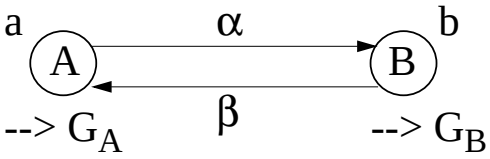
Der Algorithmus



- wenig Nachrichten
- effizient

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

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

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

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

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

a = 3 --> $\alpha = 6$ --> $G_B = 6^4 \bmod 7 = 1$

b = 4 --> $\beta = 2$ --> $G_A = 2^3 \bmod 7 = 1$

$G_A = G_B$

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...

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

Bemerkungen:

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

United States Patent [19]

Hellman et al.

[11] 4,218,582

[45] Aug. 19, 1980

[54] PUBLIC KEY CRYPTOGRAPHIC APPARATUS AND METHOD

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

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

[21] Appl. No.: 839,939

[22] Filed: Oct. 6, 1977

[51] Int. Cl.[2] H04L 9/04

[52] U.S. Cl. 178/22; 364/900

[58] Field of Search 178/22

[56] References Cited

PUBLICATIONS

"New Directions in Cryptography," Diffie et al., *IEEE Transactions on Information Theory*, vol. 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

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. ...

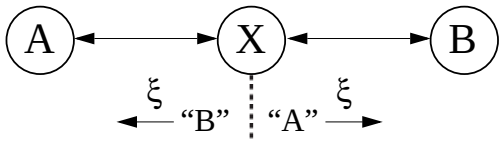
Sweet Little Secret G

- A und B könnten beide $G = G_A = G_B$ als symmetrischen DES-Schlüssel zur Verschlüsselung ihrer Nachrichten verwenden

- *Besser*: G nur als Schlüssel verwenden, um einen zufällig bestimmten Session-key zu kodieren und dem Kommunikationspartner diesen mitzuteilen
  - so wird es im Sun-RPC-Protokoll gemacht
  - Motivation: G so selten wie möglich benutzen

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

- Wie ist es aber bei *aktiven Angreifern*?

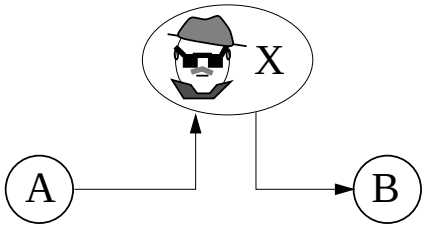


- "man in the middle"
- ein ξ für ein β bzw. α vormachen!

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

Aktive Angriffe durch Eindringen und Schlüsselfälschung

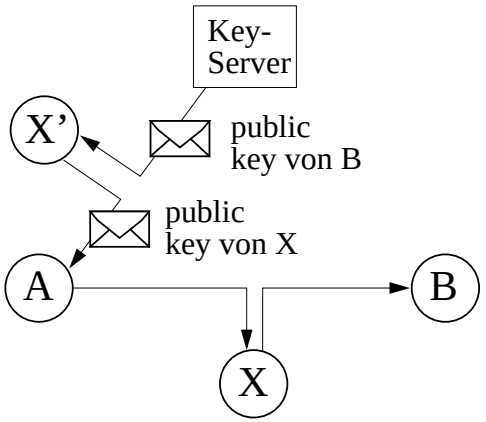
Szenario 1:



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

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

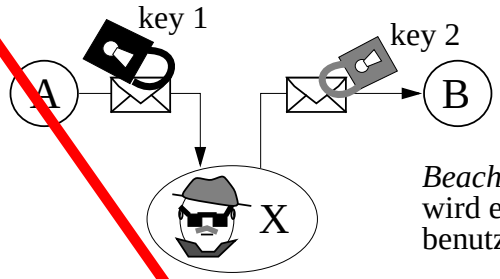
Szenario 2:



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

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

Erkennen von Eindringlingen



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

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

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

---

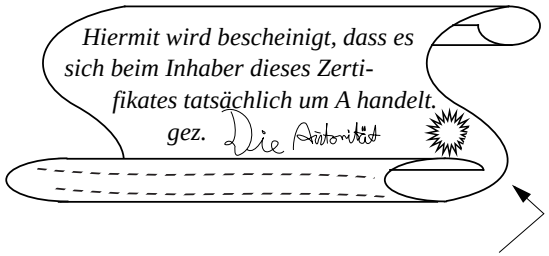
--> Gibt X die halbe Nachricht unverändert weiter, kann B diese nicht entschlüsseln --> *Fälschung erkannt*

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

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

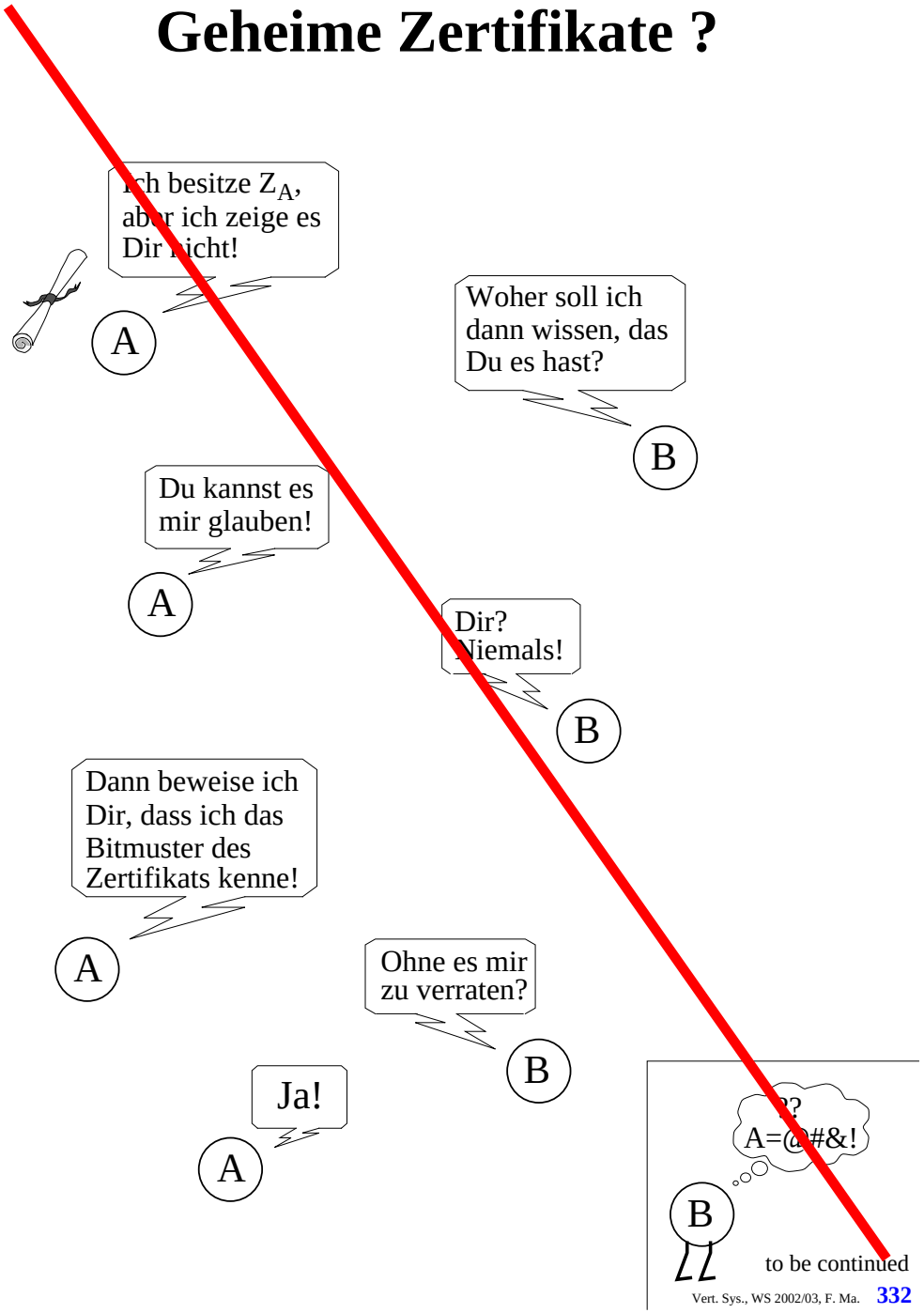
Authentifizierung mit Zertifikaten ?

- Die Idee:



- A lässt sich einmalig von einer Autorität ein *Zertifikat* Z_A mitgeben (sollte von der Autorität signiert sein)
  - Autorität gilt als vertrauenswürdig und hat A ggf. persönlich in Augenschein genommen (oder einem fremden Zertifikat vertraut)
- Wenn B an der Identität von A zweifelt, weist A B auf sein Zertifikat Z_A hin
  - Besitz des Zertifikates = Authentifizierung
- Aber: A darf Z_A nie B zeigen - sonst könnte B es sich kopieren und sich fortan als A ausgeben!
  - wie vermeidet man “raubkopierte Zertifikate”?
  - in der digitalen Welt lassen sich Bitfolgen perfekt kopieren (im Unterschied zu “fälschungssicheren Ausweisen”)
- Z_A muss offenbar ein *Geheimnis* bleiben, das ausser der Autorität und A niemand kennt!
- Taugt ein solches Geheimnis als Zertifikat??
  - wie beweist man den Besitz eines Zertifikates, ohne es zu zeigen?

Geheime Zertifikate ?

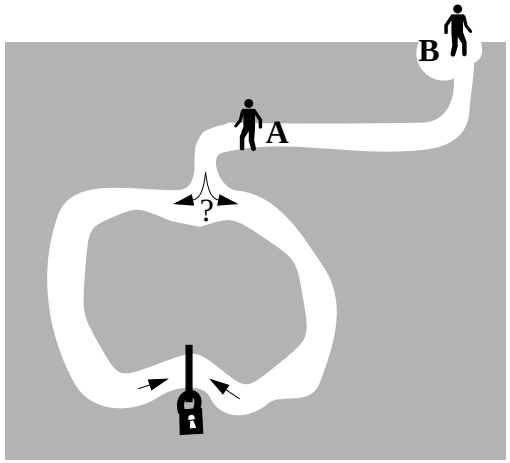


Geheime Zertifikate !

- Im Prinzip wissen wir schon, dass das geht: Der secret key sA eines asymmetrischen Verfahrens stellt ein solches Zertifikat dar
  - braucht von A nicht verraten zu werden
  - B kann dennoch überprüfen, ob A das Zertifikat hat (z.B. indem sich B von A etwas mit sA verschlüsseln lässt und anschliessend durch Anwenden von pA prüft; oder indem B ein $\{M\}_{pA}$ an A schickt und A dies mit sA entschlüsselt)

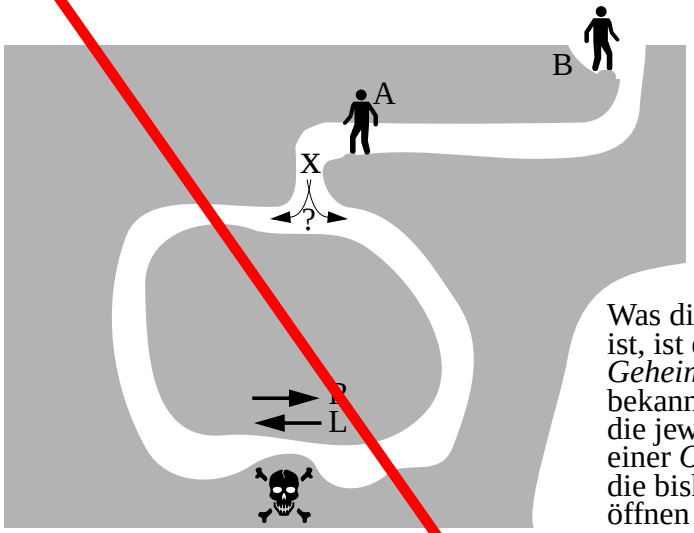
- Eine andere Realisierung geht mit “zero knowledge”
  - beweist Kenntnis eines Geheimnisses G, ohne geringste Information preiszugeben

- Ein “Höhlengleichnis” dazu (in Kurzform):



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

Ein Höhlengleichnis



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

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

- A sagt zu B: "Ich kenne das Geheimnis. Das beweise ich Dir, ohne das Geheimnis zu verraten!"

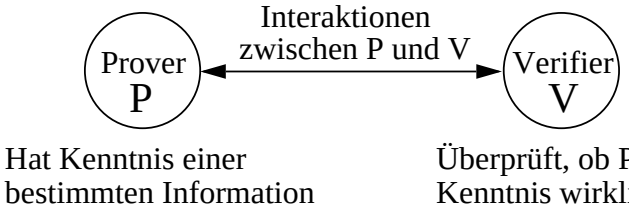
  - A begibt sich in die Höhle bis zur Engstelle; erst danach folgt B bis zur Stelle x (B sieht nicht, welche Richtung A dort eingeschlagen hat)
  - B ruft A *entweder*
    - "komm links heraus!" *oder*
    - "komm rechts heraus!" zu
  - A tut dies, indem A ggf. die Engstelle (in der richtigen Richtung) passiert
  - A und B verlassen zusammen die Höhle

- Nachdem A das ganze n Mal überlebt hat, glaubt B, dass A das Geheimnis (= Funktion der Orakelbox) kennt!
  - Die Irrtumswahrscheinlichkeit beträgt nur 2^{-n}

- B hat in diesem "interaktiven Beweis" das Geheimnis nicht erfahren! ==> "Zero knowledge proof"

Zero-Knowledge-Proofs

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



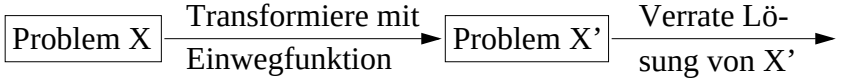
- P kann V *(praktisch) nicht betrügen:* Wenn P die Information nicht hat, sind seine Chancen, V zu überzeugen, verschwindend gering

- V *erfährt nichts* über die eigentliche Kenntnis von P

- V erfährt auch sonst nichts durch P, was V nicht auch alleine in Erfahrung bringen könnte

---

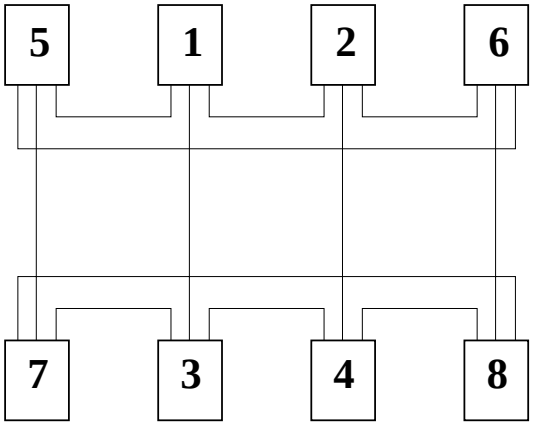
Idee:



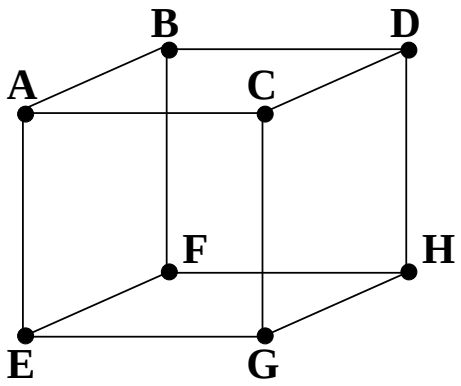
(Wobei die Lösung von X' die Lösung von X logisch impliziert, sie jedoch nicht effektiv-konstruktiv liefert)

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 Knoten-zuordnung gegebenen Isomorphismus ist allerdings “einfach”!

Zero-Knowledge mit Graphisomorphie

- P behauptet, einen Beweis zu haben, dass zwei gegebene Graphen G_1, G_2 isomorph sind, möchte den Beweis aber nicht verraten

- Folgendes Protokoll *überzeugt* V davon:

  - P erzeugt durch zufällige Permutation der Knoten einen Graphen H mit $H \sim G_1$ (und damit $H \sim G_2$). Für P ist dies einfach. V aber kann $H \sim G_1$ oder $H \sim G_2$ nicht einfacher entscheiden als $G_1 \sim G_2$

  - P sendet H an V

  - Entweder bittet V dann P
    - $H \sim G_1$ nachzuweisen, *oder*
    - $H \sim G_2$ nachzuweisen

  zufällig; bzw. von P nicht vorhersehbar

  - Da P den Graphen H konstruiert hat, kann P das gewünschte einfach tun (P hütet sich jedoch davor, auch noch die von V nicht gewünschte Alternative nachzuweisen - wieso?)

  - P und V wiederholen alles n Mal, wobei von P jedesmal ein anderer “Zeuge” H konstruiert wird (Beweissicherheit = $1-2^{-n}$)

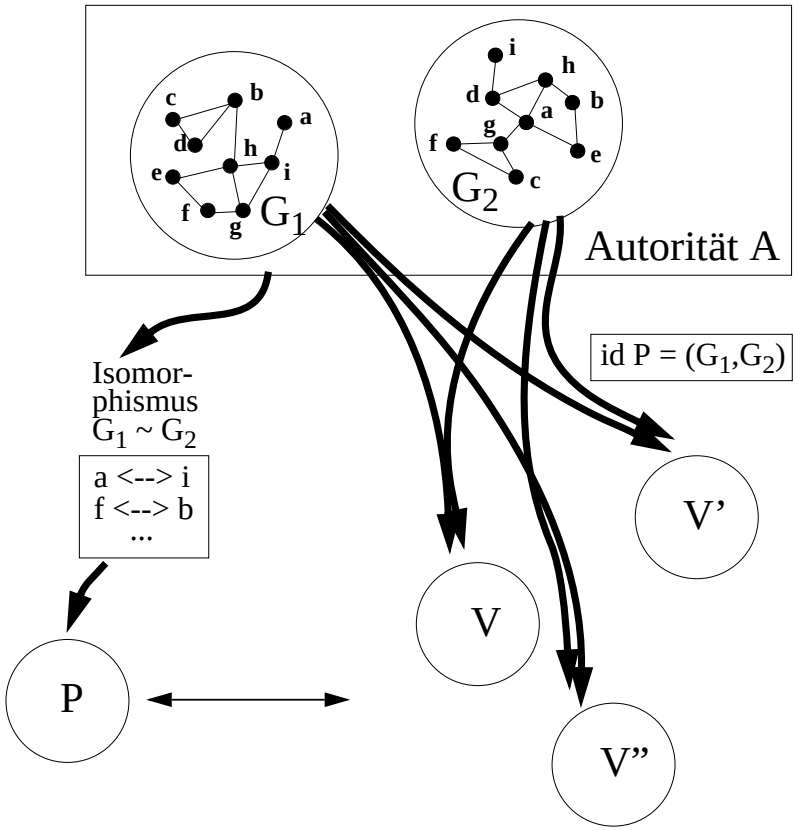
- Der Isomorphismus bleibt dabei ein Geheimnis von P!

Zero-Knowledge: Eigenschaften

- Falls P *keinen* Isomorphismus zwischen G_1 und G_2 kennt (also *lügt*), kann P keinen Graphen H konstruieren, der nachweislich isomorph zu beiden ist
  - *Verschiedene* H_1, H_2 zu finden mit $H_1 \sim G_1$ und $H_2 \sim G_2$ ist einfach; mit 50% Wahrscheinlichkeit wird P dann allerdings der Lüge überführt!
- V *lernt nichts* über die Isomorphie $G_1 \sim G_2$, *glaubt* aber schliesslich, dass P eine solche kennt
- Zur Minimierung der Interaktionen lassen sich die "Runden" *parallelisieren*: P sendet *mehrere* "isomorphe Zeugen" an V, und V sendet einen Bitvektor zurück, der die Einzelnachweise auswählt
- V kann einem Dritten W gegenüber nicht beweisen, dass P den Isomorphismus kennt: Selbst ein exaktes Protokoll der Kommunikationsvorgänge muss W nicht überzeugen: P und V könnten sich *verschworen* haben!
- Da V nichts gelernt hat, kann V sich anderen gegenüber auch nicht mit der Kenntnis schmücken, sich also *nicht für P ausgeben* (wenn die Kenntnis P identifiziert)

Es gilt: *Jeder mathematische Beweis kann in einen Zero-knowledge-Proof transformiert werden!* (Hier nur Beweisansatz: Jedes math. Theorem kann als Graph dargestellt werden, so dass Theorem gilt gdw. Graph Hamilton-Zykel besitzt; Zero-Knowledge-Beweis, dass Graph Hamilton-Zykel besitzt, wie folgt: für isomorphen Graphen H entweder Isomorphie zu G oder aber Hamilton-Zykel offenlegen)

Identitätsbeweise mit Graphisomorphie?



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

Zero-Knowledge-Identitätsbeweise

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

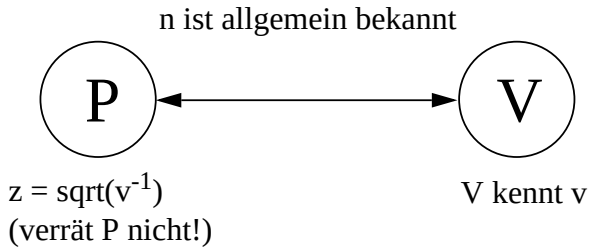
- Bemerkung: Die k-te Wurzel in einem Restklassenring mod n zu bestimmen ist sehr schwierig, wenn man nicht die Zerlegung n = pq kennt (“trap door”); die Berechnung der Zerlegung pq aus n ist genauso schwer

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

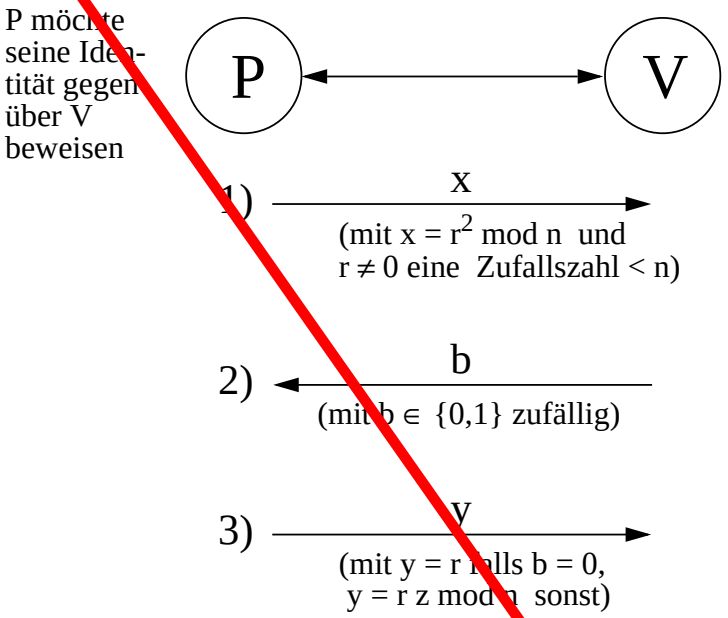
- Jeder Prozess P bekommt von X eine Zahl v sowie ein Zertifikat z. Die Zahlen n und v sind allgemein bekannt

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

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



Das Protokoll von Fiat und Shamir



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

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

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

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

Die Situation des Angreifers

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

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

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

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

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

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

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

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

Ein Beispiel

- Sei n = 35 (mit Primfaktoren 5 und 7)
  ist unrealistisch klein!

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

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

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

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

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

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

2) V sendet b = 1 zurück.

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

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

Geheimnisverrat?

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

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

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

---

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

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

---

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

Identitätsnachweise

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

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

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

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

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

Münzwurf über Telefon

Prinzip: *„Münzen in der Schatzkiste“*

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

Probleme:

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

Münzwurf über Telefon

Gegeben: kommutatives Public-Key-System

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

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

Komplexere Protokolle

Denkübungen zum Protokoll „Münzwurf über Telefon“

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

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

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

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

Komplexere Protokolle

Denkübungen zum Protokoll „Münzwurf über Telefon“ (Fortsetzung)

- **Voraussetzung bisher: Verfügbarkeit eines kommutativen Public-Key-Systems**
  - harte Forderung: Existenz solcher Systeme? Rechenaufwand?

- **Alternative 1**
  - Verwende ein nicht-kommutatives System!
  - Löse die dadurch entstehenden Probleme auf anderem Wege!
  - Beitrag der Kommutativität hinsichtlich Geheimhaltung und Festlegung (Commitment) auf einen bestimmten Nachrichteninhalt?

- **Alternative 2**
  - Verwende statt Verschlüsselung eine **Einwegfunktion**
  - Was ist zu beachten, um Ausgang des Münzwurfs nachträglich belegen zu können?

- **Einer der Mitspieler erfährt den Ausgang des Spiels zwangsläufig als erster:**
  - Was tun, wenn er bei einer Niederlage das Spiel abbricht, ohne es aufzulösen?
  - *„Wer aufhört verliert!“* – Ist das eine Lösung? (juristisch: Unschuldsvermutung!)

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

  erstes grosses Client-Server-Campusnetz, ca. 25000 Benutzer, 1200 Computer

- Wird u.a. bei DCE verwendet

  Distributed Computing Environment

- Basiert auf Needham-Schroeder-Protokoll mit symmetrischen Schlüsseln (i.a. DES)

- Public domain; es gibt auch kommerzielle Varianten

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

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

B. Clifford Neuman and Theodore Ts'o: *Kerberos: An Authentication Service for Computer Networks.* IEEE Communications Magazine, Volume 32, Number 9, pp. 33-38, September 1994. Im Internet: http://nii.isi.edu/publications/kerberos-neuman-tso.htm

RFC 1510: *The Kerberos Network Authentication Service (V5).* Im Internet: ftp://ftp.isi.edu/in-notes/rfc1510.txt oder http://ds0.internic.net/rfc/rfc1510.txt

Kerberos-"Philosophie"

- Offenes Campusnetz --> Nachrichten prinzipiell unsicher

- Kommunikation nur verschlüsselt und mit authentifizierten Partnern
  - Kenntnis des Sitzungsschlüssels als Authentitätsbeweis

- Passwörter niemals im Klartext übertragen
  - auch keine Passwortspeicherung

  "Kerberos-Server"

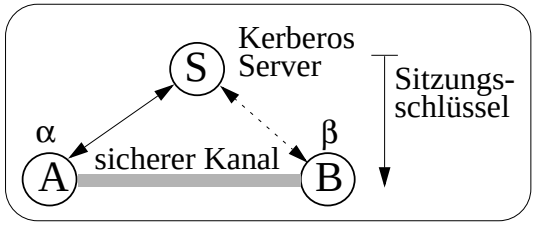
- Alle Benutzer, Clients und Server sind bei zentraler Instanz (Key Distribution Center: "KDC") akkreditiert
  - vereinbaren mit dem KDC auch ihren Geheimschlüssel ("master key")
  - ohne Akkreditierung keine Server-Berechtigungsscheine ("Tickets")
  - ohne Tickets kein Service
  - Ticket personengebunden, nur in Verbindung mit Authentitätsnachweis gültig

- Gültigkeit von Tickets / Sitzungsschlüsseln zeitlich befristet

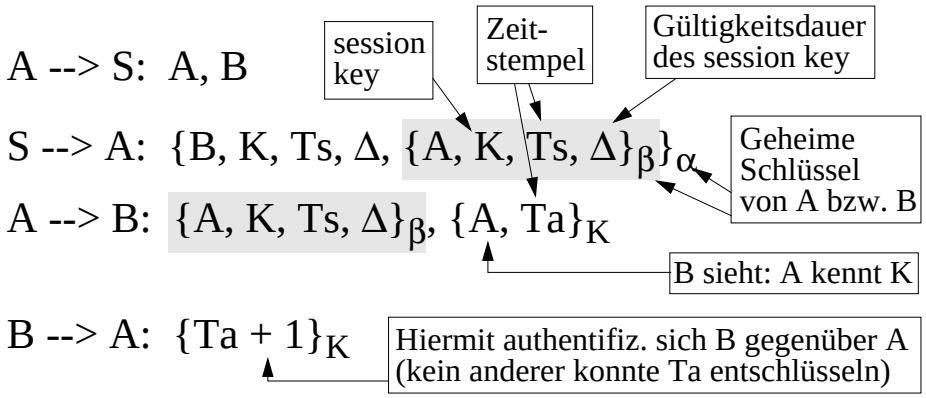
- Drei Sicherheitsstufen
  (1) Authentifizierung nur bei Einrichtung eines Kommunikationskanals
  (2) Authentifizierung bei jeder Nachricht zwischen A und B
  (3) zusätzlich Verschlüsselung der Nachrichten

Kerberos-Anwendungsbeispiel: Einrichtung eines sicheren Kanals

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

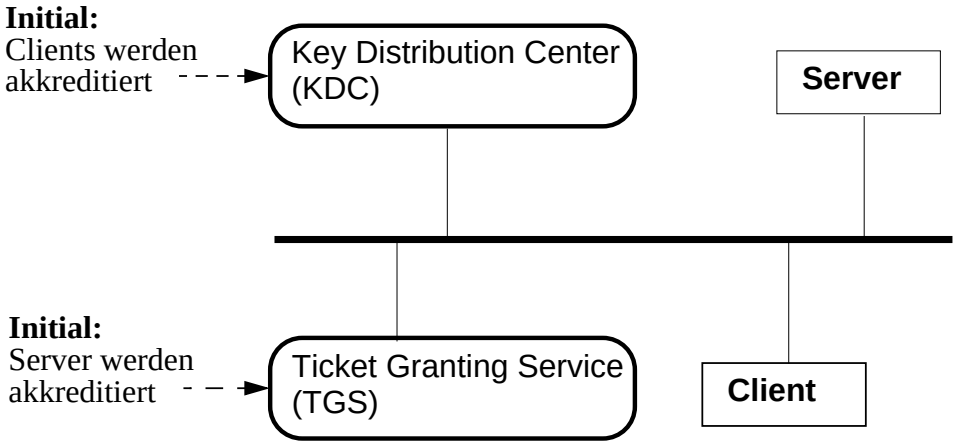


Hier: Version 4 (MIT-Version eingefroren Dez. ‘92); spätere Versionen im Prinzip nur leicht unterschiedlich



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

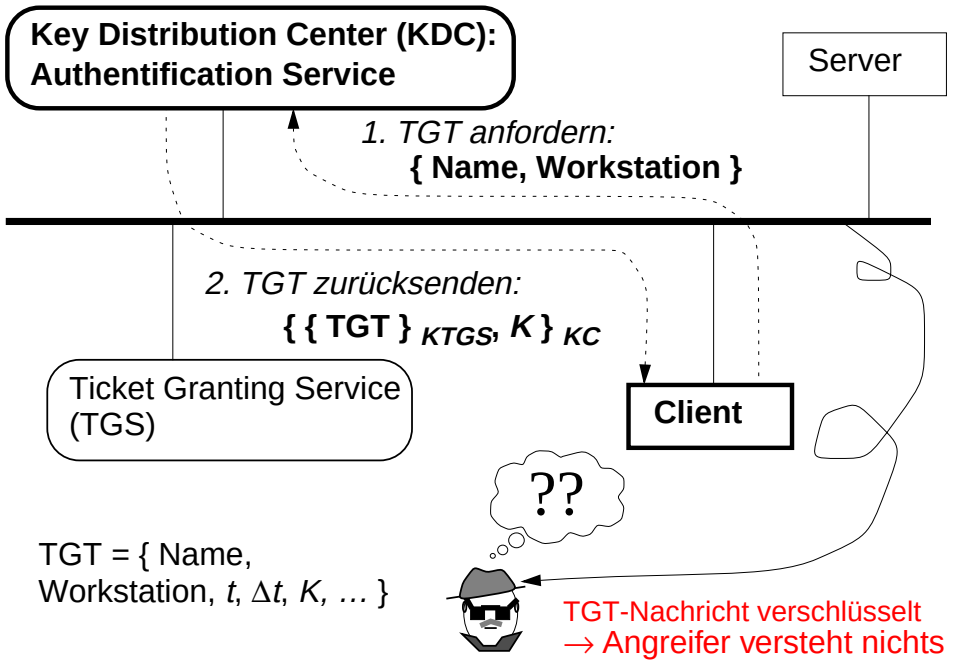
Kerberos: Akkreditierung



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

Kerberos: TGT-Anforderung

- Client erwirbt zunächst ein Ticket Granting Ticket (TGT)



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

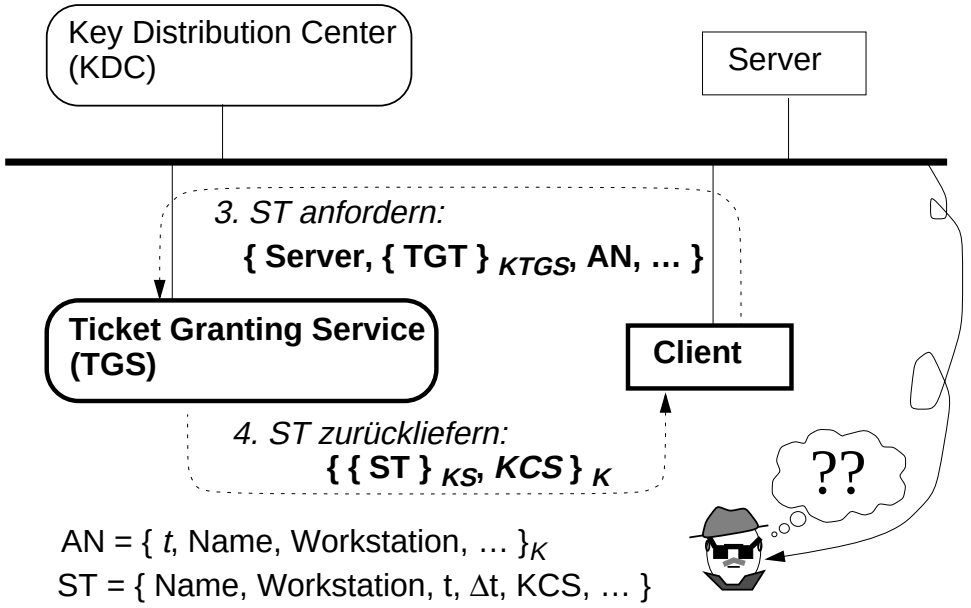
--> KDC-Nachricht ist authentisch: Nur KDC kennt noch Schlüssel KC !

--> Nur der echte Client kann TGT mittels KC nutzbar machen

--> Passwort verlässt Workstation nicht und wird sofort wieder gelöscht

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

Kerberos: Service Ticket erwerben

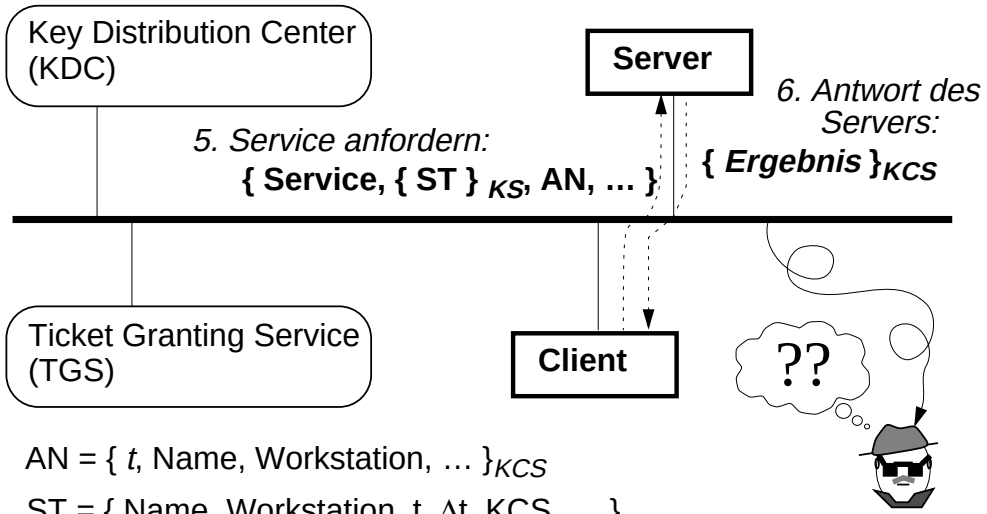


- Client: erstellt Authentizitätsnachweis AN = { t, Name, Workstation, … } $_{K}$
- Client an TGS: sendet { Server, { TGT } $_{KTGS}$, AN, … } als Request
- TGS: entschlüsselt TGT mit Schlüssel KTGS, erhält damit K;
  entschlüsselt AN mit K, vergleicht Inhalt mit TGT;
  erstellt Service Ticket ST = {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

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

--> ST höchstens für Zeitspanne Δt gültig, und nur an bezeichneter Workstation

Kerberos: Nutzung des Service



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

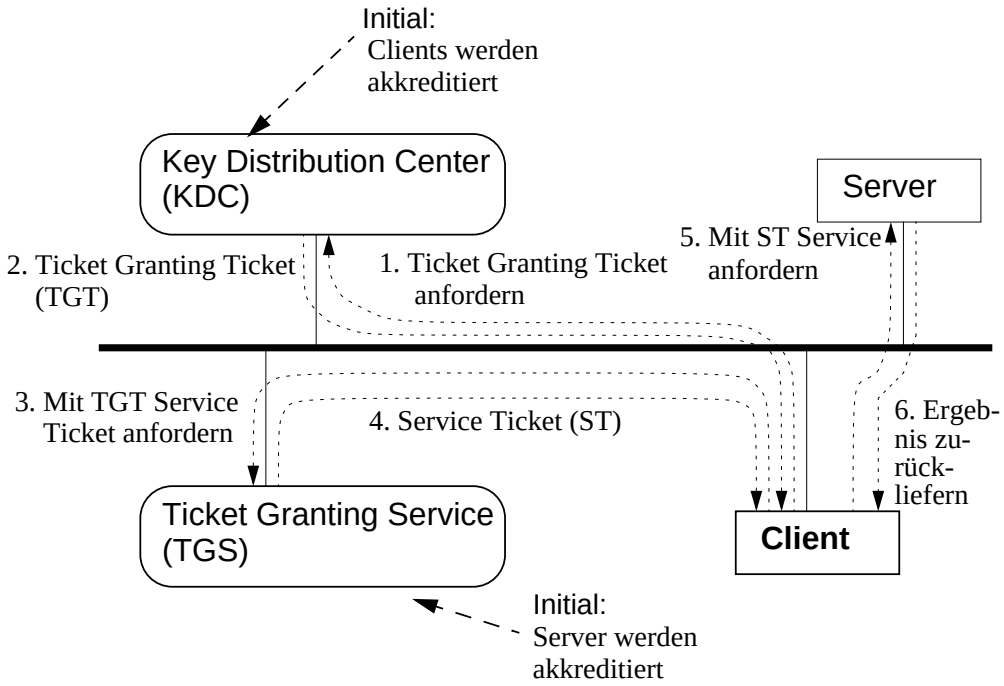
ST = { Name, Workstation, t, Δt, KCS, … }

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

--> Folgedialoge zwischen Client und Server mittels KCS verschlüsselbar

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

Kerberos: Protokollübersicht



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

Kerberos - weitere Aspekte

- Nachrichten enthalten noch weitere (technische) Angaben
  - z.B. Versionsnummer, Nachrichtentyp, Prüfsumme, Netzwerkadresse...
- Es gibt dezentrale Zuständigkeitsbereiche (“realms”)
  - lok. KDC vermittelt Zugangsticket zu KDC eines fremden Bereichs
- Kerberos-Software enthält u.a.:
  - Library mit Routinen, um Authentifizierungsanforderungen erzeugen und lesen zu können, Nachrichten zu authentifizieren und zu verschlüsseln
  - Datenbank und Verwaltungsroutinen für registrierte Nutzer (Geheimschlüssel, Gültigkeitsdauer, Verwaltungsdaten...)
  - Software zur Replikation der Datenbank (Verteilung ist wichtig, da bei Ausfall des KDC fast nichts mehr im ganzen Netz geht!)
  - “Read-only”-Netzwerkzugang
- Version 5 (inkompatibel zu Version 4!): mehr Funktionalität und allgemeiner verwendbar, z.B.:
  - Datenformate mit ASN.1 - Basic Encoding Rules
  - Verbesserung einiger Sicherheitskonzepte; Alternativen zu DES möglich
- Weiterentwicklungen?
  - z.B. asymm. Schlüssel, Einbindung von Chipkarten, verteilte Datenbank...
- Kerberos ist weit verbreitet (“Quasi-Standard”)
  - z.B. um verteilte Dateiserver (NFS, AFS) zu sichern oder modifizierte Versionen von telnet, rlogin, rcp, rsh, ftp etc. zu ermöglichen
  - kommerzielle Varianten z.T. nicht kompatibel zueinander
  - war nicht ohne weiteres aus den USA exportierbar (verwendet DES)

Kerberos - Sicherheitsaspekte

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

Schlüsselgenerierung

- Der Schlüsselgenerierungsalgorithmus von Kerberos (z.B. für TGT oder session keys) funktioniert so:

das ist gelogen!

bitweise xor

```
p = getpid() ^ gethostid ;
gettimeofday(&time, (struct timezone *) 0) ;
/* randomize start */
srandom(time.tv_usec ^ time.tv_sec ^ p ^ n++)
```

Initialisierung für eine (deterministischen) Folge von "random"-Werten

- Startwert des Zufallszahlengenerators ("seed")
- gettimeofday liefert auf "time" zwei Werte zurück:
  - tv_sec: Sekunden seit dem 01.01.1970
  - tv_usec: Mikrosekunden...

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

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

- Angeblich soll nun der Algorithmus "verbessert" sein...

- Und die Moral der Geschichte?