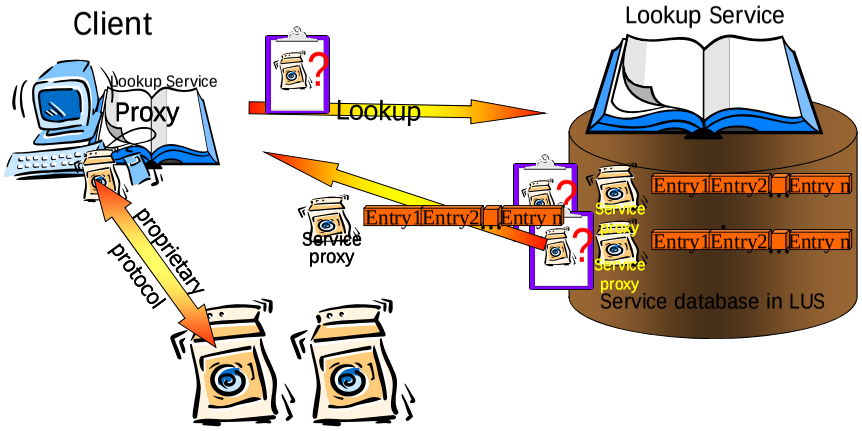
Lookup: Searching Services

- Client creates query for lookup service
  - matching by registration number of service and/or service type and/or attributes possible
  - attributes: only exact matching possible (no “larger-than”, ...)
  - wildcards are possible („null“)
- Via its proxy at the client, the lookup service returns zero, one or more matches (i.e., server proxies)
- Selection among several matches is done by client

- Client uses service by calling functions of the service proxy
- Any “private” protocol between service proxy and service provider is possible

F. Ma. 28

Lookup



F. Ma. 29

Proxies

- Proxy object is stored in the LUS upon registration
  - serialized object
  - implements the service interfaces
- Upon request, service proxy is sent to the client
  - client communicates with service implementation via its proxy: client invokes methods of the proxy object
  - proxy implementation hidden from client

F. Ma. 31

Smart Proxies

- Parts of (or the whole) service functionality may be executed by the proxy at the client
- When dealing with large volumes of data, it usually makes sense to preprocess parts of the data
  - e.g.: compressing video data before transfer
- Partition of service functionality depends on service implementer's choice
  - client needs appropriate resources

F. Ma. 32

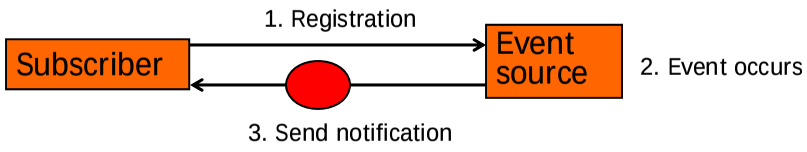
Leases

- Leases are contracts between two parties
- Leases introduce a notion of time
  - resource usage is restricted to a certain time frame
- Repeatedly express interest in some resource:
  - I'm still interested in X
    - renew lease periodically
    - lease renewal can be denied
  - I don't need X anymore
    - cancel lease or let it expire

F. Ma. 33

Distributed Events

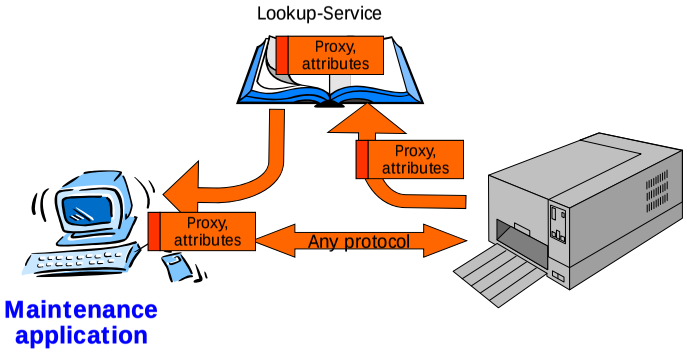
- Objects on one JVM can register interest in certain events of another object on a different JVM
- "Publisher/subscriber" model



F. Ma. 35

Distributed Events – Example

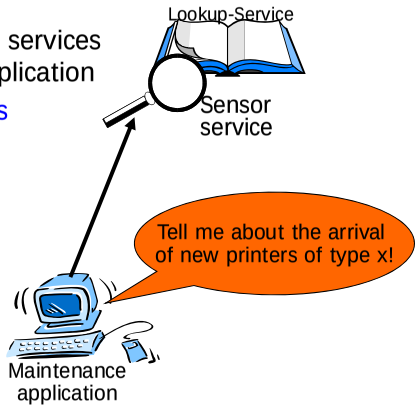
- Example: printer is plugged in
  - printer registers itself with local lookup service
- Maintenance application wants to update software



F. Ma. 36

Distributed Events – Example

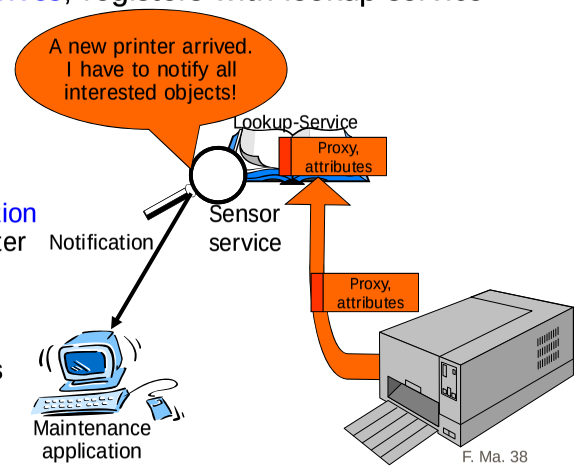
- Maintenance application is run on demand, search for printers is "outsourced"
  - "sensor service" looks for certain services on behalf of the maintenance application
  - maintenance application registers for events showing the arrival of certain types of printers
  - sensor observes the lookup service
  - notifies application as soon as matching printer arrives via distributed events



F. Ma. 37

Distributed Events – Example

- Example: printer arrives, registers with lookup service
  - printer performs discovery and join
  - sensor finds new printer in lookup service
  - checks if there is an event registration for this type of printer
  - notifies all interested objects
  - maintenance application retrieves printer proxy and updates software



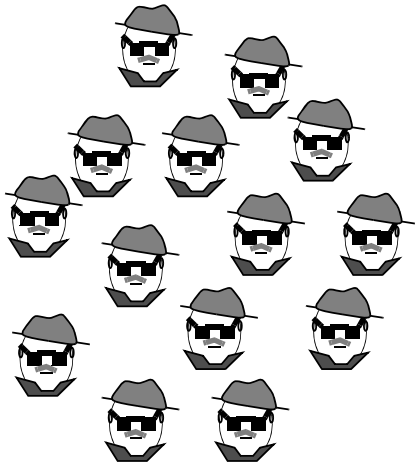
F. Ma. 38

Jini Issues and Problem Areas

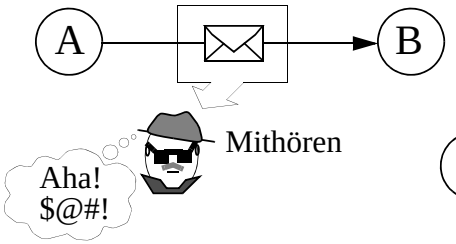
- Security
  - important especially in dynamic environments
  - services use other services on behalf of the user
- Scalability
  - how well does Jini scale to a global level?
- Java centric
- Similar, non-Java-based systems
  - UPnP, Bluetooth SDP, SLP, HAVi, Salutation, e-speak, HP Chai,...
  - open, Internet-scale infrastructures (e.g., Web services)

F. Ma. 39

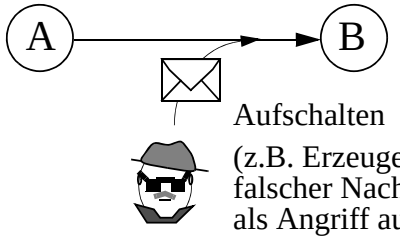
Sicherheit



Sicherheit in verteilten Systemen

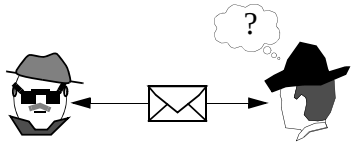


Mithören

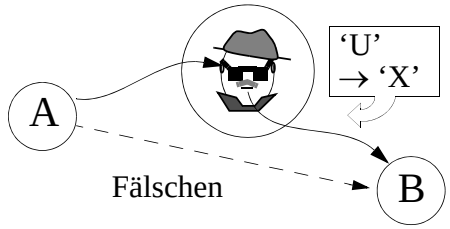


Aufschalten

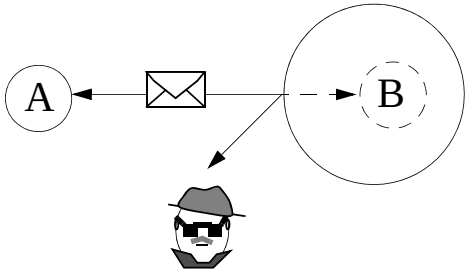
(z.B. Erzeugen falscher Nachrichten als Angriff auf die Glaubwürdigkeit)



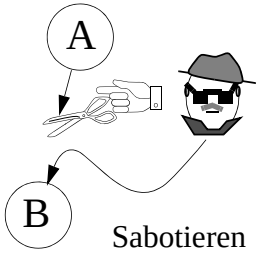
Wechselseitiges Misstrauen



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!

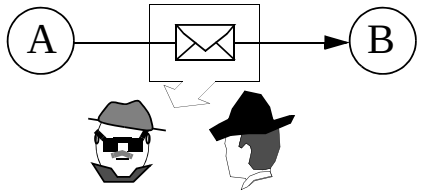
Typische Techniken und "Sicherheitsdienste":

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

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

Angriffsformen

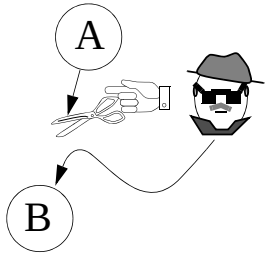
- *Passive Angriffe*: Beobachten der Kommunikation
  - Inhalt von Nachrichten in Erfahrung bringen
  - Kommunikationsverhalten analysieren (“wer mit wem wie oft?”)



→ 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!)

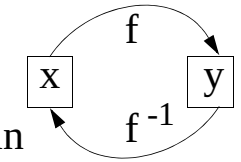
- Peer-Authentifizierung mit *Passwort*
  - typischerweise zur Authentifizierung eines Benutzers ("Client") zum Schutz des Dienstes vor unbefugter Benutzung (Autorisierung)
  - Kenntnis des Passworts gilt als Identitätsbeweis (ist das gerechtfertigt?)

- 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!)

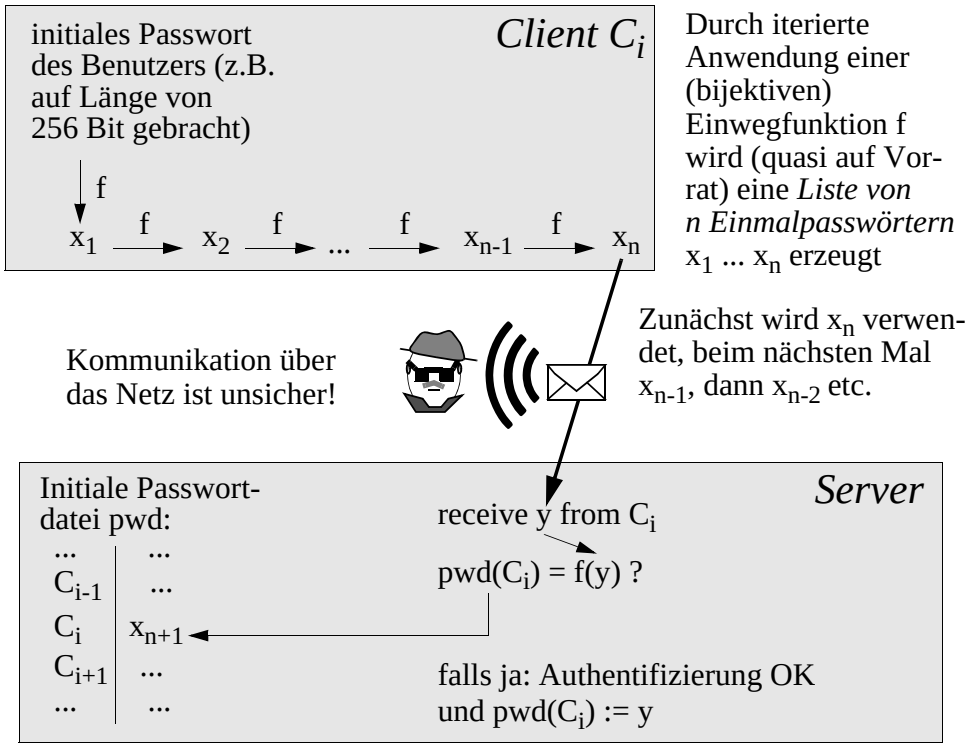
- Einwegfunktionen erscheinen zunächst ziemlich sinnlos: Ein zu $y = f(x)$ verschlüsselter Text x kann nie wieder entschlüsselt werden!

  ⇒ 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!)

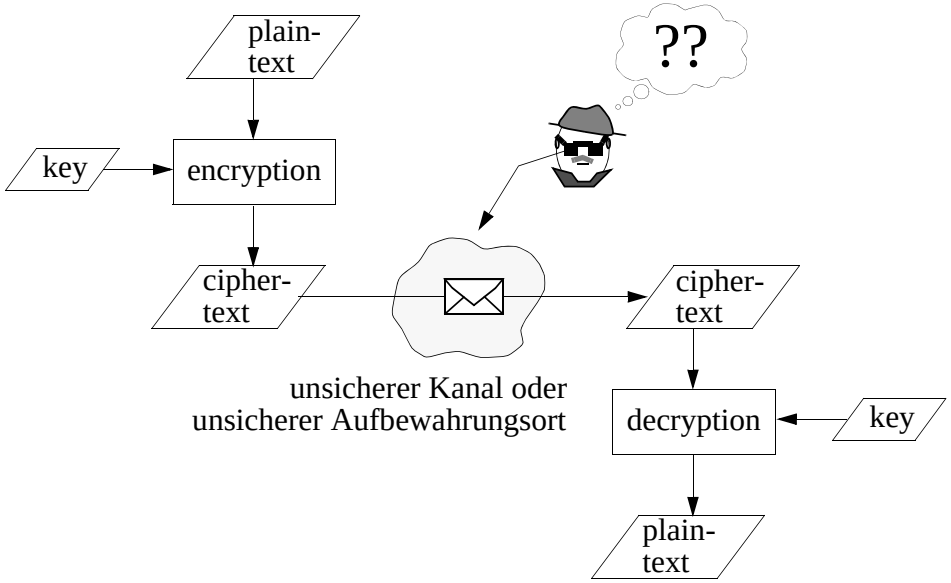
Einmalpasswörter mit Einwegfunktionen

- 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

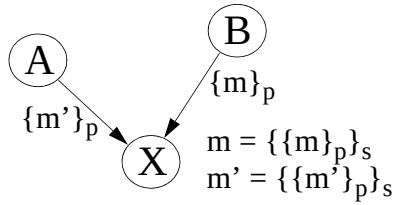
- Für jeden Prozess X existiert ein Paar (p,s)

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

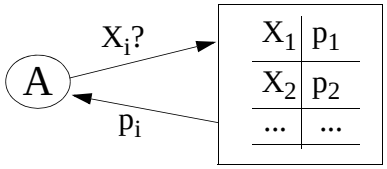
  s = *secret key* (oder "private" key) ← zum *Entschlüsseln* von mit p verschlüsselten Nachrichten

- 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

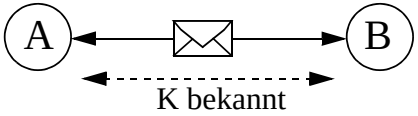
- Vorteil gegenüber symmetrischen Verfahren: vereinfachter Schlüsselaustausch
  - jeder darf den übermittelten public key p mithören
  - secret key s braucht grundsätzlich nie anderen mitgeteilt zu werden
  - bei n Teilnehmern genügen 2n Schlüssel (statt $O(n^2)$ bei sym. Schlüsseln)

- 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")

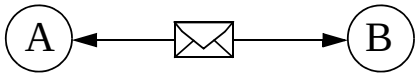
A: n' := $\{n\}_K$

A→B: n'

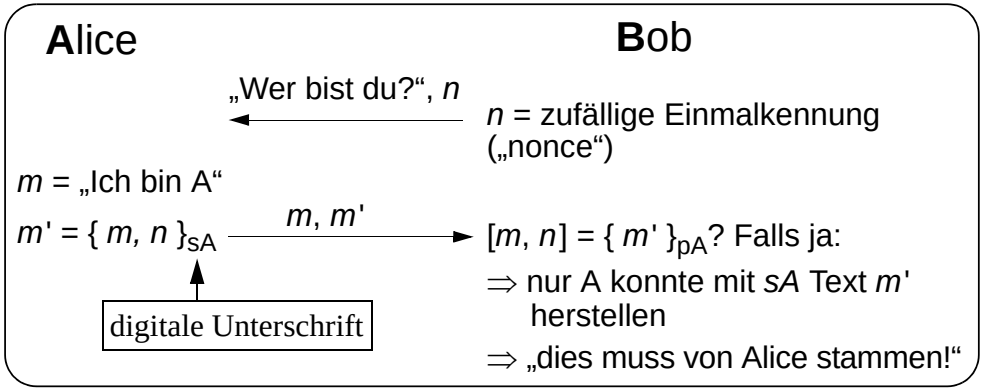
B: überprüfe, ob $\{n\}_K$ = n'

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

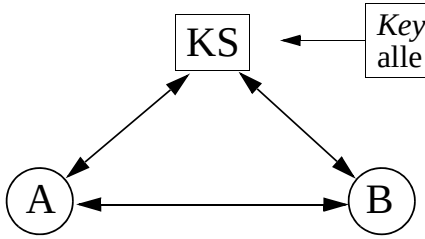
Authentifizierung mit asymmetrischen Schlüsseln



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:



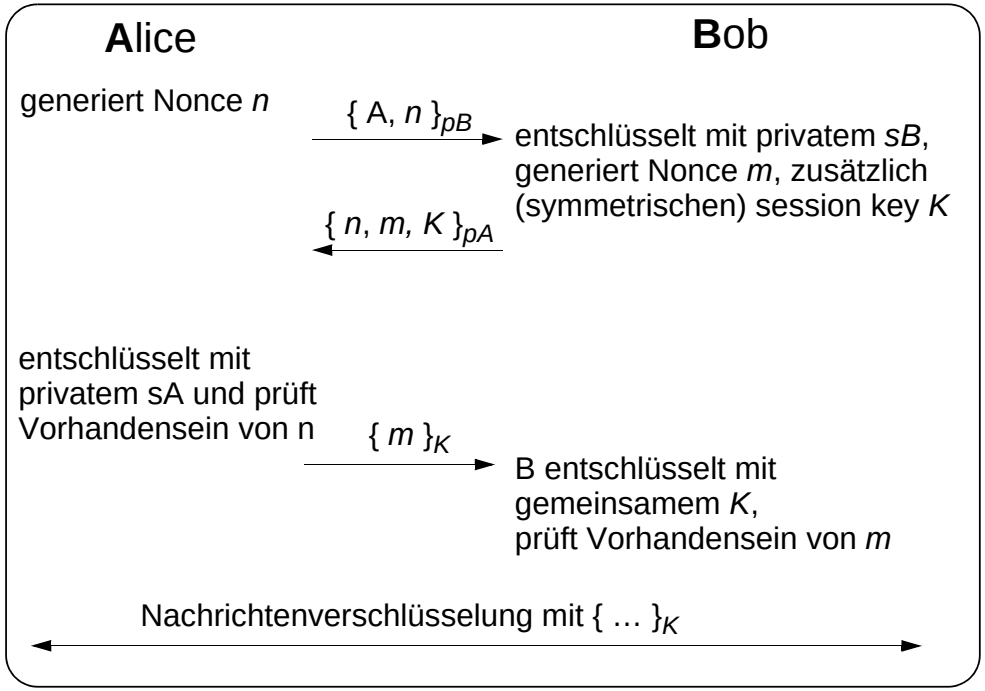
Key Server: Kennt alle public keys

- 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*

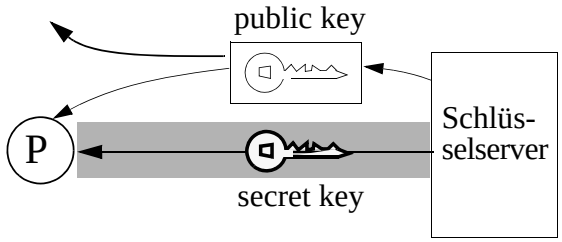
   - Empfänger akzeptiert Nachricht nur, wenn seine Zeit max. Δt abweicht.

   - 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!

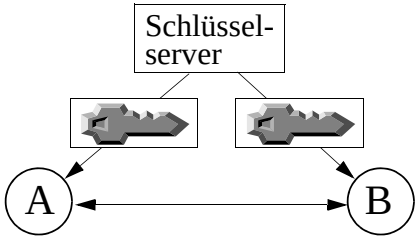
Schlüsselvergabe

- 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)

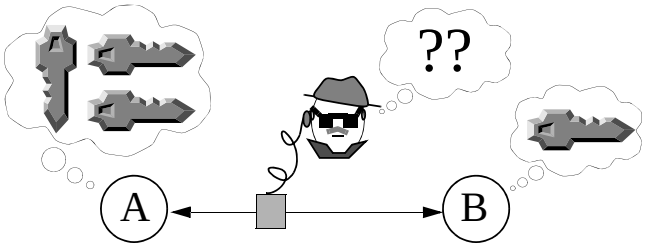
- Zur Generierung von temporären symmetrischen Schlüsseln ("session key")



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

Direkte Schlüsselvereinbarung

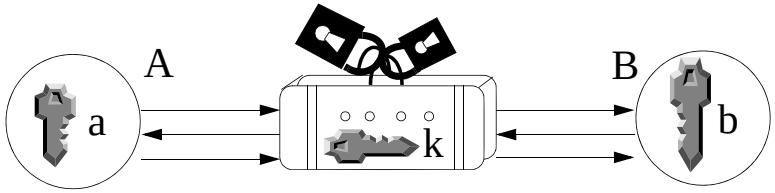


- Problem: A und B wollen sich über einen unsicheren Kanal auf einen gemeinsamen Schlüssel einigen, ohne einen Schlüsselserver zu verwenden

- 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: $\bar{\bar{x}} = x$)

gemeinsames Geheimnis

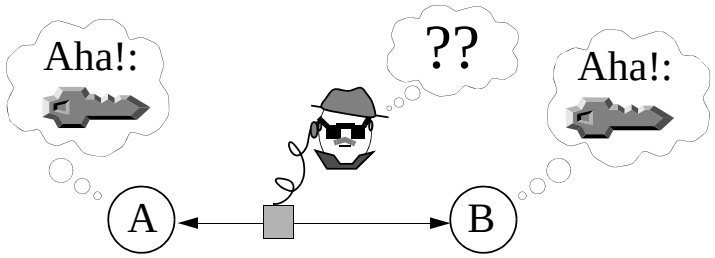
Beachte: k wird nie offen transportiert!

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

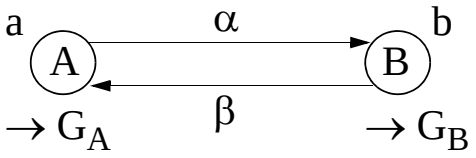


- Nutzung einer *Einwegfunktion*: $f(x) = c^x \bmod p$

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

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

Der Diffie-Hellman-Algorithmus



- wenig Nachrichten
- effizient

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

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

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

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

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

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

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

$G_A = G_B$

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

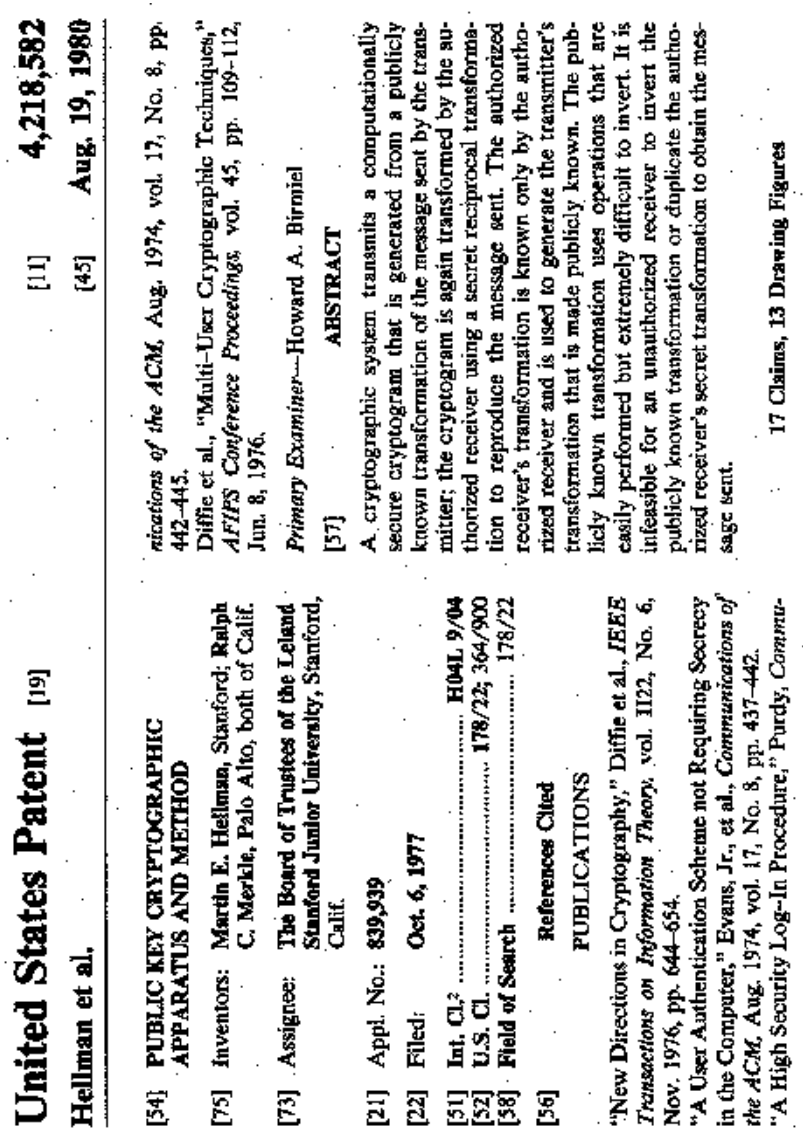
$$(c^b \bmod p)^a \bmod p = (c^a \bmod p)^b \bmod p$$

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

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

Bemerkungen:

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



United States Patent [19]
Hellman et al.

[11] 4,218,582
[45] Aug. 19, 1980

[54] PUBLIC KEY CRYPTOGRAPHIC APPARATUS AND METHOD

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

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

[21] Appl. No.: 839,939

[22] Filed: Oct. 6, 1977

[51] Int. Cl.² H04L 9/04
[52] U.S. Cl. 178/22; 364/900
[58] Field of Search 178/22

[56] References Cited
PUBLICATIONS

"New Directions in Cryptography," Diffie et al., *IEEE Transactions on Information Theory*, vol. II22, No. 6, Nov. 1976, pp. 644–654.
"A User Authentication Scheme not Requiring Secrecy in the Computer," Evans, Jr., et al., *Communications of the ACM*, Aug. 1974, vol. 17, No. 8, pp. 437–442.
"A High Security Log-In Procedure," Purdy, *Communications of the ACM*, Aug. 1974, vol. 17, No. 8, pp. 442–445.
Diffie et al., "Multi-User Cryptographic Techniques," *AFIPS Conference Proceedings*, vol. 45, pp. 109–112, Jun. 8, 1976.

Primary Examiner—Howard A. Birmiel

[57] ABSTRACT

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

17 Claims, 13 Drawing Figures

US4218582: Public key cryptographic apparatus and method

Inventors: Martin E. Hellman, Stanford; Ralph C. Merkle, Palo Alto

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

Abstract:

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

What is claimed is:

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

2. ...

...

17. ...

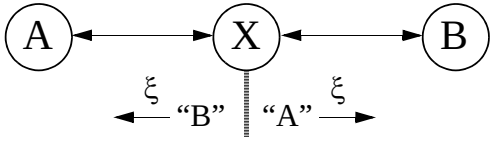
Sweet Little Secret G

- A und B könnten $G = G_A = G_B$ als symmetrischen Schlüssel zur Kodierung 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 z.B. im Sun-RPC-Protokoll gemacht
  - Motivation: G selbst so selten wie möglich benutzen

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

- Wie ist es aber bei *aktiven Angreifern*?

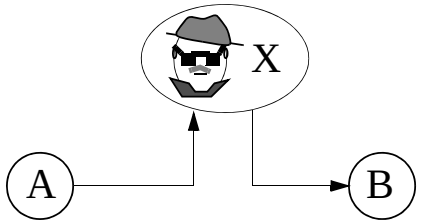


- "man in the middle"
- ein ξ für ein β bzw. α vormachen!

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

Aktive Angriffe durch Eindringen und Schlüsselfälschung

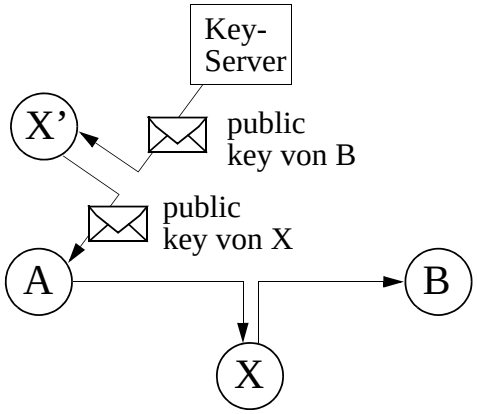
Szenario 1:



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

- Challenge-Response-Test nützt so nichts: X reicht Challenges einfach an den von ihm vorgetäuschten Partner weiter und miemt mit der abgefangenen Antwort die angenommene Identität

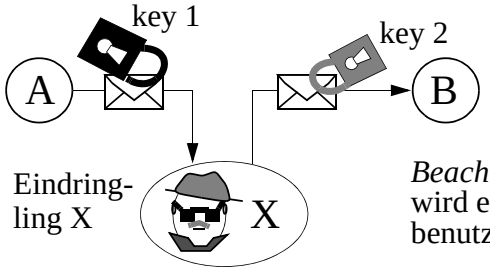
Szenario 2:



- kompromittierter Key-Server; Verschwörung X, X'
- 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 zunächst nur die "Hälfte" davon zurück
   - z.B. nur jedes zweite Bit (die "geraden" Bits)
   - B erwartet diese Hälfte der Antwort in weniger als t Zeiteinheiten
4) Ohne die andere Hälfte kann X diese nicht entschlüsseln und neu verschlüsseln
   (sofern X nicht erzwingen kann, dass key 1 = key 2 ist)
5) Erst nach t Zeiteinheiten sendet A die andere Hälfte
   - B setzt Schlüsseltexthälften zusammen und überprüft Antwort

---

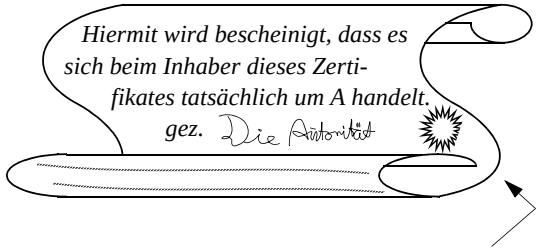
→ Gibt X die halbe Nachricht unverändert weiter, kann B diese nicht entschlüsseln → *Fälschung erkannt*

→ 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-Hellman-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...)?

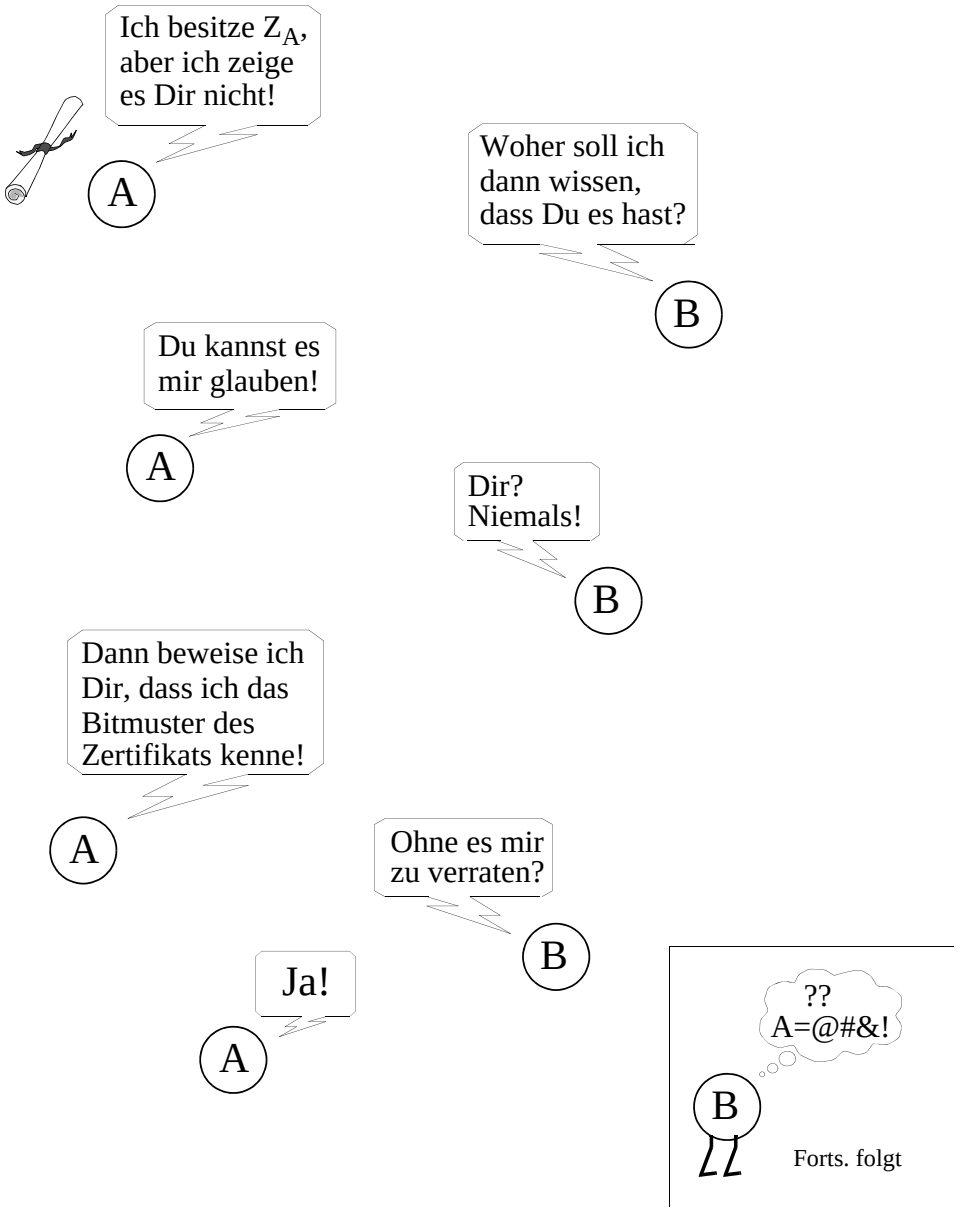
Authentifizierung mit Zertifikaten?

- 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 evtl. persönlich in Augenschein genommen (oder einem fremden Zertifikat vertraut)
  - Wenn B an der Identität von A zweifelt, wird B von A auf sein Zertifikat Z_A hingewiesen
    - Besitz des Zertifikates = Authentifizierung
  - Aber: A darf Z_A nie B zeigen - sonst könnte B es sich kopieren und sich fortan als A ausgeben!
    - in der digitalen Welt lassen sich Bitfolgen perfekt kopieren
    - wie vermeidet man “raubkopierte Zertifikate”?
  - 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?

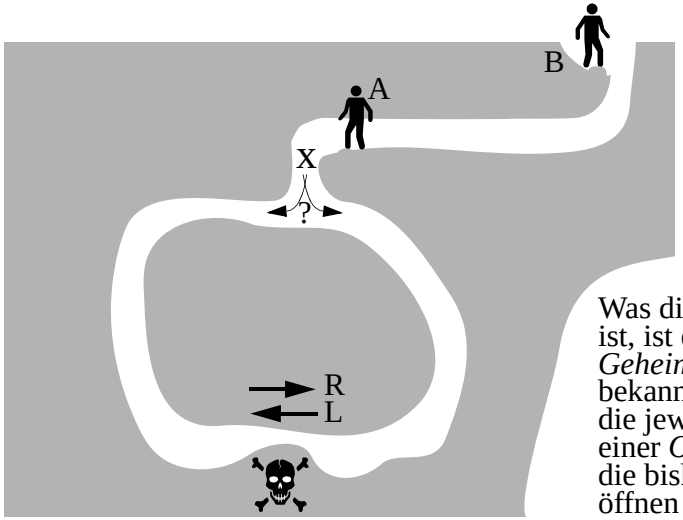


Geheime Zertifikate!

- 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 sich dies von A mit sA entschlüsseln lässt)

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

Ein Höhlengleichnis



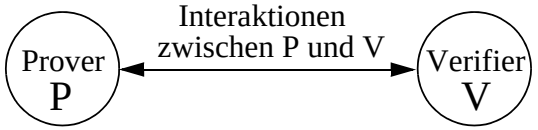
Der Höhlendämon lässt nur diejenigen die Engstelle lebendig passieren, die dort in der richtigen Richtung (L oder R) vorbeigehen.

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-Beweis

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



Hat Kenntnis einer bestimmten (“geheimen”) Information

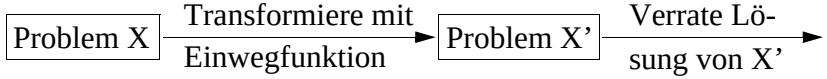
Überprüft, ob P diese Kenntnis wirklich hat

- P soll V (praktisch) *nicht betrügen* können: Wenn P die Information nicht hat, sollen seine Chancen, V zu überzeugen, verschwindend gering sein

- V soll *nichts* über die Kenntnis von P *erfahren*
  - V erfährt auch sonst nichts Relevantes von P, was V nicht auch alleine in Erfahrung bringen könnte

---

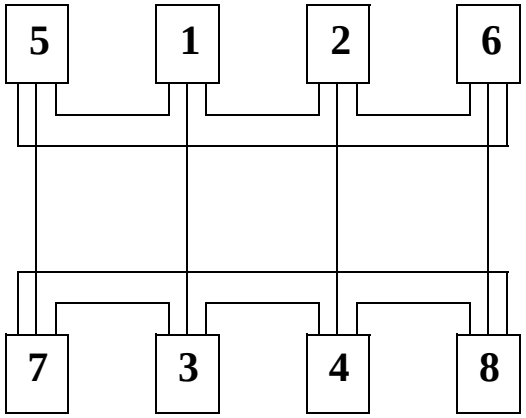
Idee: (geheime Information = Lösung eines schwierigen Problems X)



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

Beispiel: Isomorphie von Graphen

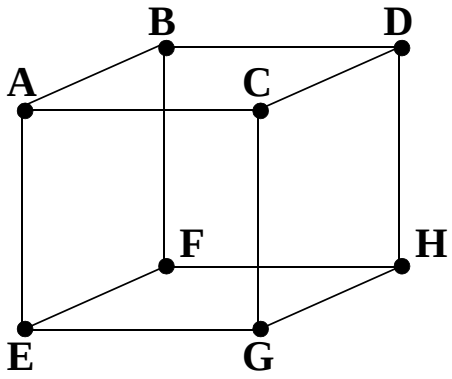
Bemerkung: Ob zwei grosse (z.B. in Form von Adjazenzmatrizen) gegebene Graphen G_1, G_2 topologisch isomorph ($G_1 \sim G_2$) sind (d.h. bis auf Umbenennung von Knoten und evtl. Kanten identisch sind), ist ein *schwieriges* Problem.



Hier nur ein kleines und daher einfaches (also unrealistisches) Beispiel

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

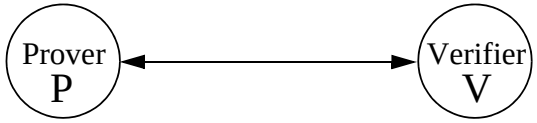
≡



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. Andere aber können $H \sim G_1$ 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 andere, von V nicht gewünschte, Alternative nachzuweisen - wieso?)
  - V kann den von P gelieferten Isomorphienachweis einfach verifizieren
  - P und V wiederholen alles n Mal, wobei von P jedesmal ein anderer "Zeuge" H konstruiert wird (Beweissicherheit = $1-2^{-n}$)
- 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 Relevantes gelernt hat, kann V sich anderen gegenüber auch nicht mit der Kenntnis schmücken
  - sich also *nicht für P ausgeben* (wenn die Kenntnis P identifiziert)

Grosse Graphen sind in der Praxis etwas unhandlich. Es gibt praktischere Ausprägungen des Zero-Knowledge-Verfahrens, z.B. das Protokoll von Fiat und Shamir. Dieses beruht auf der Schwierigkeit, die k-te Wurzel in einem Restklassenring zu berechnen.

Der Kerberos-Sicherheitsdienst

- Protokoll zur Schlüsselvergabe, Authentifizierung und Einrichtung sicherer Kommunikationskanäle
- Am MIT entwickelt im Rahmen eines ersten grossen Client-Server-Campusnetzes (ab 1986)
- Basiert auf Needham-Schroeder-Protokoll mit symmetrischen Schlüsseln

  R.M. Needham, M.D. Schroeder: *Using Encryption for Authentication in Large Networks of Computers.* CACM 21(12), pp. 993-999, 1978
- Public domain; es gibt auch kommerzielle Varianten
- Es gibt noch eine Reihe weiterer (neuerer) Systemdienste zur Erhöhung der Sicherheit in offenen vert. Systemen
  - z.B. ssh, VPN etc.

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

RFC 1510: *The Kerberos Network Authentication Service (V5)*, www.rfc-archive.org/getrfc.php?rfc=1510

Vgl. auch *Wikipedia*

Kerberos-Prinzipien

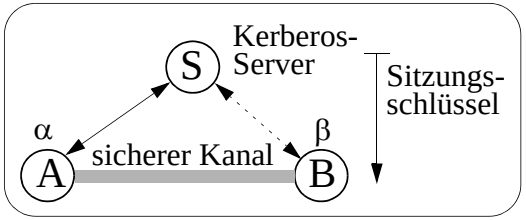
- Offenes Netz → Nachrichten prinzipiell unsicher
- Kommunikation erfolgt daher i.Allg. verschlüsselt und nur mit authentifizierten Partnern
  - Kenntnis des Sitzungsschlüssels als Authentitätsbeweis
- Passwörter niemals im Klartext übertragen
  - auch keine Passwortspeicherung

"Kerberos-Server"

- Benutzer, Clients und Server sind bei zentraler Instanz (Key Distribution Center: "KDC") akkreditiert
  - vereinbaren mit dem KDC auch ihren Geheimschlüssel ("master key")
  - ohne Akkreditierung keine Server-Berechtigungsscheine ("Tickets")
  - ohne Tickets kein Service
  - Ticket nur in Verbindung mit Authentitätsnachweis gültig
- Gültigkeit von Tickets / Sitzungsschlüsseln zeitlich befristet
- Mehrere Sicherheitsstufen möglich, z.B.:
  (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

- Gegenseitige Authentifizierung (via Kerberos Server)
- Verwendung eines Sitzungsschlüssels (“session key”)
- Ein chiffriertes Quadrupel $\{X, K, Ts, \Delta\}_\gamma$ heisst “Ticket”
  - Teilnehmer X, Sitzungsschlüssel K, Zeitstempel Ts, Gültigkeitsdauer Δ
  - Tickets kann man an andere (“vertrauenswürdige”) Instanzen weitergeben



Hier: Version 4; andere Kerberos-Versionen im Prinzip nur leicht unterschiedlich

$A \rightarrow S$: A, B

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

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

$B \rightarrow A$: $\{Ta + 1\}_K$

Sitzungsschlüssel

Zeitstempel

Gültigkeitsdauer des Sitzungsschlüssels

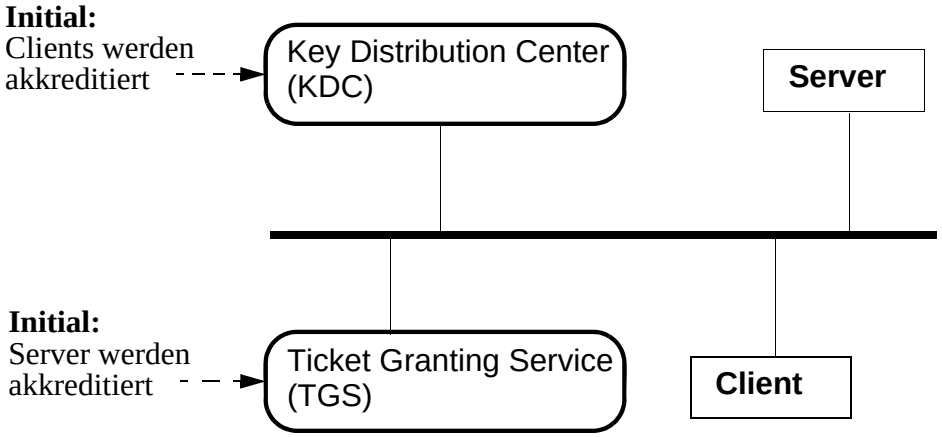
Geheime Schlüssel von A bzw. B

B sieht: A kennt K

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

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

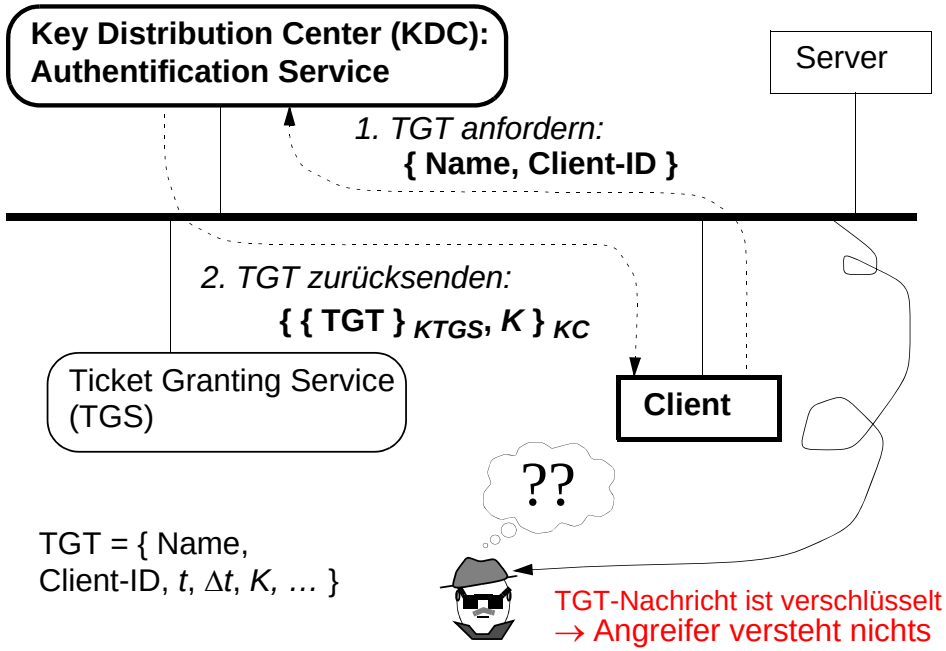
Kerberos: Akkreditierung



- Benutzer (Clients) 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 ($\rightarrow$ Lastverteilung)

Kerberos: TGT-Anforderung

- Client erwirbt zunächst ein Ticket Granting Ticket (TGT)



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

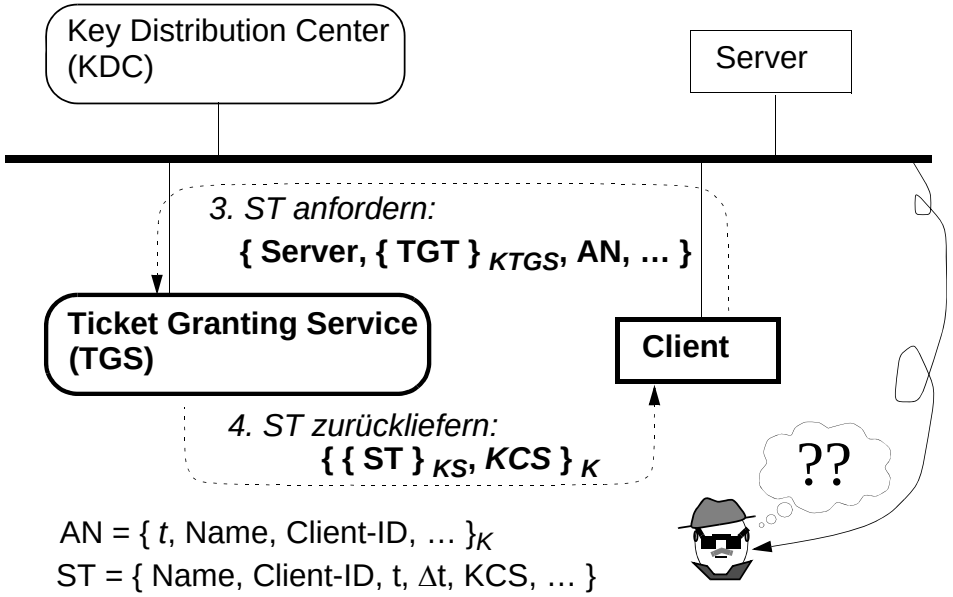
→ KDC-Nachricht ist authentisch: Nur KDC kennt noch Schlüssel KC !

→ Nur der echte Client kann TGT mittels KC nutzbar machen

→ Passwort verlässt Client-Rechner nicht

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

Kerberos: Service Ticket erwerben

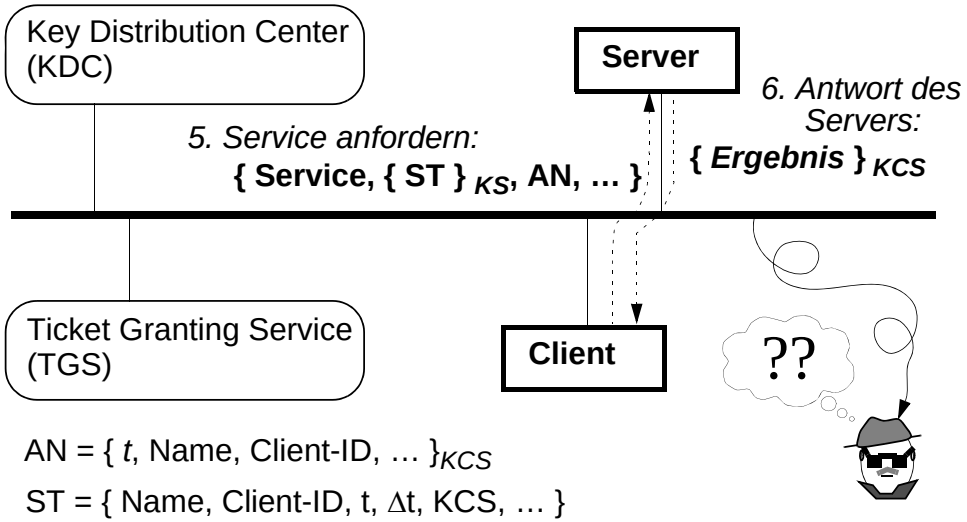


- Client: erstellt Authentizitätsnachweis AN = { t, Name, Client-ID, … } $_{K}$
- Client sendet an TGS { Server, { TGT } $_{KTGS}$, AN, … } als Request
- TGS: entschlüsselt TGT mit Schlüssel KTGS, erhält damit K;
  entschlüsselt AN mit K, vergleicht Inhalt mit TGT;
  erstellt Service Ticket ST = { Name,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

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

→ ST höchstens für Zeitspanne Δt gültig

Kerberos: Nutzung des Service

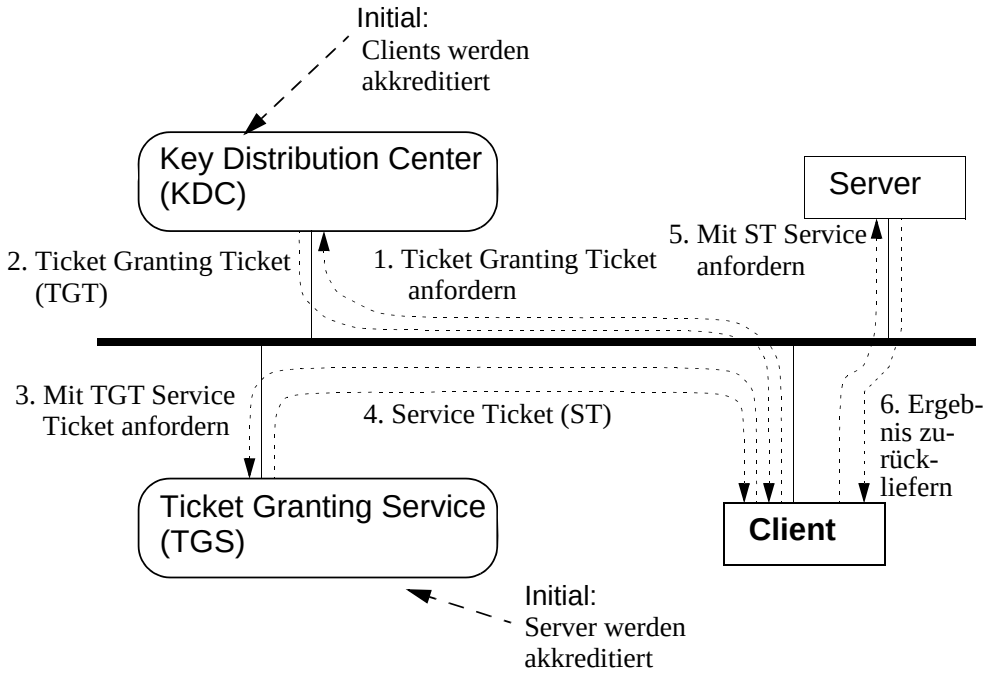


- Client: erstellt Authentizitätsnachweis AN = { t, Name, Client-ID, … } $_{KCS}$
- Client an Server: sendet { Service, { ST } $_{KS}$, AN, … } als Service-Request
- Server: entschlüsselt ST mit KS, erhält damit KCS;
  entschlüsselt AN mit KCS, vergleicht Inhalt mit ST;
  leistet Service und erzeugt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

→ Folgedialoge zwischen Client und Server mittels KCS verschlüsselbar

→ ST als Einmal-Ticket oder evtl. 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 i. Allg. für mehrere Anfragen beim Ticket-Service gültig

Kerberos - weitere Aspekte

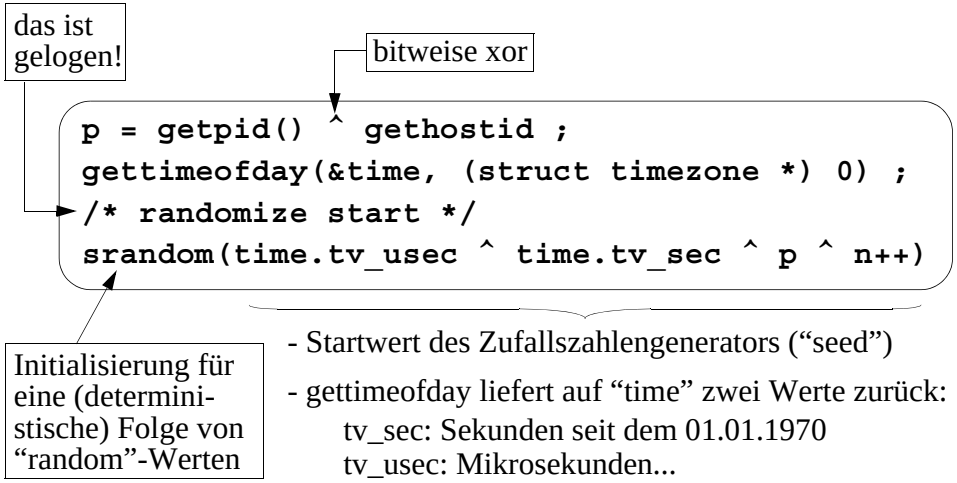
- Nachrichten enthalten noch weitere (technische) Angaben
  - z.B. Versionsnummer, Nachrichtentyp, Prüfsumme, Netzwerkadressen,...
- Es gibt dezentrale Zuständigkeitsbereiche (“realms”)
  - lok. KDC vermittelt Zugangsticket zu KDC eines fremden Bereichs
- Kerberos-Software enthält u.a.:
  - Library mit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

Kerberos - Sicherheitsaspekte

- KDC und TGS müssen geschützt werden
  - z.B. gegen unbefugtes Lesen der Datenbank, Verändern der Daten, denial of service,...
- Tickets sollen 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 evtl. mehrfachen Ticketmissbrauch
- Replays sind innerhalb der Gültigkeitsdauer (typw.: einige Minuten bis Stunden) prinzipiell möglich!
  - Server sollte alte, noch gültige Tickets speichern, um Replays erkennen zu können
- “Erster” Schlüssel basiert auf einem Passwort → Off-line-Attacke durch Raten gängiger Passworte
- Angriffe ausserhalb von Kerberos
  - fremde Tickets lesen (Netz-Sniffer, Superuser-Rechte beschaffen,...)
  - “Hijacking” von TCP-Verbindungen
  - gefälschte Kerberos-Software mit Hintertüren auf Download-Servern

Schlüsselgenerierung

- Die Schlüsselgenerierung von Kerberos (z.B. für TGT oder Sitzungsschlüssel) funktionierte ursprünglich so:



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

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

- Der Algorithmus ist inzwischen "verbessert"...

- Und die Moral der Geschichte?