Authentifikation

Was ist Authentifikation?

Authentifikation ist der Beweis, dass die behauptete Identität eines Individuums oder einer Maschine mit seiner/ihrer wahren Identität übereinstimmt.

Wozu benötigt man Authentifikation?

Authentifikation ist überall dort notwendig, wo bestimmte Dienste oder Daten nur bestimmten Personen oder Gruppen zur Verfügung gestellt werden sollen.

Arten der Authentifikation

Authentifikation basiert auf

- Wissen (Passwort),
- Besitz/Token (Magnetkarte, SmartCard) oder
- einer speziellen Eigenschaft (Biometrie) (Fingerabdruck, Netzhaut, Unterschrift).

Kombinationen der verschiedenen Authentifikationsarten sind möglich, z.B. Magnetkarte mit PIN oder Sprechen eines bestimmten Satzes.

Authentifikation durch Wissen

Vorteile

- einfach zu implementieren,
- einfach im Transport,
- leichter Wechsel bei Kompromittierung.

Nachteile

- kann erraten werden (Brute Force Attack),
- möglicherweise schwierig zu merken,
- muss irgendwo gespeichert werden.

Authentifikation durch Besitz

Vorteile

- keine Brute Force Attack (Erraten) möglich,
- es muss nichts auswendig gelernt werden.

Nachteile

- Brute Force Attack (Ausrauben) möglich,
- Token kann zu Hause vergessen werden,
- Token kann verloren werden,
- spezielle Hardware erforderlich.

Authentifikation durch Biometrie

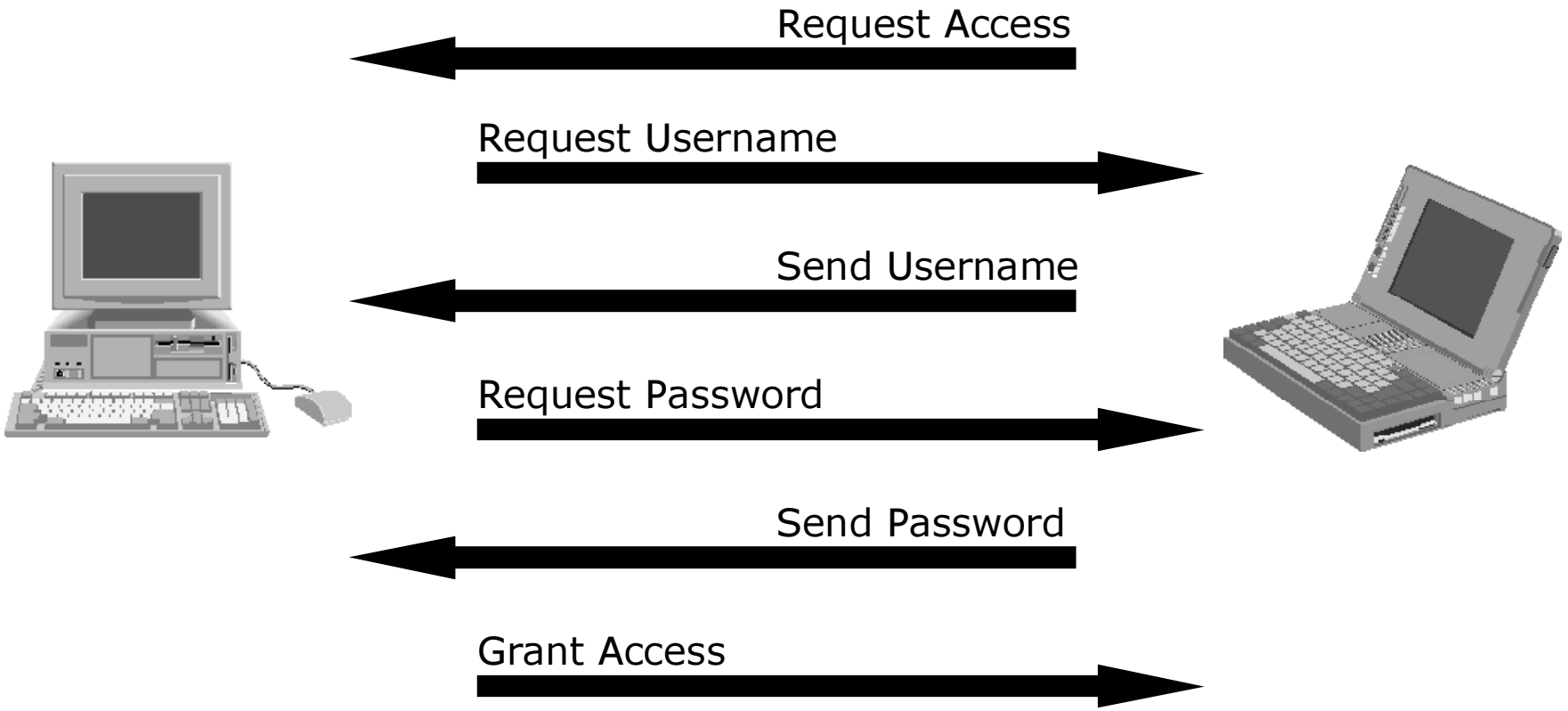
Vorteile

- keine Brute Force Attack (Erraten) möglich,
- es muss nichts auswendig gelernt werden,
- kann (normalerweise) nicht gestohlen werden.

Nachteile

- möglicherweise gesundheitsgefährdend,
- in den meisten Fällen viel zu teuer.

Simples Passwortprotokoll



Beispiel (Authentifikation in HTTP)

```
Client: GET /secret/42.txt HTTP/1.0

Server: HTTP/1.0 401 Unauthorized
        WWW-Authenticate: Basic realm = "Access for /secret/"

Client: GET /secret/42.txt HTTP/1.0
        Authorization: QWxhZGRpbjpvcGVuIHNlc2FtZQ==

Server: HTTP/1.0 200 OK
        Der Sinn des Lebens ist...
```

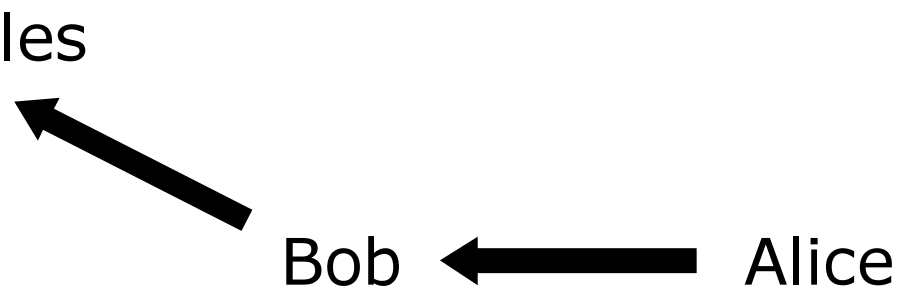
Ausschnitt aus RFC 1945 (HTTP 1.0)

To receive authorization, the client sends the user-ID and password, separated by a single colon (":") character, within a base64 encoded string in the credentials.

Sicherheitsprobleme

Eve kann das Passwort hören.

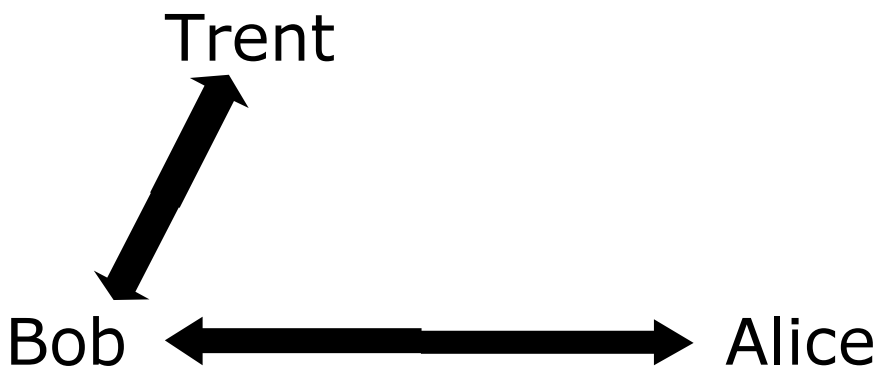
Benutzt Alice mehrmals dasselbe Passwort, kann Bob sich gegenüber Charles als Alice ausgeben:



Kommunikationsstruktur

Eigenständiges Protokoll...

...oder mit trusted third party.



Konventionelle Authentifikation ...

... durch zeitlich gleichbleibende Passwörter
(„weak authentication”)

Speichern der Passwörter

- im Klartext
- „verschlüsselt”

Sicherheitsrisiken

Replay-Attacken, exhaustive Passwortsuche,
(Offline-)Dictionary-Attacken

Sicherheitstechniken

aufwendige Algorithmen, Dictionary Check, Salting,
Passphrasen, Sperren von Accounts

Einmal-Passwörter

Jedes Passwort darf nur einmal verwendet werden.
(„towards strong authentication”)

Realisierung durch

- eine gemeinsame Liste von Einmal-Passwörtern,
- sequentiell aktualisierte Passwörter,
- Hash-Funktionen (z.B. Lamport-Schema).

Lamport-Schema

Alice will sich bei Bob anmelden.

Gegeben sind eine (One-Way-)Hash-Funktion H und eine Konstante t, nach der das Protokoll neu gestartet werden muss (Obergrenze).

Initialisierung

- Alice denkt sich ein Geheimnis S aus,
- sie schickt den Startwert $S_0 = H^t(S)$ an Bob,
- Bob setzt seinen Alice-Counter $Cnt_{Alice} := 1$.

Lamport-Schema

Der i-te Authentifikationsvorgang ($1 \leq i \leq t$):

Alice schickt Bob eine Nachricht

$$M_i := (\text{„Alice”}, i, S_i = H^{t-i}(w)).$$

Bob überprüft, ob $i = Cnt_{Alice}$ (sein interner Alice-Counter) und ob $H(S_i) = S_{i-1}$ ist. Wenn beides stimmt, glaubt er Alice, dass sie Alice ist, erhöht Cnt_{Alice} um eins und speichert S_i für die nächste Authentifikation.

Challenge-Response-Techniken

Challenge/Response-Identifikation („strong authentication")

Ziel: Beweis der Kenntnis eines Geheimnisses, ohne dieses preiszugeben.

- Zeitlich veränderliche Herausforderung (Challenge) an den Claimant bzw. Alice (üblicherweise eine Zufallszahl),
- die Antwort (Response) ist abhängig vom Geheimnis und der Herausforderung.

C/R mit symmetrischen Verfahren

- **Einseitige Authentifikation mit Zeitstempel**

  A -> B: $Encrypt_{Key}$(Timestamp, „Bob”).

- **Einseitige Authentifikation mit Zufallszahlen**

  B -> A: Random,
  A -> B: $Encrypt_{Key}$(Random, „Bob”).

- **Gegenseitige Authentifikation**

  B -> A: $Random_B$,
  A -> B: $Encrypt_{Key}$($Random_A$, $Random_B$, „Bob”),
  B -> A: $Encrypt_{Key}$($Random_B$, $Random_A$).

C/R mit Einwegfunktionen

Ersetzung der symmetrischen Verschlüsselung durch eine schlüsselabhängige Einwegfunktion.

Gegenseitige Authentifikation

B -> A: $Rnd_{B,}$
A -> B: Rnd_A, $H_{Key}(Rnd_A, Rnd_B, „Bob”)$,
B -> A: $H_{Key}(Rnd_B, Rnd_A, „Alice”)$.

C/R mit Public-Key-Verfahren

Authentifikation durch

- Public-Key-Entschlüsselung
  (z.B. Needham-Schroeder)

  A -> B: $P_B(Random_A, A)$,

  B -> A: $P_A(Random_A, Random_B)$,

  A -> B: $Random_B$.

- digitale Signatur
  (verschiedene Signatur-Verfahren)

Zero-Knowledge-Verfahren

Motivation

Immer, wenn Alice ihr Passwort an Bob sendet, um sich anzumelden, kann Bob sich gegenüber einem dritten als Alice ausgeben.

Zentrale Eigenschaft eines ZK-Protokolls

Alice beweist Bob ihre Identität, ohne dabei irgend eine weitere Information über das Geheimnis (Passwort) preiszugeben.

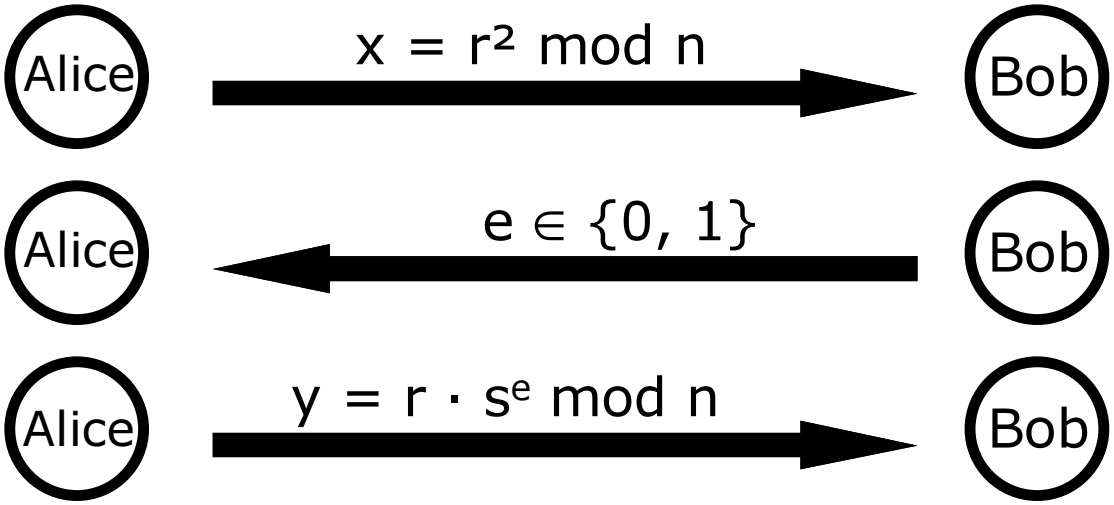
ZK-Protokoll: Fiat-Shamir

Voraussetzungen

- Ein Trustcenter wählt ein Modul $n = pq$ mit geheimen Primfaktoren p und q,
- Alice wählt eine geheime Zahl s und registriert $v = s^2 \pmod{n}$ als ihren öffentlichen Schlüssel.

ZK-Protokoll: Fiat-Shamir

Es folgen t Protokollrunden:



Bob akzeptiert die Protokollrunde, wenn $y^2 = x \cdot v^e$ ist.

r: von Alice ausgewählte Zufallszahl e: von Bob gewähltes Zufallsbit

ZK-Protokoll: Fiat-Shamir

Die Sicherheit hängt an der Zufälligkeit von e.

Fall I: Der Angreifer weiß, dass das nächste e = 0 ist.

Überlisten leicht möglich durch freies Wählen von y statt r, so dass $x = y^2$ ist.

Fall II: Der Angreifer weiß, dass das nächste e = 1 ist.

Überlisten leicht möglich durch freies Wählen von y statt r, so dass $x = y^2 / v$ ist.

Der Angreifer muss sich für I oder II entscheiden, bevor er e kennt!

ZK-Protokoll: Fiat-Shamir

Wann ist Bob von Alices Identität überzeugt?

Die Wahrscheinlichkeit, dass ein Angreifer t Runden Fiat-Shamir übersteht, ist offenbar $p_{FS}(t) = 2^{-t}$.

Das Handbook of Applied Cryptography empfiehlt, je nach Einsatzgebiet, Rundenzahlen t=20 oder t=40.

$p_{FS}(20) = 9{,}54 \cdot 10\text{-}7.$ $p_{FS}(40) = 9{,}09 \cdot 10^{-13}.$

Encrypted Key Exchange

Problem

- sichere Secrets (lange Zahlen) sind schwer zu merken,
- gut zu merkende Passwörter sind leicht zu erraten

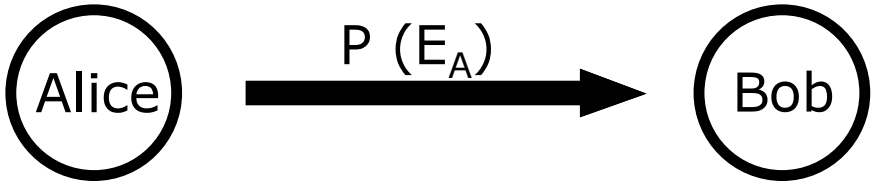
Lösung: Encrypted Key Exchange

- sichere Authentifikation, auch bei schlecht gewählten Passwörtern, durch
- verschlüsselte Übertragung zufälliger Session Keys

Encrypted Key Exchange

Vereinfachter Protokollablauf (RSA-EKE)

1. Alice erzeugt ein zufälliges Public-Key-Paar (E_A,D_A), verschlüsselt es mit ihrem Passwort P und schickt es Bob.

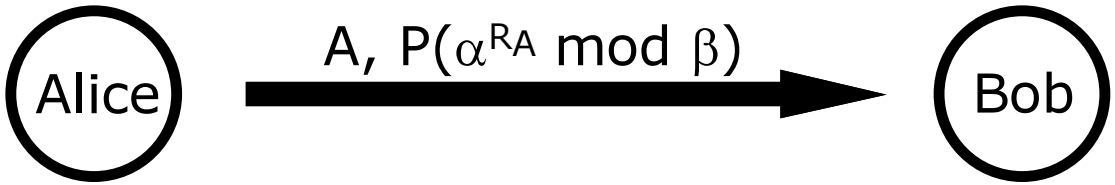


2. Bob, der P kennt, weiß nun, dass er es mit Alice zu tun hat, erzeugt eine Zufallszahl R (Session Key) und schickt sie ihr verschlüsselt zurück.

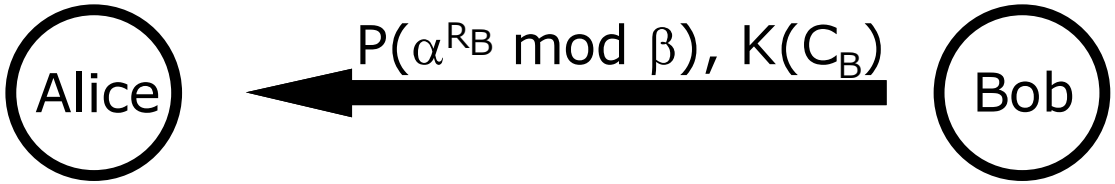
Encrypted Key Exchange

EKE mit Diffie-Hellman (DH-EKE)

1. Alice erzeugt eine Zufallszahl R_A.

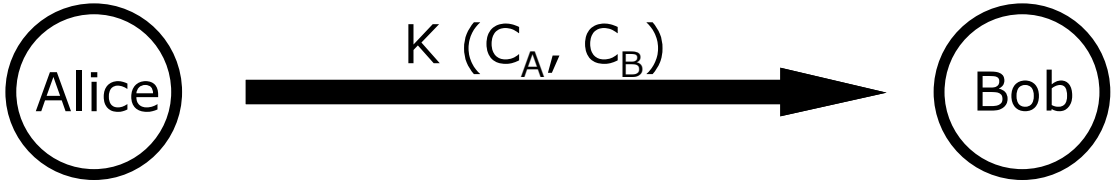


2. Bob macht es ihr mit seiner Zahl R_B nach und berechnet den Session Key K.

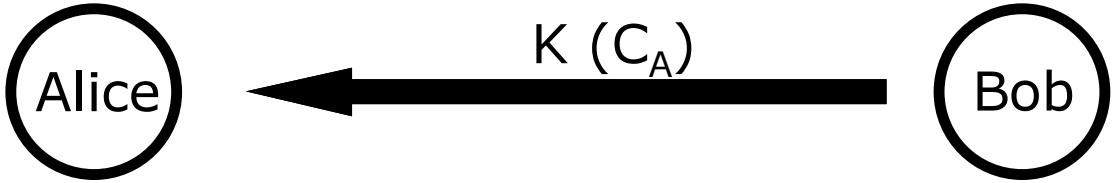


Encrypted Key Exchange

3. Alice berechnet K, löst die Aufgabe C_B und stellt nun Bob ihrerseits ein Random-Challenge C_A.



4. Bob löst seine Aufgabe und sendet das Ergebnis an Alice.



5. Beide sind glücklich.

Augmented EKE

Sicherheit gegen Kompromittierung der Passwortdatei

- Speichern eines Hash-Werts an Stelle des Passworts,
- Protokoll wie Standard-EKE, nur mit H(P) statt P,
- Alice beweist Bob ihre Kenntnis von P durch Anwendung einer digitalen Signatur oder kommutative Hash-Funktionen.

Augmented EKE

Protokollablauf

A -> B: A, $\{\alpha^{R_A} \bmod \beta\}H(P)$ mit Zufallszahl R_A

B -> A: $\{\alpha^{R_B} \bmod \beta\}H(P)$, $\{Challenge_B\}\ \alpha^{R_A R_B}$

A -> B: $\{Challenge_A, Challenge_B\}\ \alpha^{R_A R_B}$

B -> A: $\{Challenge_A\}\ \alpha^{R_A R_B}$

A -> B: $\{\ \{\alpha^{R_A R_B} \bmod \beta\}S(P)\ \}\ \alpha^{R_A R_B}$

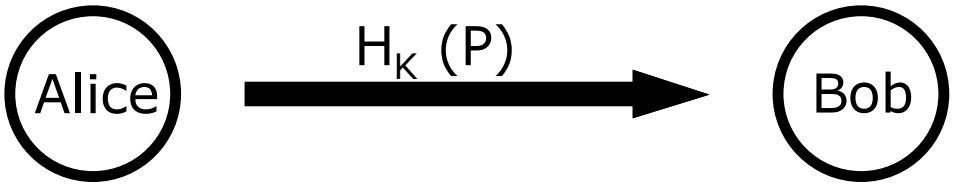
Protokoll ist erfolgreich, wenn $\{\ \{K\}S(P)\ \}H(P) = K$
mit $K = \alpha^{R_A R_B} \pmod \beta$.

Augmented EKE

A-EKE mit kommutativen Hash-Funktionen

Sei $H(P) = H_0(P)$ Element einer Menge kommutativer Einweg-Hashfunktionen.

A und B erzeugen den Session Key K, aus dem sie die kommutative Funktion H_K ableiten.



Bob überprüft jetzt bloß, ob $H_K(H_0(P)) = H_0(H_K(P))$.

Augmented EKE

A-EKE mit digitaler Signatur

Bob ist von Alices wahrer Identität überzeugt, wenn
{ {K}S(P) }H(P) = K ist.

Einfache Lösung:

S(P) ist Alices Secret Key, mit dem sie K signiert. Bob überprüft dann mit dem Public Key H(P) die Authenti-zität.

Sicherheit von EKE

Sarvar Patel:
"Number Theoretic Attacks On Secure Password Schemes"
(1997 IEEE Symposium on Security and Privacy)

Es gibt Attacken gegen alle drei im Originalpaper vorgestellten EKE-Varianten:

RSA-EKE, DH-EKE, ElGamal-EKE.

Die Attacken können nur mit hohem Aufwand (DH- und ElGamal-EKE) bzw. gar nicht (RSA-EKE) abgewehrt werden.

Secure Remote Password Protocol

- vorgestellt 1998 von Thomas Wu auf dem 1998 Internet Society Network and Distributed System Security Symposium,
- Schlüsselaustausch ohne Chiffrierung,
- basiert auf AKE (Asymmetric Key Exchange),
- „klassisches" Zero-Knowledge-Protokoll.

Secure Remote Password Protocol

Protokollablauf

	Carol		Steve
1.		$\xrightarrow{C}$	(lookup s, v)
2.	$x = H(s, P)$	$\xleftarrow{s}$	
3.	$A = g^a$	$\xrightarrow{A}$	
4.		$\xleftarrow{B, u}$	$B = v + g^b$
5.	$S = (B - g^x)^{a+ux}$		$S = (Av^u)^b$
6.	$K = H(S)$		$K = H(S)$
7.	$M_1 = H(A, B, K)$	$\xrightarrow{M_1}$	(verify M_1)
8.	(verify M_2)	$\xleftarrow{M_2}$	$M_2 = H(A, M_1, K)$

Verifier
$v = g^x$
Salt
Random
Random
Random
$= g^{(ab+bux)}$

Kerberos

Kerberos: Authentifikation in offenen Netzen

- entwickelt von Miller und Neuman im Project Athena des MIT,
- transparenter Authentifikationsmechanismus über zentralen Key Server,
- sicher gegen Eavesdropping,
- Authentifikation des Benutzers für eine Menge von Diensten durch einen einzigen Login-Vorgang.

Kerberos

Grundlagen

- Authentifikation basiert auf Needham-Schroeder,
- Timestamps gegen Replay-Attacken,
- Verschlüsselung mittels DES.

Message-Levels

- **Basic**
  Authentifikation nur bei Aufbau der Verbindung
- **Safe Message**
  Authentifikation beim Versenden jeder Nachricht
- **Private Message**
  Authentifikation und Verschlüsseln jeder Nachricht

Kerberos

Die Elemente der Kerberos-Architektur

- **Administration Server**
  Lese- und Schreibzugriff auf die Kerberos-Datenbank
- **Authentication Server („Kerberos”)**
  Nur-Lese-Zugriff auf die Datenbank, Authentifikation der Benutzer/Dienste, Erzeugen der Session Keys
- **Name Server („Hesiod”)**
  Kein Datenbankzugriff, Verzeichnisdienst
- **Clients**
  Login, Passwort ändern, Anzeigen/Löschen von Tickets

Kerberos

Teilnehmernamen in Kerberos

(werden für Benutzer und Server vergeben)

Schema

primary_name.instance@realm

Beispiele

jis@LCS.MIT.EDU
rlogin.priam@ATHENA.MIT.EDU

Kerberos

Zwei Arten von Credentials

- Ticket: für einen einzigen Client und einen einzigen Server, kann mehrmals verwendet werden

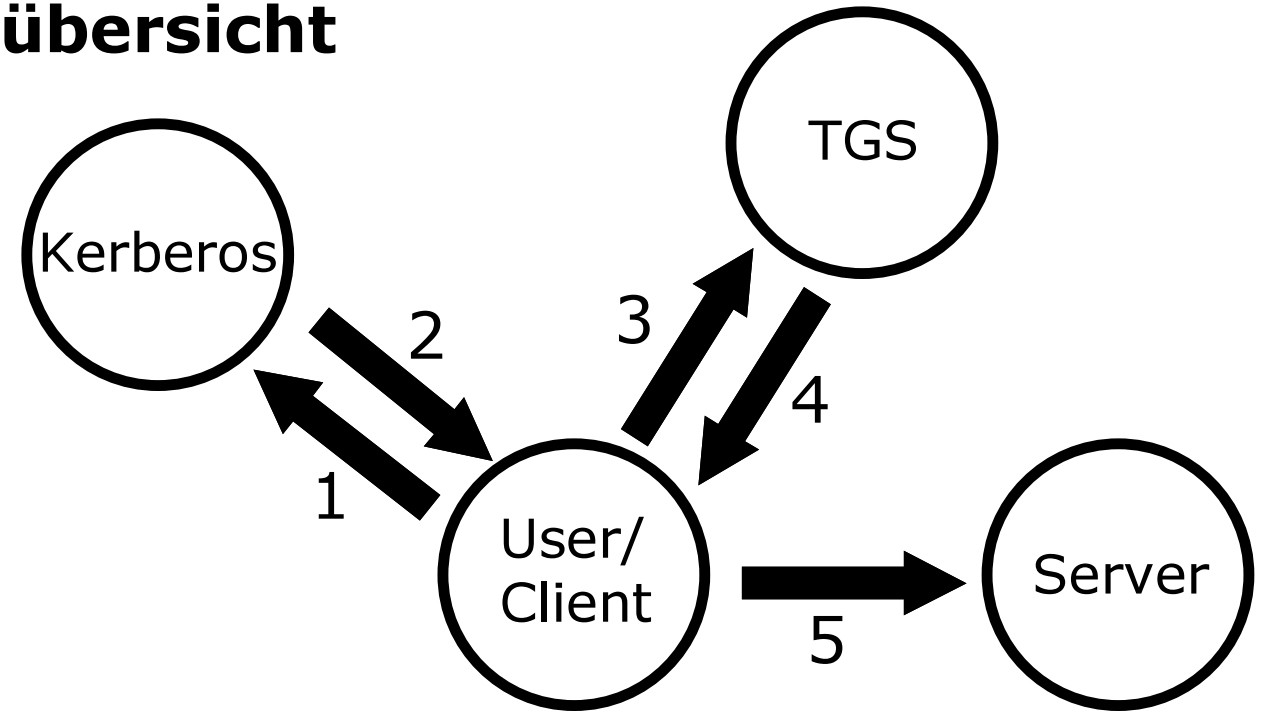
  $\{ c, s, Addr_c, Timestamp, Lifetime, K_{s,c} \} K_s$

- Authenticator: wird vom Client selbst erzeugt, kann nur einmal verwendet werden

  $\{ c, Addr_c, Timestamp \} K_{s,c}$

Kerberos

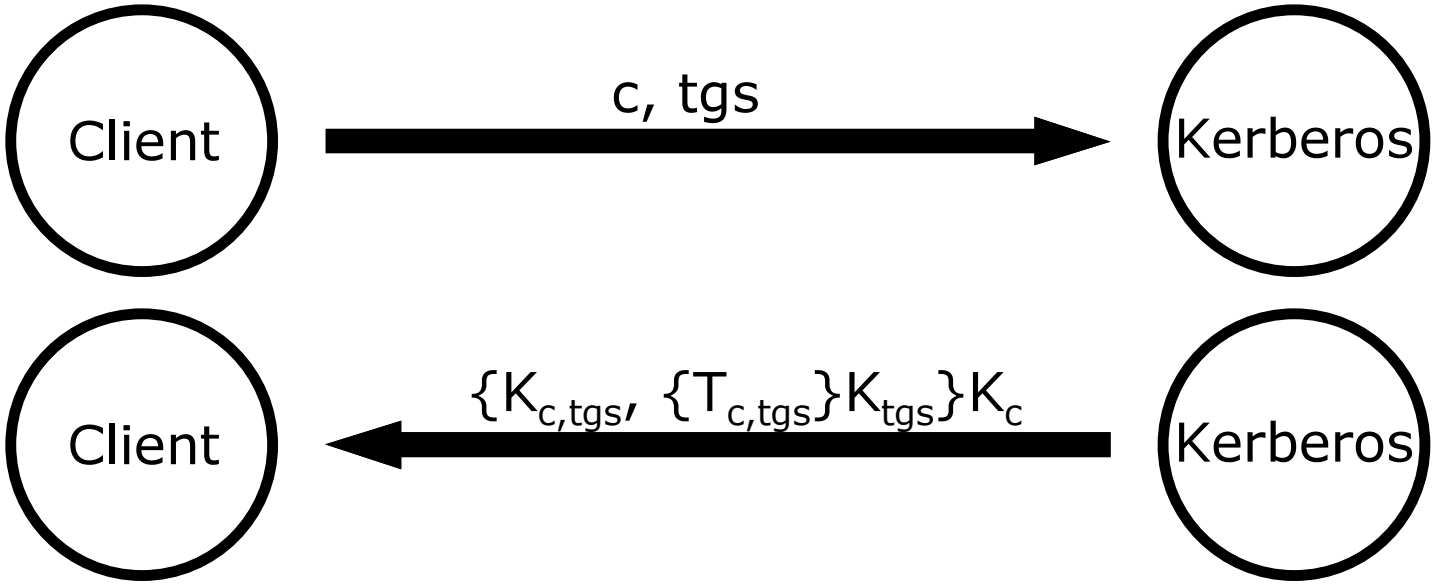
Protokollübersicht



1. TGS-Ticket anfordern
2. TGS-Ticket
3. Server-Ticket anfordern
4. Server-Ticket
5. Service anfordern

Kerberos

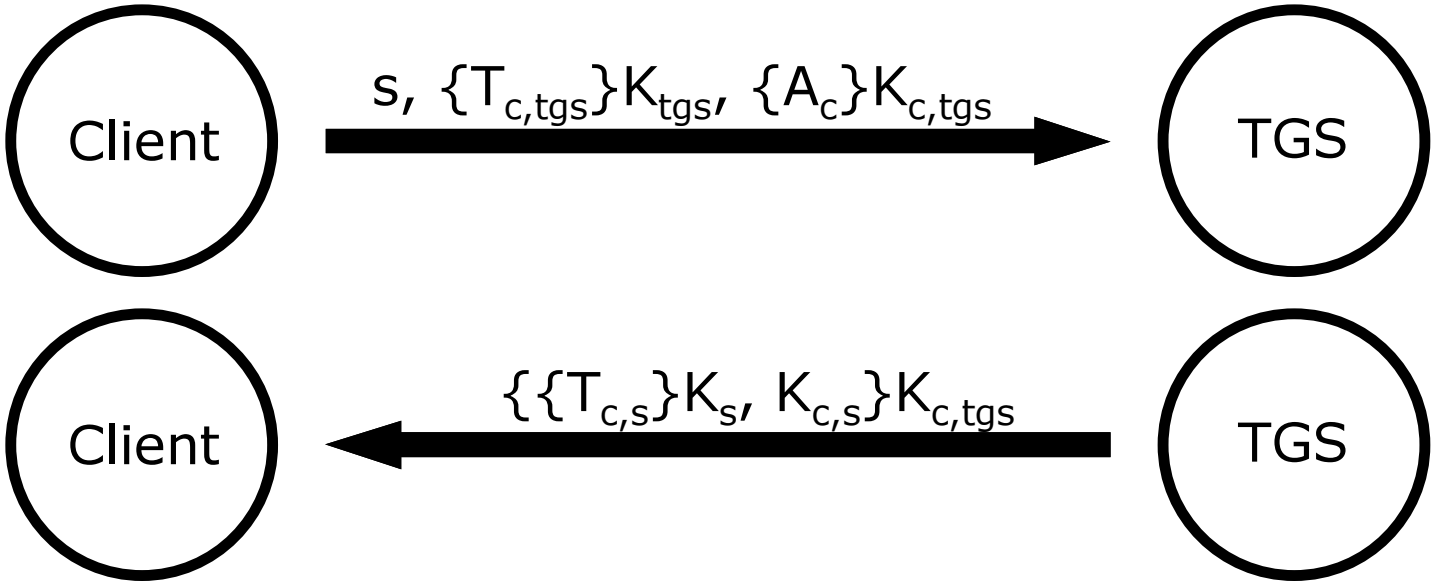
Anfordern des initialen (TGS-)Tickets / Login



c: Client tgs: Ticket Granting Server K_x: x's Private Key
$K_{x,y}$: Session Key für x und y $T_{x,y}$: Ticket für x, y zu benutzen

Kerberos

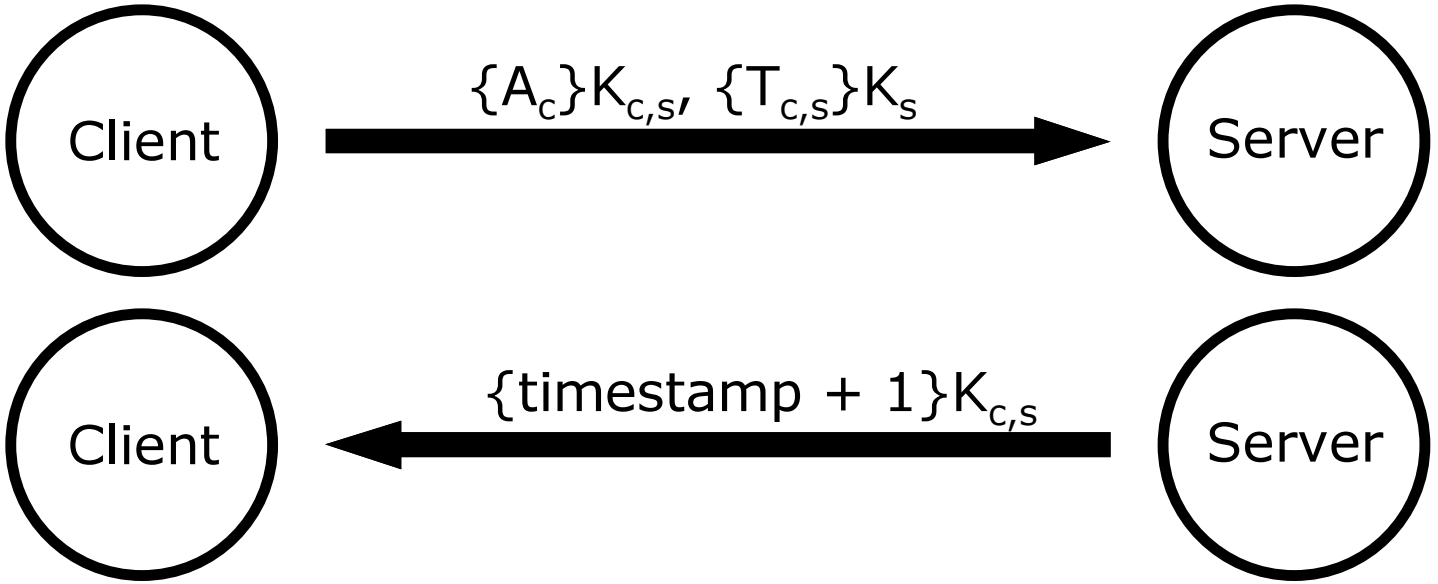
Anfordern eines Server-Tickets



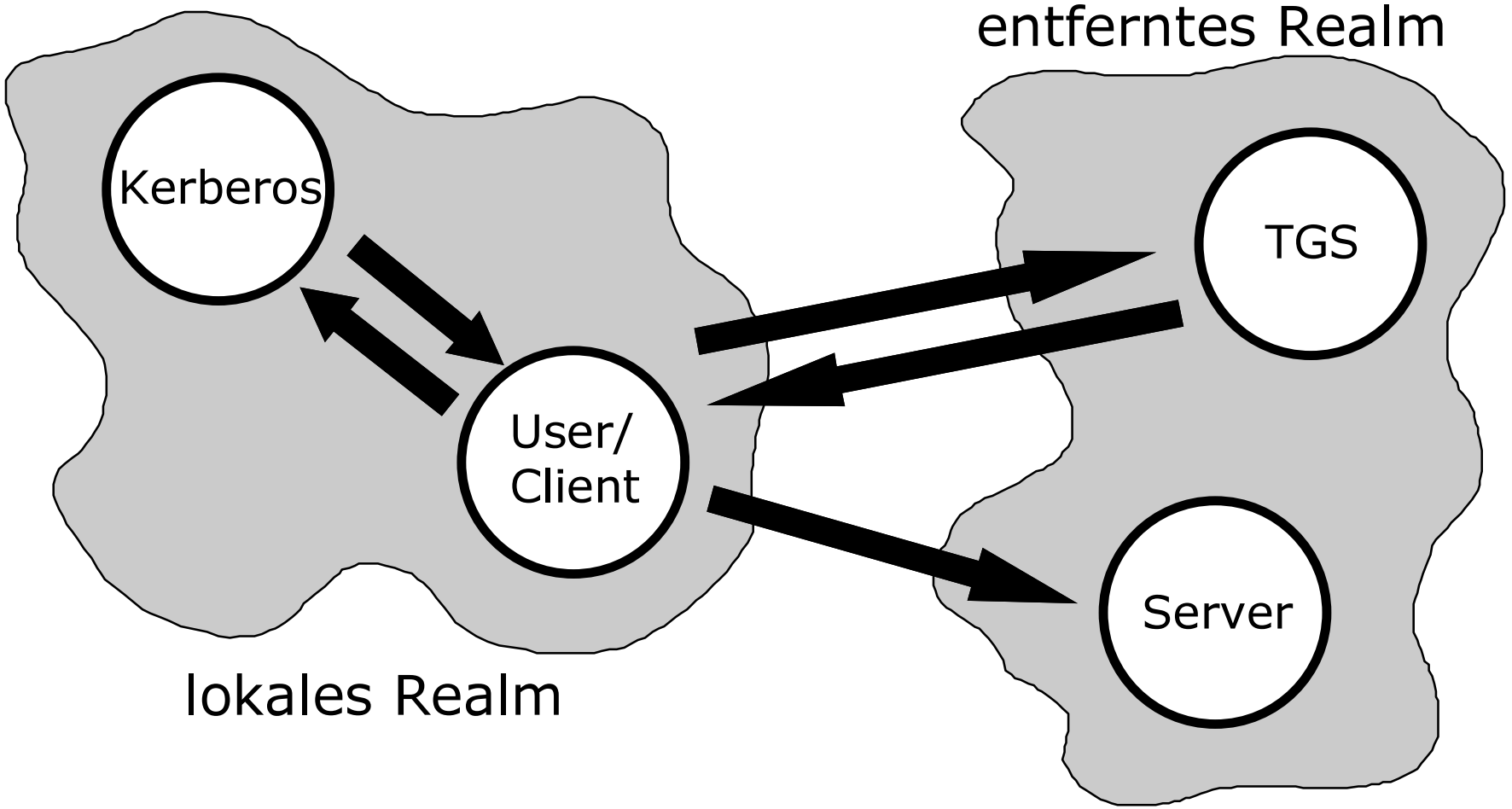
c: Client tgs: Ticket Granting Server K_x: x's Private Key
$K_{x,y}$: Session Key für x und y $T_{x,y}$: Ticket für x, y zu benutzen

Kerberos

Anfordern eines Dienstes



c: Client A_x: Authenticator von x K_x: x's Private Key
$K_{x,y}$: Session Key für x und y $T_{x,y}$: Ticket für x, y zu benutzen

Kerberos

Interaktion mit anderen Kerberoi