HTTP(S) und Webserver

Vorlesung Hochschule Mannheim – Webanwendungen

Martina Kraus

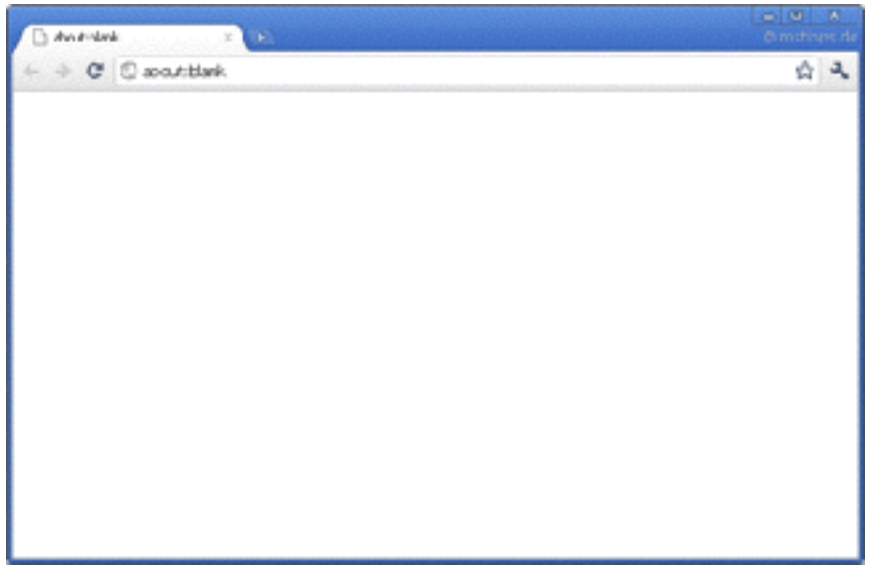
HyperText Transfer Protocol (HTTP)

- einfaches Protokoll für die Übertragung von Dateien (Hypertext-Dokumenten) über das Internet
- Request/Response-Verfahren über eine TCP-Verbindung
  - mehrere Nachrichten über dieselbe Verbindung
  - Verbindung kann auch nach jedem Request abgebaut werden
- Klartext-Protokoll
  - reines ASCII
  - unverschlüsselt
  - zeilenorientiert

HTTP - Geschichte

- Wer hat‘s erfunden? Tim Berners-Lee und Roy Fielding
  …zusammen mit HTML
- 1996: HTTP/1.0: Jede Anfrage eine neue TCP-Verbindung
- 1999: HTTP/1.1: Mehrere Anfragen pro TCP-Verbindung
- 2015: HTTP/2: Beschleunigung (Multiplex), Datenkompression, push-Verfahren

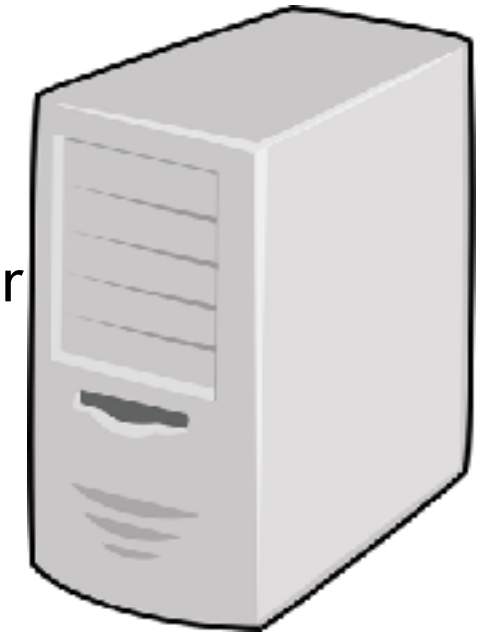
Überblick



1. Anfrage - Request

2. Bearbeitung der Anfrage

3. Antwort - Response



HTTP-Protokoll (Beispiel)

Befehl vom Client an Server

```
GET /index.html
```

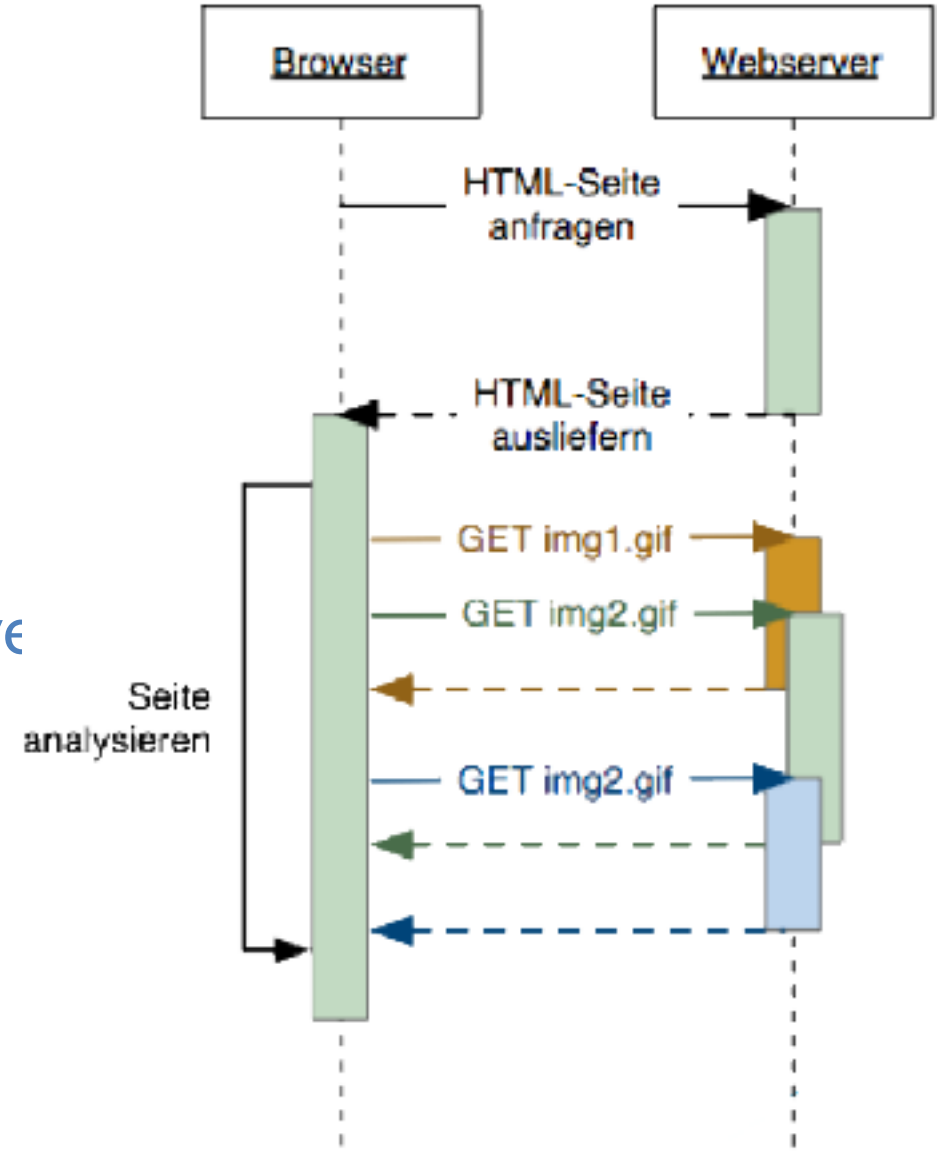
Antwort von Server an Client

```
<html>
<head>
        <title>Hello!</title>
</head>
<body>
        <h1>Hello World!</h1>
</body>

</html>
```

Interaktion bei HTTP

- Browser stellt Anfrage nach HTML-Seite
- Server liefert Seite
- Browser analysiert Seite und fordert alle abhängigen Ressourcen (parallel) vom Serve



Eigenschaft: Zustandslosigkeit

- HTTP ist grundsätzlich zustandslos
  - keine Zustand zwischen zwei Aufrufen eines Clients
    ➔ Server „weiß“ bei einer zweiten Anfrage nichts von der ersten Anfrage
  - Verbindung kann zwischen Aufrufen abgebaut werden ohne Verlust von Daten, Informationen, etc.
  - Browser-Sitzungen brauchen nicht geschlossen zu werden
- Das Protokoll ist zustandslos! Für den Anwender „fühlt“ es sich anders an
  - Cookies
  - Sessions
  - sind aber kein Bestandteil des Protokolls

Definition: idempotent / sicher

Idempotent:

Einen HTTP-Request, bei dem mehrfache (erfolgreiche) Zugriffe, die gleiche Wirkung haben wie ein einmaliger (erfolgreicher) Zugriff, nennt man ***idempotent***.

Sicher:

Einen HTTP-Request, bei dem ein (erfolgreicher) Zugriffe **keine** unerwünschten Seiteneffekte auf dem Server erzeugt nennt man ***sicher***.

Kein Zusammenhang mit Passwörtern, Authentifizierung, Verschlüsselung.

HTTP – in a nutshell

Aufforderung an den Server,

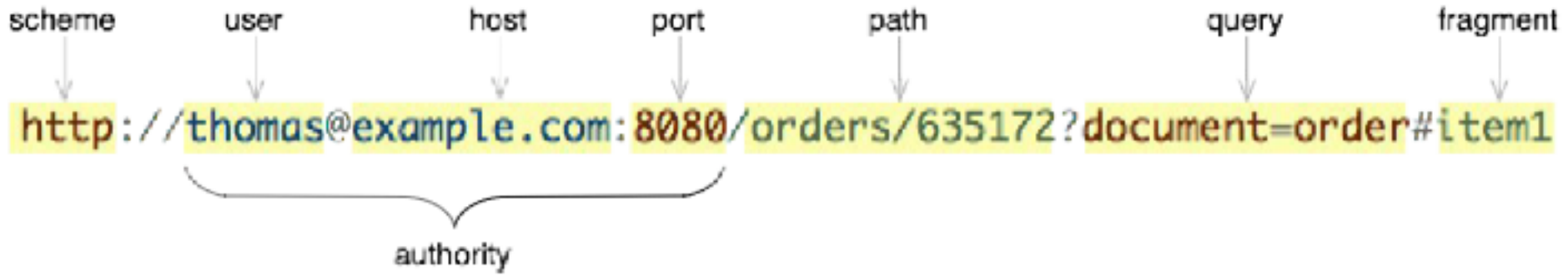
- **…auf eine bestimmte Ressource**
- …eine bestimmte Aktion
- …mit bestimmten zusätzlichen Daten/Optionen

durchzuführen

Ressource: Adressierung per URI

- HTTP arbeitet immer mit einer Ressource
- Bspw. eine Datei: physische Ressource:
  http://www.server.de/file.html
- 0der abstrakte Ressource (vom Dienstanbieter festgelegte Objekte):
  http://www.facebook.com/messages/t/hans.wurst
- Obermenge von
  - **Uniform Resource Name (URN) - ISBN** dauerhafter, ortsunabhängiger Bezeichner
  - **Uniform Resource Locator (URL) - Links**identifiziert und lokalisiert eine Ressource
- Definiert in RFC 1738 und RFC 3986

Uniform Resource Identifier (URI)



- scheme: gibt den Typ der URI an und definiert die Bedeutung der folgenden Teile (z.B. http, ftp, mailto)
- authority: kommt für viele URI-Schemata vor und bestimmt eine Instanz, die die Namen der URI zentral verwaltet
- path: hierarchisch organisierte Angabe, die auf die Ressource verweist
- query: nicht hierarchische Angaben zur Identifikation der Ressource
- fragment: verweist auf einen Teil der Ressource

Beispiele: URI

- **http**://douglasadams.com
- **file**://fileserver/Users/hans/Sites/index.html
- **file**:///Users/hans/Sites/index.html
- **mailto**:jens.doose@fantastic-bits.de
- **myprotocol**://anything/to/something

- Scheme
- Path
- Authority
- Mail-Adresse

Sonderzeichen in URI

Zeichen	Bedeutung	Codierung
<Space>	Probleme in Pfadangaben	+ oder %20
+	Ersetzt Leerzeichen in URLs	%43
@	trennt Benutzername und Passwort von Servernamen	%40
:	Trennt Hostname und Portnummer	%3A
/	trennt Pfadkomponenten voneinander	%2F
?	leitet Parameter ein	%3F
&	trennt Parameter voneinander	%26
=	weist Parameter einen Wert zu	%3D
#	leitet Fragmentbezeichner ein	%23
%	hexadezimaler ASCII / UTF-8 Zeichencode folgt	%25

HTTP – in a nutshell

Aufforderung an den Server,

- …auf eine bestimmte Ressource
- **…eine bestimmte Aktion**
- …mit bestimmten zusätzlichen Daten/Optionen

durchzuführen

HTTP-Aktionen (Verben) (1)

- *GET* – Fordert eine Ressource an
  - sollte keine Daten auf dem Server verändern (sicher)
  - Formulardaten können in der URI übergeben werden
- *POST* – Sendet Daten zum Server
  - darf Daten auf dem Server modifizieren (nicht sicher)
  - Formulardaten werden im Body des Request übertragen
- *HEAD* – Fordert die Metadaten einer Ressource an
  - wie GET, nur ohne Body
  - wird vom Browser für das Caching verwendet
- *OPTIONS* – Liefert die Fähigkeiten des Servers

HTTP-Aktionen (Verben) (2)

- *DELETE* – Löscht bestehende Ressource
  - Für Web-Anwendungen nicht relevant (wichtig für REST)
- *PUT* – Aktualisiert eine Ressource oder legt sie neu an
  - inverse Operation zu GET
  - Für Web-Anwendungen nicht relevant (wichtig für REST)
- (weitere)

HTTP – in a nutshell

Aufforderung an den Server,

- …auf eine bestimmte Ressource
- …eine bestimmte Aktion
- **…mit bestimmten zusätzlichen Daten/Optionen**

durchzuführen

GET

GET übermittelt alle Formulardaten in der URL

- Variablen werden mit ? an die URL angehängt
- für jede Variable name=wert, durch & getrennt (Querystring)

- Querystring ist in URL sichtbar; Speicherung in Bookmarks
- Manuelle Editierbarkeit des Querystrings
- URLs werden sehr lang
  ➔ Längenbeschränkung Webserver/Browser (Apache 8190 Bytes)
- GET soll keine Daten auf dem Server ändern

POST

POST übermittelt Formulardaten im Request-Body

- kurze URLs
- Beliebig große Daten können übertragen werden
- Parameter nicht in Bookmarks speicherbar
- Parameter tauchen nicht in URL auf
- POST ist für Datenänderungen geeignet

Überblick: HTTP-Verben

VERB	Sicher	Idempotent	URI zeigt auf Ressource	Cachebar	Semantik definiert
GET	?	?	✓	✓	✓
HEAD	?	?	✓	✓	✓
PUT	?	?	✓		✓
POST	?	?			
OPTIONS	?	?			✓
DELETE	?	?	✓		✓

Überblick: HTTP-Verben

VERB	Sicher	Idempotent	URI zeigt auf Ressource	Cachebar	Semantik definiert
GET	✓	✓	✓	✓	✓
HEAD	✓	✓	✓	✓	✓
PUT		✓	✓		✓
POST					
OPTIONS	✓	✓			✓
DELETE		✓	✓		✓

HTTP-Request

- *Request-Line*: erste Zeile des Requests
  - HTTP-Methode der Anfrage (*HTTP-Verb*)
  - URI der Ressource die angefordert wird
  - HTTP-Version (heute immer HTTP/1.1)

```
GET / HTTP/1.1
Host: www.hs-mannheim.de
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
Cache-Control: max-age=0
```

HTTP-Response

- *Response-Line*: erste Zeile des Responses
  - HTTP-Version (heute immer HTTP/1.1)
  - Status-Code

```
HTTP/1.1 200 OK
Date: Fri, 30 Sep 2011 21:10:50 GMT
Server: Apache/2.2.10 (Linux/SUSE)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Content-Encoding: gzip
Content-Length: 7010
Content-Type: text/html; charset=utf-8
```

HTTP-Status-Codes

Range	Name	Beschreibung
1xx	Informational	Response wurde ausgelöst, der Request ist aber noch nicht vollständig verarbeitet z. *B. 101 Protocol Switch*
2xx	Success	Keine Probleme aufgetreten und der Request konnte verarbeitet werden z. B. *200 OK*
3xx	Redirect	Client muss weitere Schritte durchführen, damit der Request bearbeitet werden kann z. B. *303 See Other*
4xx	Client Error	Ursache des Fehlers liegt im Verantwortungsbereich des Clients z. B. *404 Not Found*
5xx	Server Error	Ursache des Fehlers liegt im Verantwortungsbereich des Servers z. B. *500 Internal Server Error*

1xx - Informational

Code	Name	Beschreibung
100	Continue	Anfrage noch in Bearbeitung
101	Switching Protocols	Server stimmt Protokollwechsel zu (z. B. auf HTTPS/ WebSockets)
102	Processing	Anfrage läuft noch. Verhindert Timeout

2xx - Success

Code	Name	Beschreibung
200	OK	Anfrage erfolgreich bearbeitet
201	Created	Angeforderte Ressource erzeugt, URL im Location-Header
202	Accepted	Anfrage akzeptiert, wird später ausgeführt
204	No Content	Anfrage akzeptiert; keine Ergebnisdaten
206	Partial Content	Angeforderter Teil erfolgreich übertragen (Range-Request)

3xx - Redirect

Code	Name	Beschreibung
301	Moved Permanently	Ressource ist dauerhaft umgezogen, Adresse im Location-Header
303	See Other	Antwort unter Adresse, die im Location-Header steht (GET)
304	Not Modified	Antwort gegenüber voriger Anfrage unverändert
307	Temporary Moved	Ressource temporär umgezogen, Adresse im Location-Header

4xx – Client Error

Code	Name	Beschreibung
400	Bad Request	Ungültige Anfrage
401	Unauthorized	Authentifizierung notwendig
403	Forbidden	Zugriff verboten (bspw. auch: authentifiziert, aber nicht autorisiert)
404	Not Found	Ressource nicht gefunden
405	Method not allowed	Methode (GET/POST) nicht erlaubt
418	I‘m a teapot	Teekanne statt Kaffeekanne

5xx – Server Error

Code	Name	Beschreibung
500	Internal Server Error	Allgemeiner Serverfehler: Programmfehler, etc.
501	Not Implemented	Funktionalität vom Server nicht implementiert
503	Service unavailable	Service steht nicht zur Verfügung
504	Gateway Time-out	Für die Anfrage ist ein weiterer Server notwendig; die Anfrage an diesen führte zu einem Time-Out

Content Negotiation

- Der Client teilt dem Server mit, welche Medientypen und Formate er empfangen möchte bzw. versteht *(content negotiation)*

```
GET / HTTP/1.1
Host: www.hs-mannheim.de
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
Cache-Control: max-age=0
```

```
HTTP/1.1 200 OK
Date: Fri, 30 Sep 2011 21:10:50 GMT
Server: Apache/2.2.10 (Linux/SUSE)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Content-Encoding: gzip
Content-Length: 7010
Content-Type: text/html; charset=utf-8
```

MIME-Type

Content Negotiation basiert auf *MIME-Typ (MIME type)*

- *MIME = Multi Purpose Internet Mail Extensions*
- Definiert den Typ einer Nachricht im Internet
- Besteht aus *Medientyp (content type)* und *Subtyp (subtype)*
- Syntax: `Medientyp/Subtyp`
- Ursprünglich in RFC1049 (E-Mail) spezifiziert, aktuelle Spezifikation ist RFC2047
- Verwaltung durch IANA (Internet Assigned Numbers Authority)

Medientyp

Code	Name	Beschreibung (Medientyp/Subtyp)
`text`	Textuelle Daten	`text/plain, text/html, text/xml`
`audio`	Audio	`audio/ogg, audio/mp4, audio/mpeg`
`image`	Bilder	`image/jpeg, image/tiff, image/gif`
`application`	Programmspezifische Daten	`application/zip, application/pdf; application/json`
`multipart`	Mehrteilige Daten	`multipart/signed`

Komprimierung

- Die meisten Webserver unterstützen on-the-fly-Komprimierung
- Aktivierung per Konfiguration
- Client spezifiziert bei Request die akzeptierten Kompressionsverfahren (Header: `accept-encoding: gzip, deflate`)
- Formate (Auswahl)
  - deflate
  - gzip
  - Brotli (br)
- Text: 60-85%. Bilder/Medien: 0%

Authentication vs. Authorization

Authentication:
Wer ist jemand?
Ist der User wirklich der, der er vorgibt zu sein?
Passen Benutzername und Passwort zusammen?

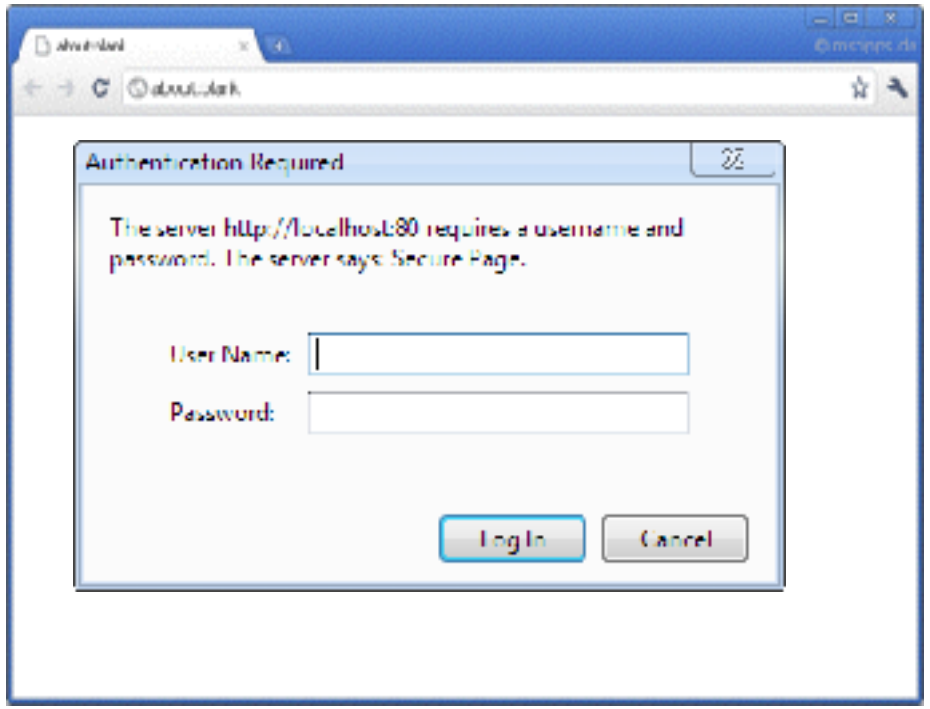
Authorization:

NACH der Authentication:

Darf die authentifizierte Person die angefragte Aktion durchführen?

- Darf User „Hans“ diese Datei löschen?
- Darf User „Jasmine“ auf diese URL zugreifen?

Basic Authentication

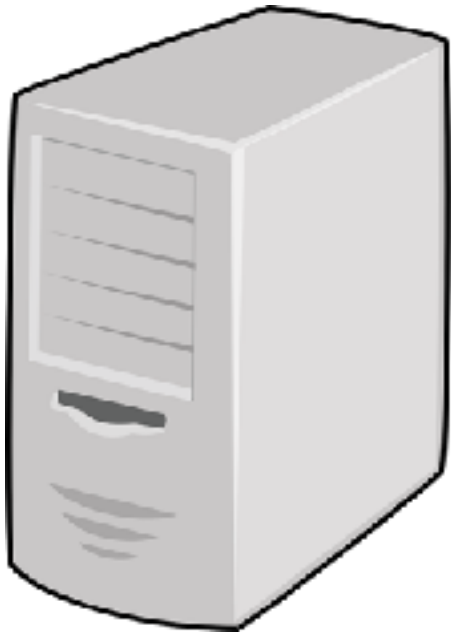


1. Request

2. 401 Unauthorized
`WWW-Authenticate: Basic realm="Domain"`

3. Anfrage + Credentials

4. 200 OK Response



Basic Authentication

- Einfaches Verfahren
- Oft per Webserver konfigurierbar
  → kein Aufwand für Skripte etc.
- Nachteil: Base64-Kodierung. Keine Verschlüsselung! Passwort ist abhörbar!
- OK bei HTTPS (verschlüsselte Verbindung)

Digest Access Authentication

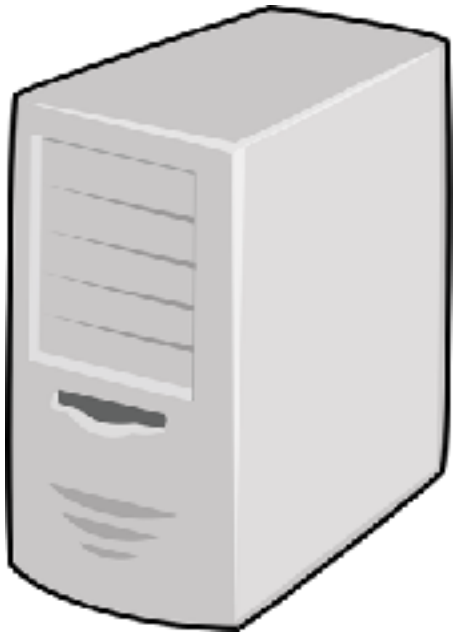
Berechnung Hash:
Username, Password + Token

1. Request

2. 401 Unauthorized
`WWW-Authenticate: Basic realm="Domain"`
+ zufälliges Token

3. Anfrage + Hash

4. 200 OK Response



Digest Access Authentication

- Einfaches Verfahren
- Oft per Webserver konfigurierbar
  → kein Aufwand für Skripte etc.
- Token wird jedes Mal neu generiert, vom Server generiertes Token kann für eine gewisse Zeit verwendet werden
- Durch Berechnung eines Hash-Wertes Rekonstruktion des Passworts „unmöglich“… naja:
- Meist MD5-Hashberechnung – inzwischen unsicher
- Man-in-the-Middle attacks, keine Verifikation des Servers