

Replays

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

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

- > (fast) alle Nachrichten sind verschieden
- aufwendiges Protokoll aus mehreren Nachrichten

2) Verwendung von mitkodierten *Sequenznummern*

- nur bei einer Nachrichtenfolge zwischen 2 Prozessen möglich

3) Mitverschlüsseln der *Absendezeit*

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

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

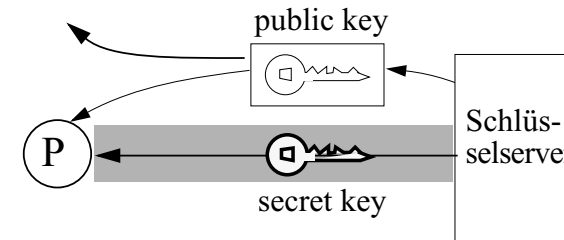
- Zeitfenster ΔT geschickt wählen!

- Nachrichtenlaufzeiten berücksichtigen!
- zu gross -> unsicher durch mögliche Replays
- zu klein -> exakte oder häufige Uhrensynchronisation nötig (z.B. vor jede Nachricht oder nach einem ‘reject’)

- Angreifer darf Zeitservice nicht manipulieren können!

Schlüsselvergabe

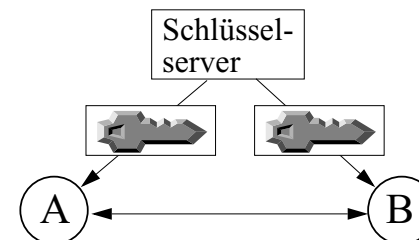
- Zur Vergabe eines Paares von public-, secret-keys:



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

- Zur Generierung von temporären symmetrischen Schlüsseln (“session key”)

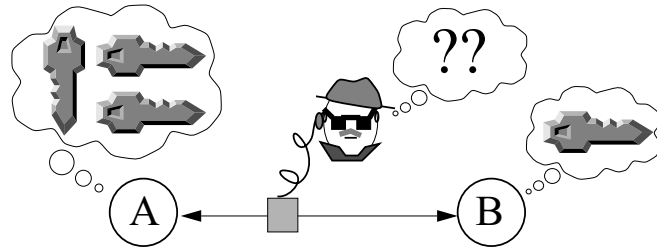
-> erhöhte Sicherheit gegen Angreifer



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

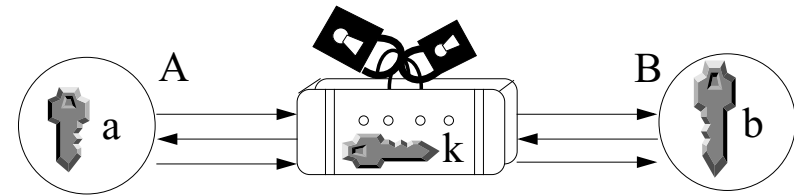
- Schlüsselserver kann session keys nach Übertragung bei sich löschen
- Aufwendiges Public-key-Verfahren nur ein Mal pro “Session”, tatsächliche Nachrichten zwischen A und B effizienter per symmetr. Schlüssel

Direkte Schlüsselvereinbarung



- Problem: A und B wollen sich über einen unsicheren Kanal auf einen gemeinsamen Schlüssel einigen, ohne einen Schlüsselservers 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. $A \rightarrow B: \{k\}_a$ a und b sind "lokal erfunden"
4. B verschlüsselt dies mit seinem Schlüssel b
5. $B \rightarrow A: \{\{k\}_a\}_b$
6. A entschlüsselt mit seinem Schlüssel a :
 $\{\{\{k\}_a\}_b\}_a = \{\{\{k\}_a\}_a\}_b = \{k\}_b$

Forderung!

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

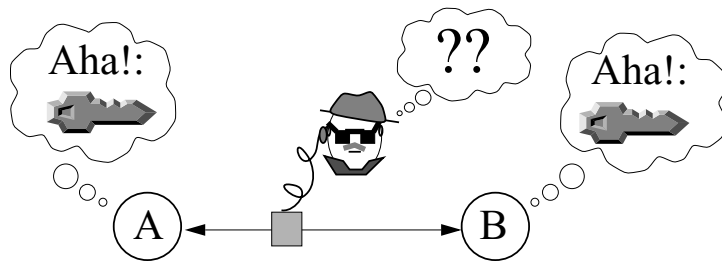
Beachte: k wird nie offen transportiert!

Denkübung: Geht hier *xor* mit "one-time pads" a, b ?

- *xor* erfüllt die Forderung (ist assoziativ und kommutativ)
- *xor* mit one-time pads ist sicher (wirklich?) und effizient
- aber: Wenn Schritt 3 ($\{k\}_a$) und Schritt 5 ($\{\{k\}_a\}_b$) abgehört wird, dann kann daraus der Schlüssel b ermittelt werden, so dass aus dem abgehörten Schritt 7 ($\{k\}_b$) das geheime k ermittelt werden kann!
- gibt es anstelle von *xor* andere (sichere!) Verschlüsselungsoperationen?

Schlüsselvereinbarung mit Diffie-Hellman-Verfahren

Ziel: A und B sollen sich über einen unsicheren Kanal auf ein gemeinsames "Geheimnis" G einigen, ohne dass ein Angreifer es erfährt



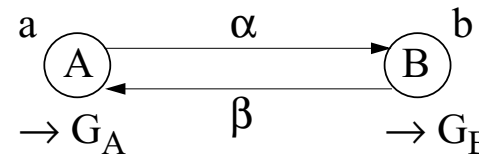
- Nutzung einer *Einwegfunktion*: $f(x) = c^x \bmod p$
 ($1 < c < p$; i.a. ist p eine grosse Primzahl)

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

- im RPC-Protokoll von Sun wird z.B. $c=3$ gewählt und $p = d4a0ba0250b6fd2ec626e7efd637df76c716e22d0944b88b$ (hex.); eine Zahl aus 192 Bits (die Parameter c und p sind kein Geheimnis)

- gelegentlich wird für p auch ein Produkt aus zwei grossen Primzahlen empfohlen, oder es wird $p = 2^n$ gewählt, da dann die mod-Operation besonders einfach zu berechnen ist

Der Algorithmus



- wenig Nachrichten
 - effizient

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

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

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

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

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

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

$$G_A = G_B$$

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: $(k \bmod p)^n \bmod p = k^n \bmod p$ ← Restklassenarithmetik...

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

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. IT22, No. 6, Nov. 1976, pp. 644-654.

"A User Authentication Scheme not Requiring Secrecy in the Computer," Evans, Jr., et al., *Communications of the ACM*, Aug. 1974, vol. 17, No. 8, pp. 437-442.

"A High Security Log-In Procedure," Purdy, *Communi-*

ications of the ACM, Aug. 1974, vol. 17, No. 8, pp. 442-445.

Diffie et al., "Multi-User Cryptographic Techniques," *AFFS 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 publicly known transformation that is made publicly known. The publicly known transformation uses operations that are easily performed but extremely difficult to invert. It is infeasible for an unauthorized receiver to invert the publicly known transformation or duplicate the authorized receiver's secret transformation to obtain the message sent.

17 Claims, 13 Drawing Figures

US4218582: Public key cryptographic apparatus and method

Inventor(s): Hellman; Martin E. , Stanford, CA Merkle; Ralph C. , Palo Alto, CA

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

Abstract:

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

What is claimed is:

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

2. ...

...

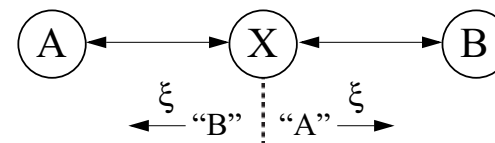
17. ...

Sweet Little Secret G

- A und B könnten beide $G = G_A = G_B$ als symmetrischen Schlüssel (z.B. DES) zur Verschlüsselung ihrer Nachrichten verwenden
- *Besser*: G nur als Schlüssel verwenden, um einen zufällig bestimmten session key zu kodieren und dem Kommunikationspartner diesen mitzuteilen
 - so wird es z.B. im Sun-RPC-Protokoll gemacht
 - Motivation: G so selten wie möglich benutzen

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

- Wie ist es aber bei *aktiven Angreifern*?

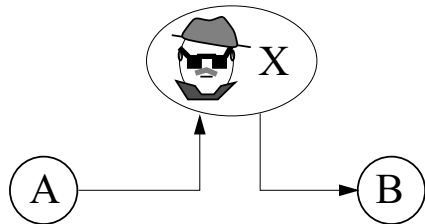


- "man in the middle"
- ein ξ für ein β bzw. α vormachen!

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

Aktive Angriffe durch Eindringen und Schlüsselfälschung

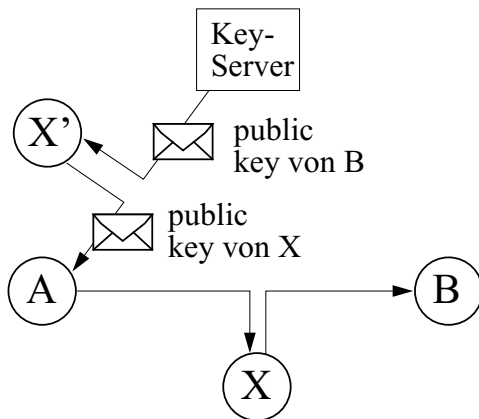
Szenario 1:



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

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

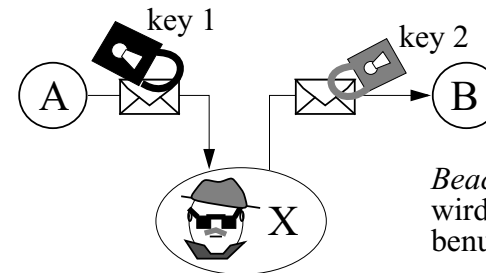
Szenario 2:



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

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

Erkennen von Eindringlingen



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

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

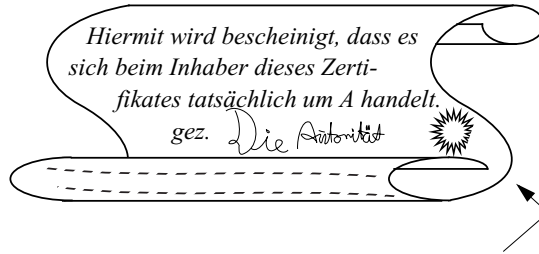
→ Gibt X die halbe Nachricht unverändert weiter, kann B diese nicht entschlüsseln → *Fälschung erkannt*

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

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

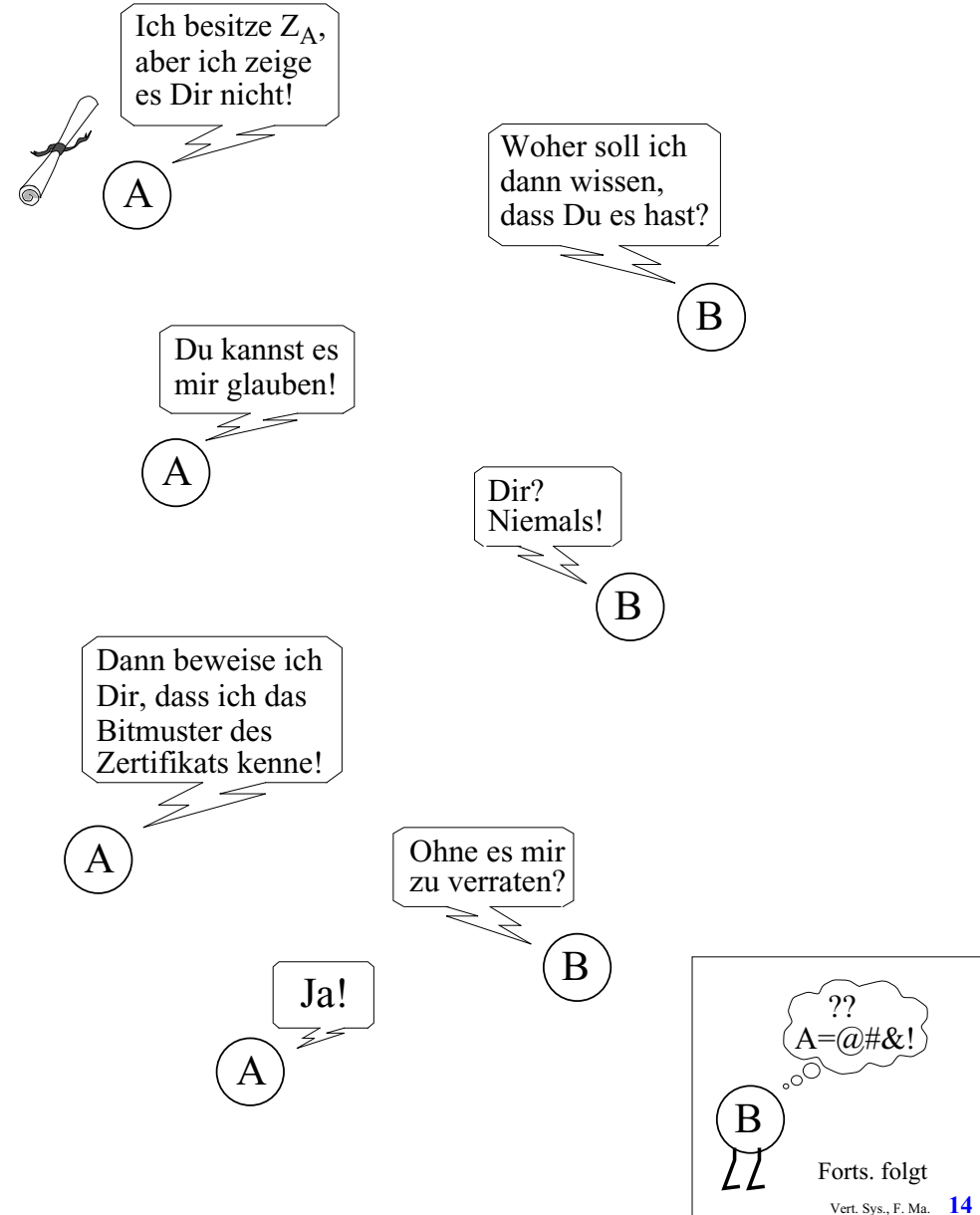
Authentifizierung mit Zertifikaten ?

- Die Idee:



- A lässt sich einmalig von einer Autorität ein *Zertifikat* Z_A mitgeben (sollte von der Autorität signiert sein)
 - Autorität gilt als vertrauenswürdig und hat A ggf. persönlich in Augenschein genommen (oder einem fremden Zertifikat vertraut)
- Wenn B an 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!
 - wie vermeidet man "raubkopierte Zertifikate"?
 - in der digitalen Welt lassen sich Bitfolgen perfekt kopieren (im Unterschied zu "fälschungssicheren Ausweisen")
- Z_A muss offenbar ein *Geheimnis* bleiben, das ausser der Autorität und A niemand kennt!
- Taugt ein solches Geheimnis als Zertifikat??
 - wie beweist man den Besitz eines Zertifikates, ohne es zu zeigen?

Geheime Zertifikate ?



Forts. folgt

Geheime Zertifikate !

- Im Prinzip wissen wir schon, dass das geht: Der secret key s_A eines asymmetrischen Verfahrens stellt ein solches Zertifikat dar

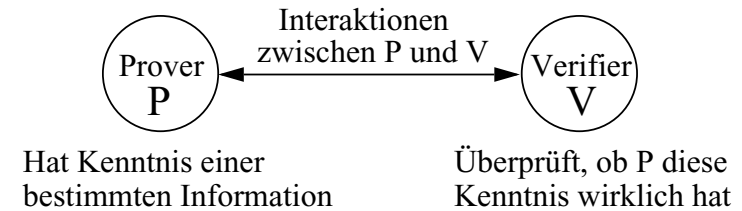
- braucht von A nicht verraten zu werden
- B kann dennoch überprüfen, ob A das Zertifikat hat (z.B. indem sich B von A etwas mit s_A verschlüsseln lässt und anschliessend durch Anwenden von p_A prüft; oder indem B ein $\{M\}_{p_A}$ an A schickt und sich dies von A mit s_A entschlüsseln lässt)

- Eine andere Realisierung geht mit “zero knowledge”

- beweist Kenntnis eines Geheimnisses G, ohne relevante Information preiszugeben

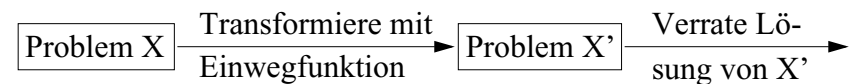
Zero-Knowledge-Proofs

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



- P kann V (*praktisch*) *nicht betrügen*: Wenn P die Information nicht hat, sind seine Chancen, V zu überzeugen, verschwindend gering
- V *erfährt nichts* über die eigentliche Kenntnis von P
- V erfährt auch sonst nichts Relevantes von P, was V nicht auch alleine in Erfahrung bringen könnte

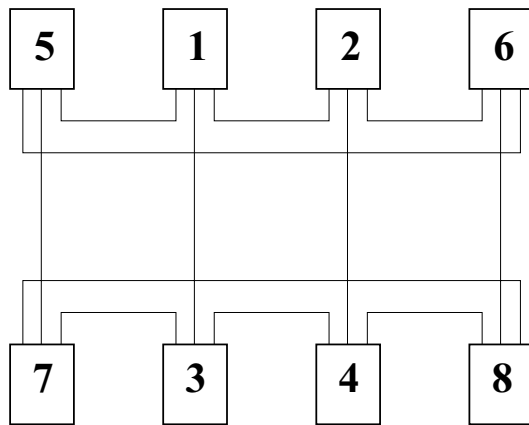
Idee:



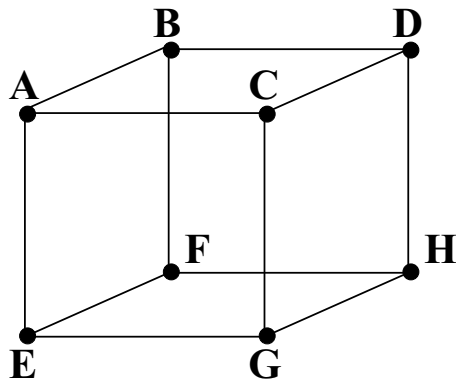
(Wobei die Lösung von X' die Lösung von X logisch impliziert, sie jedoch nicht effektiv-konstruktiv liefert)

Beispiel: Isomorphie von Graphen

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



≡



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

Überprüfung eines durch eine Knotenzuordnung gegebenen Isomorphismus ist allerdings "einfach"!

Zero-Knowledge mit Graphisomorphie

- P behauptet, einen Beweis zu haben, dass zwei gegebene Graphen G_1, G_2 isomorph sind, möchte den Beweis aber nicht verraten
- Folgendes Protokoll *überzeugt* V davon:
 - P erzeugt durch zufällige Permutation der Knoten einen Graphen H mit $H \sim G_1$ (und damit $H \sim G_2$). Für P ist dies einfach. 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
- 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?)
- 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!

zufällig; bzw. von P nicht vorhersehbar

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 (1986). 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 des Athena-Projekts
 - war dort ab 1986 im Einsatz
- Basiert auf Needham-Schroeder-Protokoll mit symmetrischen Schlüsseln (i.a. DES)
- Public domain; es gibt auch kommerzielle Varianten
- In heutigen “offenen” verteilten Systemen gibt es noch andere Systemdienste zur Erhöhung der Sicherheit
 - z.B. ssh, VPN etc.

erstes grosses Client-Server-Campusnetz

R.M. Needham, M.D. Schroeder: *Using Encryption for Authentication in Large Networks of Computers*. CACM 21(12), pp. 993-999, 1978

J.I. Schiller: *Sicherheit im Daten-Nahverkehr*. Spektrum der Wissenschaften 1/1995, pp. 50-57, Januar 1995

B. Clifford Neuman and Theodore Ts'o: *Kerberos: An Authentication Service for Computer Networks*. IEEE Communications Magazine, Volume 32, Number 9, pp. 33-38, September 1994.

RFC 1510: *The Kerberos Network Authentication Service (V5)*,
<ftp://ftp.isi.edu/in-notes/rfc1510.txt>



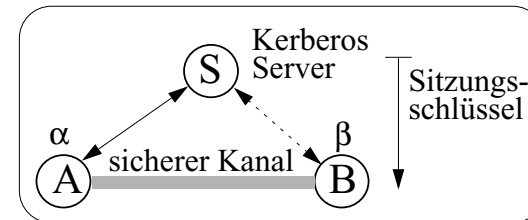
Kerberos-“Philosophie”

- Offenes Campusnetz → Nachrichten prinzipiell unsicher
- Kommunikation daher i.a. verschlüsselt und nur mit authentifizierten Partnern
 - Kenntnis des Sitzungsschlüssels als Authentizitätsbeweis
- Passwörter niemals im Klartext übertragen
 - auch keine Passwortspeicherung
- Alle Benutzer, Clients und Server sind bei zentraler Instanz (Key Distribution Center: “KDC”) akkreditiert
 - vereinbaren mit dem KDC auch ihren Geheimschlüssel (“master key”)
 - ohne Akkreditierung keine Server-Berechtigungsscheine (“Tickets“)
 - ohne Tickets kein Service
 - Ticket nur in Verbindung mit Authentizitätsnachweis gültig
- Gültigkeit von Tickets / Sitzungsschlüsseln zeitlich befristet
- Drei Sicherheitsstufen möglich
 - (1) Authentifizierung nur bei Einrichtung eines Kommunikationskanals
 - (2) Authentifizierung bei jeder Nachricht zwischen A und B
 - (3) zusätzlich Verschlüsselung der Nachrichten

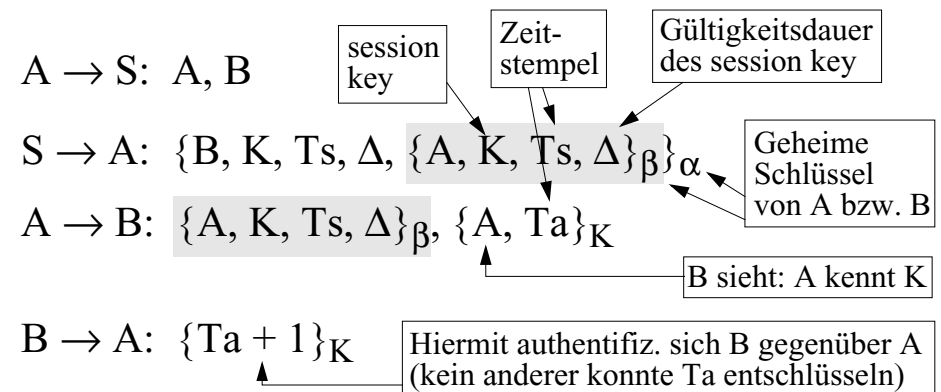
“Kerberos-Server”

Kerberos-Anwendungsbeispiel: Einrichtung eines sicheren Kanals

- Wechselseitige Authentifizierung (via Kerberos Server)
- Verwendung eines Sitzungsschlüssels (“session key”)
- $\{X, K, Ts, \Delta\}_\gamma$ heisst “Ticket”
 - Tickets kann man an andere (“vertrauenswürdige”) Instanzen weitergeben

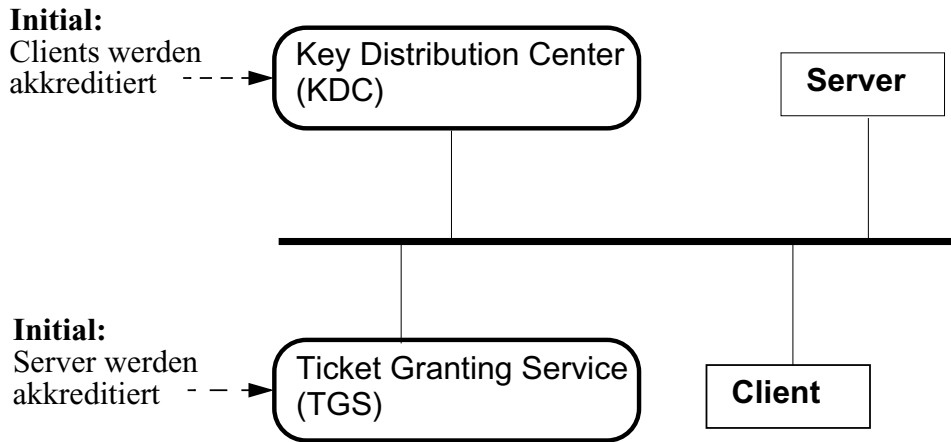


Hier: Version 4; spätere Versionen im Prinzip nur leicht unterschiedlich



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

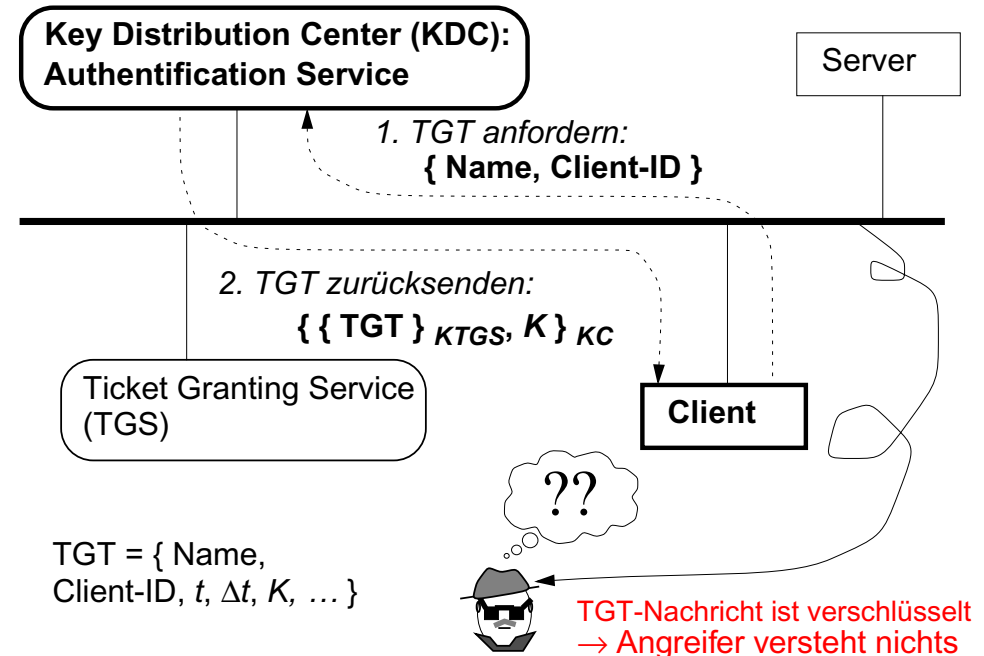
Kerberos: Akkreditierung



- Benutzer und deren Passwörter (= Schlüssel) werden dem KDC bekannt gemacht
- TGS und dessen geheimer Schlüssel werden ebenfalls beim KDC akkreditiert
- Server und deren geheime Schlüssel werden dem TGS bekannt gemacht
 - es kann mehrere TGS-Server geben (→ Lastverteilung)

Kerberos: TGT-Anforderung

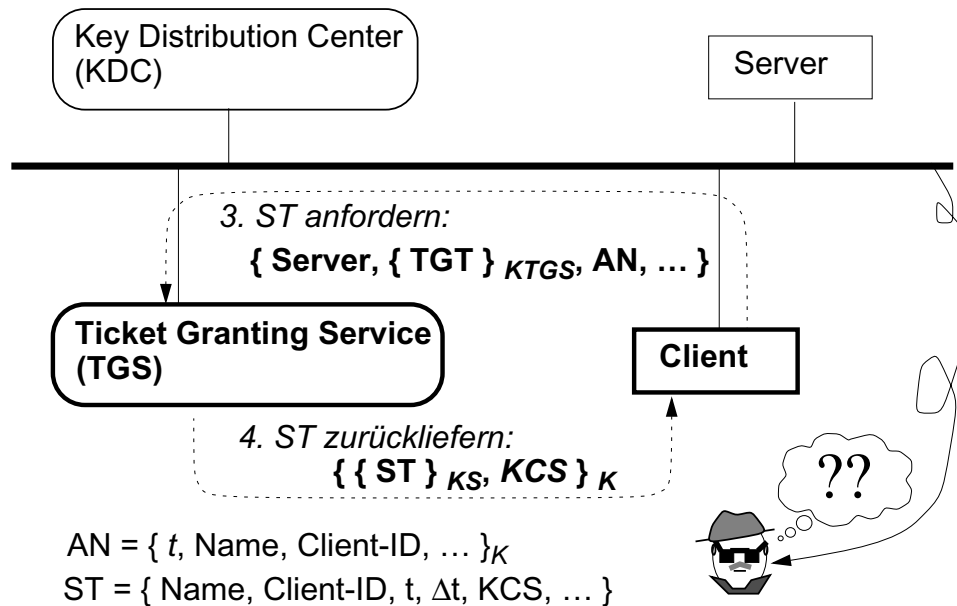
- Client erwirbt zunächst ein Ticket Granting Ticket (TGT)



- Client an KDC: sendet $\{ \text{Name, Client-ID} \}$ im Klartext
- KDC: wählt K ; erstellt $\text{TGT} = \{ \text{Name, Client-ID, t, Δt, K, ...} \}$
- KDC an Client: sendet $\{ \{ \text{TGT} \}_{K_{TGS}}, K \}_{K_C}$ zurück;
 $K_C = h(\text{Passwort}); K_{TGS} = \text{TGS-Schlüssel}; K = \text{Sitzungsschlüssel}$
- Client: gewinnt $\{ \text{TGT} \}_{K_{TGS}}$ 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 K_C !
- Nur der echte Client kann TGT mittels K_C nutzbar machen
- Passwort verlässt Client-Rechner nicht und wird sofort wieder gelöscht
- TGT ist verschlüsselt, nur für Zeitspanne Δt gültig, geht nur an Client

Kerberos: Service Ticket erwerben

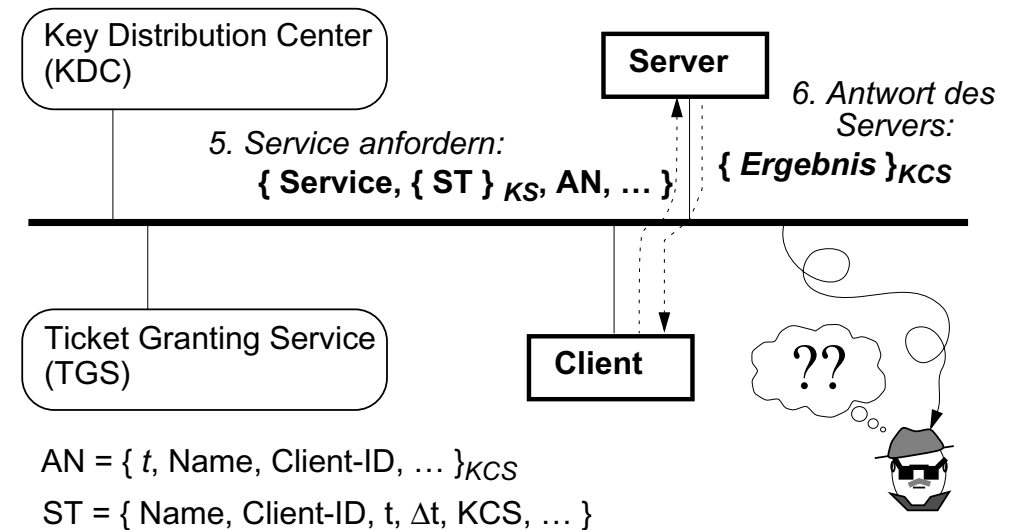


- Client: erstellt Authentizitätsnachweis $\text{AN} = \{ t, \text{Name}, \text{Client-ID}, \dots \}_K$
- Client sendet an TGS $\{ \text{Server}, \{ \text{TGT} \}_{K_{TGS}}, \text{AN}, \dots \}$ als Request
- TGS: entschlüsselt TGT mit Schlüssel K_{TGS} , erhält damit K ; entschlüsselt AN mit K , vergleicht Inhalt mit TGT; erstellt Service Ticket $\text{ST} = \{ \text{Name}, \text{Client-ID}, t, \Delta t, K_{CS}, \dots \}$
- TGS sendet an Client $\{ \{ \text{ST} \}_{K_S}, K_{CS} \}_K$ zurück
- Client: gewinnt $\{ \text{ST} \}_{K_S}$ und K_{CS} durch Entschlüsselung mit K :
 - (chiffriertes) ST berechtigt zur Nutzung des Servers
 - K_{CS} 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

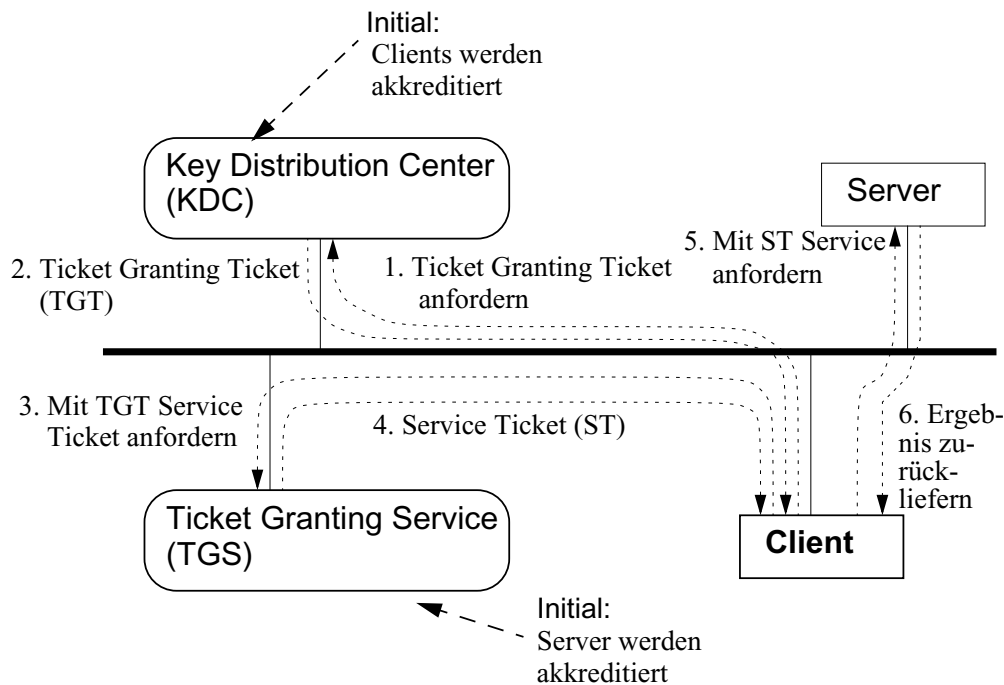


- Client: erstellt Authentizitätsnachweis $\text{AN} = \{ t, \text{Name}, \text{Client-ID}, \dots \}_{K_{CS}}$
- Client an Server: sendet $\{ \text{Service}, \{ \text{ST} \}_{K_S}, \text{AN}, \dots \}$ als Service Request
- Server: entschlüsselt ST mit K_S , erhält damit K_{CS} ; entschlüsselt AN mit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

→ Folgedialoge zwischen Client und Server mittels K_{CS} verschlüsselbar

→ ST als Einmal-Ticket oder ggf. innerhalb Δt mehrfach nutzbar

Kerberos: Protokollübersicht



- Protokoll ist zweistufig:

- Client kommuniziert nur selten mit dem KDC (1,2) → eigentlicher Geheimschlüssel (Passwort-basiert) wird nur selten benutzt
- ein TGT ist für mehrere Anfragen beim Ticket-Service gültig

Kerberos - weitere Aspekte

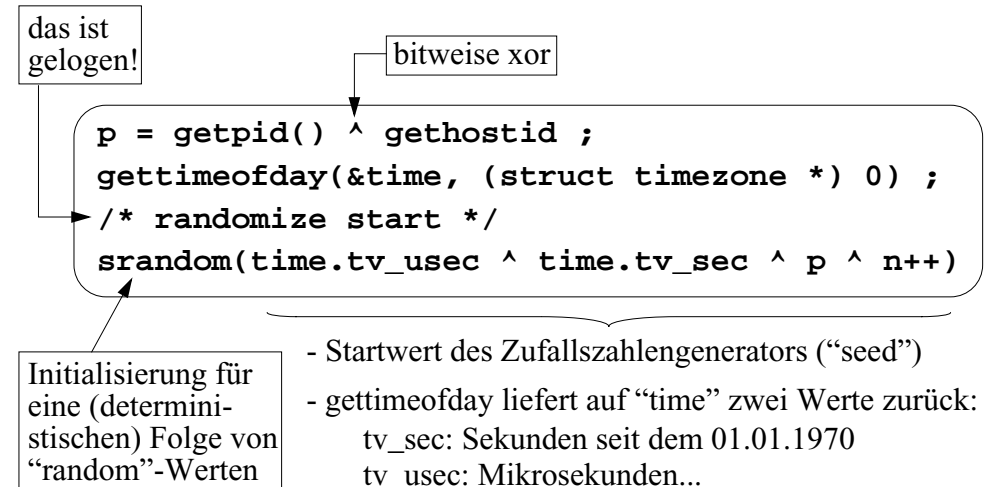
- Nachrichten enthalten noch weitere (technische) Angaben
 - z.B. Versionsnummer, Nachrichtentyp, Prüfsumme, Netzwerkadresse...
- Es gibt dezentrale Zuständigkeitsbereiche (“realms”)
 - lok. KDC vermittelt Zugangsticket zu KDC eines fremden Bereichs
- Kerberos-Software enthält u.a.:
 - Library mit Routinen,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

Kerberos - Sicherheitsaspekte

- KDC und TGS müssen geschützt werden
 - z.B. gegen unbefugtes Lesen der Datenbank, Verändern der Daten, Einschleichen, denial of service...
- Tickets müssen vom Client in einem “sicheren Speicherbereich” aufbewahrt werden
 - Master key (aus Passwordeingabe des Benutzers abgeleitet) wird sobald wie möglich aus dem Speicher gelöscht
- Uhren der Kommunikationspartner und der Kerberos-Server müssen “verlässlich” synchronisiert werden
 - innerhalb eines gewissen Toleranzintervalls von einigen Minuten
 - Störung des Uhrenabgleichs erlaubt ggf. mehrfachen Ticketmissbrauch
- Replays sind innerhalb der Gültigkeitsdauer (typw.: einige Minuten bis Stunden) prinzipiell möglich!
 - Server sollte alte, noch gültige Tickets speichern, um Replays ggf. erkennen zu können
- Auf Public-Domain-Servern (und CDs etc.) könnte gefälschte Software vorhanden sein (“trojanische Pferde”)
- “Erster” Schlüssel basiert auf einem Passwort → Off-line-Attacke durch Raten gängiger Passworte
- Hintertüren ausserhalb von Kerberos
 - fremde Tickets lesen (Netz-sniffer, Superuser-Rechte beschaffen...)
 - “Hijacking” von TCP-Verbindungen

Schlüsselgenerierung

- Die Schlüsselgenerierung von Kerberos (z.B. für TGT oder session keys) 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.a. nur in wenigen Bits von n
- Könnte jemand vielleicht absichtlich die Uhr des Servers auf einen falschen (d.h. bekannten) Wert synchronisieren?
 - welche Granularität hat eigentlich die Uhr?
- Der Algorithmus ist inzwischen “verbessert”...
- Und die Moral der Geschichte?