

# Basiskonzepte des Rechnerbetriebs

E-Mail, WWW und Sicherheit

Arvid Terzibaschian

# E-Mail

# Geschichte der E-Mail

---

- ▶ **E(lectronic)-Mail ab Anfang 1970:**
  - ▶ vor der Entwicklung des Internets
  - ▶ ursprünglich Bezeichnung für alle elektronisch ausgetauschten Dokumente (auch Fax)
  - ▶ erst nur Text, seit Anfang 90er Jahre auch Dateianhänge
    - ▶ Multipurpose Internet Mail Extension (MIME)
  - ▶ erste E-Mail-Varianten setzten voraus, dass beide Teilnehmer gleichzeitig online sind (vgl. Skype)
- ▶ **seit 1982 standardisiertes Protokoll zum Verschicken: SMTP**
  - ▶ Datenaustausch mit Store-and-Forward-Modell
    - ▶ E-Mails werden von Mailserver zu Mailserver gereicht
    - ▶ Clients schickt und empfängt bei Bedarf

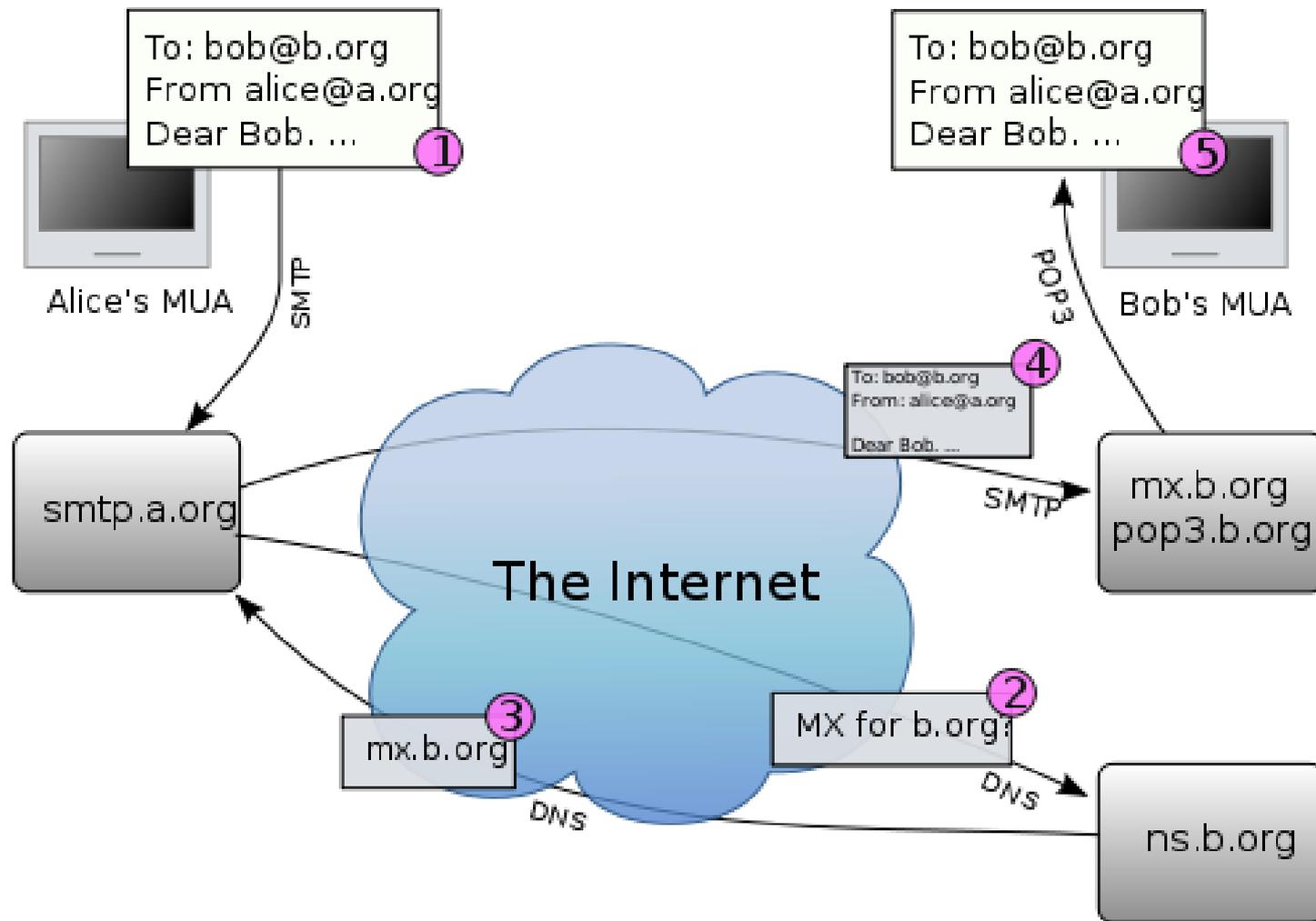


# E-Mail Adressen

---

- ▶ auf klassischen UNIX-Mailservern hat jeder User ein Postfach
  - ▶ `/var/mail/[Benutzername]`
- ▶ jeder Nutzer hat (min.) eine Adresse
  - ▶ Benutzername@domainname
  - ▶ Z.B. [terzibas@uni-potsdam.de](mailto:terzibas@uni-potsdam.de)
- ▶ jeder Nutzer kann außerdem Alias-Adresse besitzen
  - ▶ Z.B: [arvid.terzibaschian@uni-potsdam.de](mailto:arvid.terzibaschian@uni-potsdam.de)
- ▶ um eine E-Mail zu versenden, werden Clients angeboten
  - ▶ genannt: Mail User Agent (MUA) (z.B. pine, xemacs, mailtool)
- ▶ heute oft direkt per Web-Client
  - ▶ ZEIK: <https://webmail.uni-potsdam.de>

# E-Mail von alice@a.org zu bob@b.org



# Aufbau einer E-Mail

---

- ▶ **Envelope („Umschlag“)**
  - ▶ nur Sender, Empfänger, (auch bcc)
  - ▶ wichtig für den Transport, wird beim Eintreffen „zerstört“
- ▶ **Header**
  - ▶ Absender, Empfänger
  - ▶ verwendete Server zum Versand
  - ▶ Formatinformationen (Schriftsatz mit Umlauten, ...)
- ▶ **Body**
  - ▶ Text (im Klartext!)
    - ▶ per Leerzeile von Header getrennt
  - ▶ Anhänge
  - ▶ Signatur (vorgeschrieben für Geschäfts-E-Mails)



# Beispiel für einen E-Mail Header

---

**Return-Path:** <example\_from@abc.com>

X-SpamCatcher-Score: 1 [X]

**Received:** from [136.167.40.119] (HELO abc.com)

by fe3.abc.com (CommuniGate Pro SMTP 4.1.8)

with ESMTP-TLS id 61258719 for example\_to@mail.abc.com;

**Message-ID:** <4129F3CA.2020509@abc.com>

**Date:** Wed, 21 Jan 2009 12:52:00 -0500 (EST)

**From:** Taylor Evans <example\_from@abc.com>

**User-Agent:** Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.0.1)

**X-Accept-Language:** en-us, en

**MIME-Version:** 1.0

**To:** Jon Smith <example\_to@mail.abc.com>

**Subject:** Business Development Meeting

**Content-Type:** text/plain; charset=us-ascii; format=flowed

**Content-Transfer-Encoding:** 7bit

# E-Mails verschicken

---

## ▶ Mail Transfer Agent

- ▶ leitet E-Mails per SMTP-Protokoll weiter, welche von MUA verschickt werden
- ▶ erstellt:
  - Envelope
  - Header
- ▶ verschickt Daten im Klartext – jeder Transportserver kann mitlesen

## ▶ Beispiel:

- ▶ `sendmail -t < [email-datei]`

## ▶ Sender kann frei! gewählt werden

- ▶ SPAM/Phishing-Versender nutzen dies aus

# World-Wide-Web

# WWW: World Wide Web

---

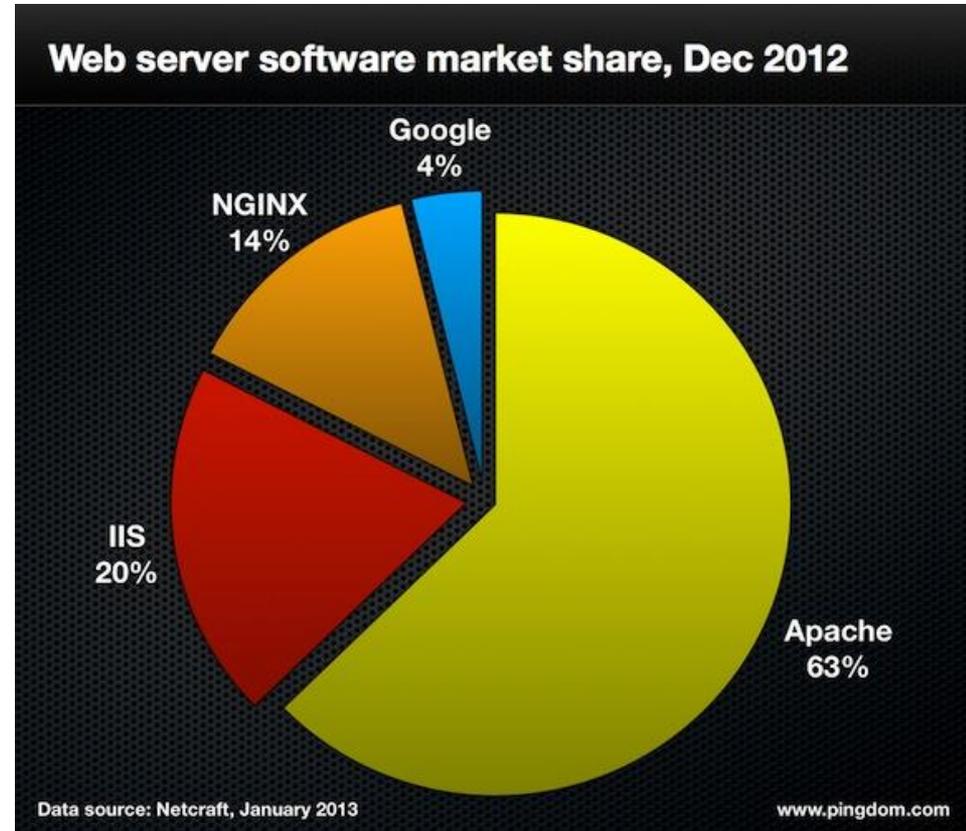
- ▶ jeder Nutzer kann frei Informationen einspeisen und auslesen
- ▶ Kommunikationsmodell Client-Server
  - ▶ Client: Internetbrowser
  - ▶ Server: HTTP-Server (z.B. Apache, LAMP/XAMP)
    - ▶ HTTP-Protokoll arbeitet über TCP-IP Protokoll
    - ▶ statische Informationen sind HTML-kodiert (Web 1.0)
      - Hyper Text Markup Language
    - ▶ dynamische Inhalte per
      - Serverseitig per Sprachen wie PHP, Perl, Java, C/C++, ...
      - Clientseitig per Java-Script, Flash, Java
  - ▶ Darstellung von HTML und Java-Script sind browserabhängig!
    - ▶ Kompatibilitätsprobleme beim Entwickeln von Webseiten
- ▶ Verknüpfung von Informationen per „Hyperlink“



www.shutterstock.com 88820890

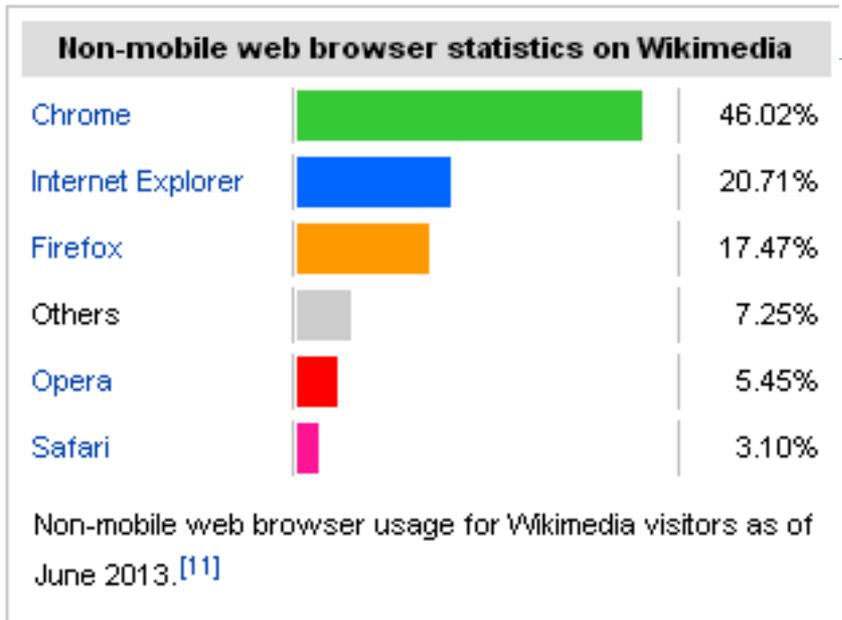
# WWW-Server

- ▶ stellt Informationen bereit
  - ▶ auf (öffentlichen) Domains per HTTP-Anfragen auf Port 80
  - ▶ De-Facto-Standard ist Apache-Webserver
    - ▶ kostenlos
    - ▶ für Linux und Windows
  - ▶ Aufgaben:
    - ▶ Daten bereitstellen
    - ▶ Zugriffskontrolle
    - ▶ Dienste anbieten



# WWW-Clients

- ▶ umgangssprachlich „Browser“
- ▶ stellen Anfragen an Server
- ▶ interpretieren und stellen dar:
  - ▶ HTML,
  - ▶ Layout-Informationen (CSS),
  - ▶ JavaScript.
- ▶ Interpretation nicht immer 100% standardisiert
  - ▶ Webseiten müssen oft für jeden Browser angepasst werden



# WWW-Adressen

---

- ▶ Adressierung von Ressourcen Serverabhängig!
- ▶ meist durch URL (Uniform Resource Locator)
  - ▶ `protokoll://[user:passwort@]hostname[:port]/pfad/datei`
  - ▶ Protokoll im Netz: http
    - ▶ weitere z.B. ftp/file, ...
  - ▶ Benutzername und Passwort selten nötig
  - ▶ Port ist Standard 80
  - ▶ Datenformat ist HTML
  - ▶ Beispiel
    - ▶ <http://arvid:test@www.cs.uni-potsdam.de:80/ml/teaching/ws13/epr.html>
    - ▶ blau = optional

# HTML-Dokumente

---

- ▶ beschreiben den (statischen) Inhalt von Webseiten
- ▶ HTML ist Mischung aus Text und Struktur-Tags

```
<tag attribut="wert">Text</tag>
```



- ▶ Tags können verschachtelt sein.
- ▶ Nicht alle Tags benötigen End-Tag </...>.
- ▶ Tags können Attribute haben:
  - ▶ Ziel eines Links : `<a href="ziel-url">Dies ist ein Link</a>`
  - ▶ Quelle eines Bildes: ``
- ▶ Liste der möglichen Tags ist festgelegt
  - ▶ Beispiele: `<a>`, `<p>`, `<li>`, `<img>`, ...
- ▶ detaillierter Lehrgang zu HTML auf
  - ▶ <http://www.selfhtml.org>

# HTML-Dokumente

---

## ▶ Beispiel für eine Webseite



```
<html>
  <head>
    <title>Meine erste HTML-Seite</title>
  </head>
  <body>
    <h1>Basiskonzepte des Rechnerbetriebs</h1>
    <p>Textabsatz mit viel Informationen</p>
    
    <a href="http://www.cs.uni-potsdam.de">Link zum IfI</a>
  </body>
</html>
```

# HTML-Dokumente

## ► einige wichtige Tags:

Tag	Erklärung
<code>&lt;html&gt;...&lt;/html&gt;</code>	Beginn/Ende HTML-Dokument
<code>&lt;head&gt;...&lt;/head&gt;</code>	Meta-Daten-Bereich (Titel,Autor,...)
<code>&lt;title&gt;...&lt;/title&gt;</code>	Titel der Webseite
<code>&lt;body&gt;&lt;/body&gt;</code>	Bereich Inhalt der Webseite
<code>&lt;p&gt;&lt;/p&gt;</code>	Text-Absatz
<code>&lt;hN&gt;...&lt;/hN&gt;</code>	Überschrift der Kategorie $N = 1 \dots 6$
<code>&lt;br&gt;</code>	Zeilenumbruch
<code>&lt;div&gt;&lt;/div&gt;</code>	Strukturierungselement (Container)
<code>&lt;!-- ... --&gt;</code>	unsichtbarer Kommentar
<code>&lt;img src="URL"&gt;</code>	Bild anzeigen von Adresse <i>URL</i>
<code>&lt;a href="URL"&gt;Text&lt;/a&gt;</code>	Verweis auf Adresse <i>URL</i> , nur <i>Text</i> sichtbar

# HTML-Dokumente - Layout

---

- ▶ HTML-Style-Tags werden „Cascading Style Sheet“ (CSS) genannt
  - ▶ legen Formatierung von Text, Farben sowie Anordnung von Elementen fest
  - ▶ sind statisch
- ▶ Angabe z.B. per Style-Tag direkt im HTML-Dokument :

```
<style type="text/css">  
  Selektor { Attribut: Wert; ... Attribut: Wert }  
</style>
```

  - ▶ *Selektor* wählt Elemente aus HTML-Dokument, für die der Stil festgelegt werden soll
  - ▶ Menge von *Attribut-und-Wert-Paaren* setzen den Stil fest

# HTML-Dokumente – CSS-Beispiel

```
<html>
<head>
  <title>Meine erste HTML-Seite</title>
  <style type="text/css">
    .mystyle { color : red;}
  </style>
</head>
<body>
  <h1 class="mystyle">Basiskonzepte Rechnerbetrieb</h1>
  <p>Textabsatz mit viel Informationen</p>
  
  <a href="http://www.cs.uni-potsdam.de">Link zum IfI</a>
</body>
</html>
```

# HTML-Dokumente: Formulare

- ▶ Formulare erfragen Informationen vom Nutzer und erlauben so dem Server entsprechende Aufgaben zu erfüllen
- ▶ typische Anwendungen:
  - ▶ Anmeldungen,
  - ▶ Bestellungen,
  - ▶ Kontaktformular.
- ▶ Client schickt Formular-daten an Server
- ▶ Server verarbeitet diese (z.B. mit PHP, ...) und handelt entsprechend:
  - ▶ Nutzer in Datenbank anlegen,
  - ▶ Bestellung in Datenbank anlegen
  - ▶ E-Mail verschicken

Please complete the form below. Mandatory fields marked \*

Delivery Details	
Name *	<input type="text"/>
Address *	<input type="text"/>
Town/City	<input type="text"/>
County *	<input type="text"/>
Postcode *	<input type="text"/>
Is this address also your invoice address? *	
<input type="radio"/> Yes	
<input type="radio"/> No	

# HTML-Dokumente: Skripte

---

- ▶ *Skripte* erlauben *dynamische* Inhalte in Webseiten
- ▶ **Serverseitig:**
  - ▶ zum dynamischen Erstellen eines HTML-Dokuments
  - ▶ typische Sprachen PHP, CGI, Perl, Java, ...
  - ▶ typische Anwendungen:
    - ▶ Einfügen von Nutzerdaten (Profilbild, Nachrichten, Namen, ...) in die HTML-Webseite vor dem Abschicken zum Webbrowser
- ▶ **Clientseitig:**
  - ▶ typische Sprachen JavaScript, Flash, Java-Applets
  - ▶ typische Anwendungen:
    - ▶ dynamische Layouts auf der Webseite (Menüs), Flash-Spiele,
    - ▶ Video und Musik
  - ▶ Skripte werden in speziellen Tags in HTML-Dokumente eingebunden

# Cookies

---

- ▶ **Cookies sind kleine Datenspeicher im Webbrowser**
  - ▶ Dienen der Identifizierung eines Nutzers
    - ▶ Server schickt Cookie bei erster Anfrage an Client („Browser“)
      - Z.B. eindeutige Session-ID
    - ▶ Browser speichert Cookie lokal
    - ▶ Browser schickt Cookie bei jeder Anfrage an Server
    - ▶ Server kann damit leicht Anfrage dem Nutzer zuordnen
      - Session-ID im Cookie identifiziert Nutzer auf Server
  - ▶ Cookies beinhalten keinen Script-Code, sind reine Datenspeicher
  - ▶ Cookies sind beschränkt auf eine Domain.
    - ▶ Browser darf bei Anfragen an andere Domains Cookies nicht mitsenden



# Sicherheit

# Sicherheit im Netz

---

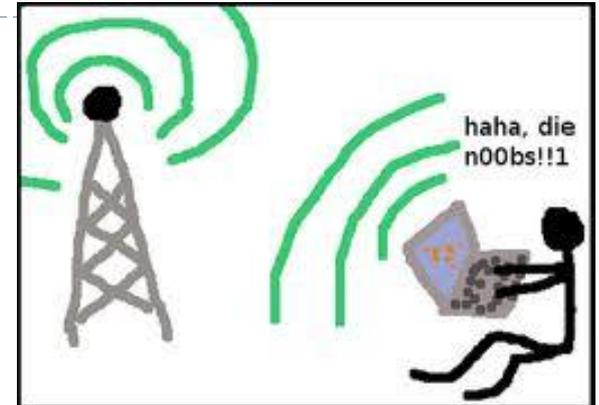
- ▶ **Problem:** Viele Protokolle übertragen Daten (auch Passwörter) im Klartext
  - ▶ SMTP,
  - ▶ HTTP,
  - ▶ TELNET,
  - ▶ FTP,
  - ▶ TCP/IP generell.
- ▶ jeder Netzteilnehmer (z.B. ein Host, der Pakete weiterleitet) kann Zugang zu sensiblen Informationen erhalten
- ▶ Lösungsansatz A): Häufiges Ändern der Passwörter, Panik und Angst.
- ▶ Lösungsansatz B): Kryptographie.



# Mögliche Angriffe im Netz

## ▶ Packet-Sniffing:

- ▶ Auslesen von vorbeikommenden Datenpaketen mit der Netzwerkkarte

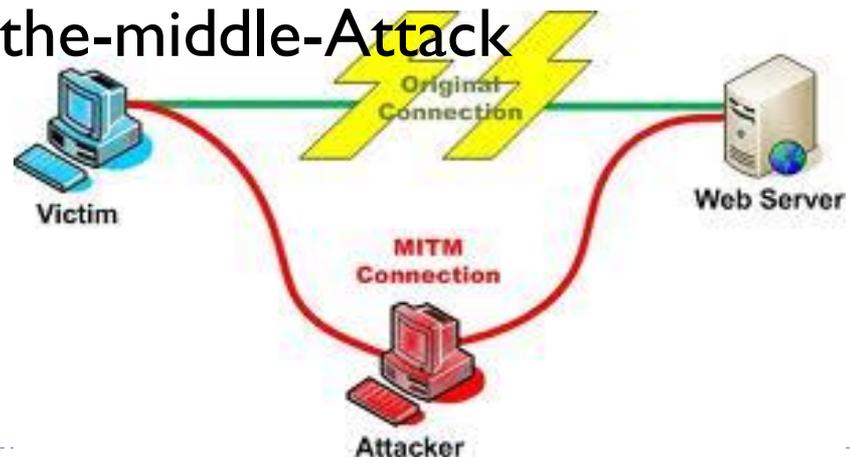


## ▶ Hijacking

- ▶ nach Passworteingabe eines Nutzers seine Session übernehmen

## ▶ DNS-Spoofing & Man-in-the-middle-Attack

- ▶ Manipulation des Nameserver bzw. Umleiten über den eigenen Server



# Sicherheit durch Kryptographie

---

## ▶ Was kann Kryptographie absichern?

- ▶ Schutz von Daten vor nicht-autorisiertem Zugriff
  - ▶ bei Übertragung zwischen zwei Hosts
- ▶ Erkennen von Datenmanipulation
- ▶ Verifikation des Autors von Nachrichten (z.B. mit zertifizierten Schlüsseln)



## ▶ Was kann Kryptographie nicht sicherstellen?

- ▶ Verhindern, dass ein Angreifer Daten löscht oder verändert
- ▶ Verhindern, dass ein Angreifer ein Programm/Rechner manipuliert (Trojaner, Virus, ...)
- ▶ Verhindern, dass ein Angreifer z.B. einen Quantencomputer baut und alles entschlüsselt (siehe ENIGMA vs. Turing)

# Arten von Kryptographie

---

- ▶ **symmetrisch: Ein Schlüssel für Sender und Empfänger**

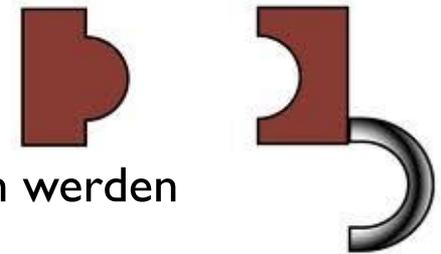
- ▶ Mono- oder polyalphabetische Verschlüsselung
- ▶ UNIX-Programm „crypt“ (ENIGMA-Maschine)
- ▶ DES/3DES/AES (Data/Advanced Encryption Standard)
  - ▶ Z.B. EC-PIN



- ▶ **asymmetrisch: Jeder Teilnehmer hat ein Schlüsselpaar**

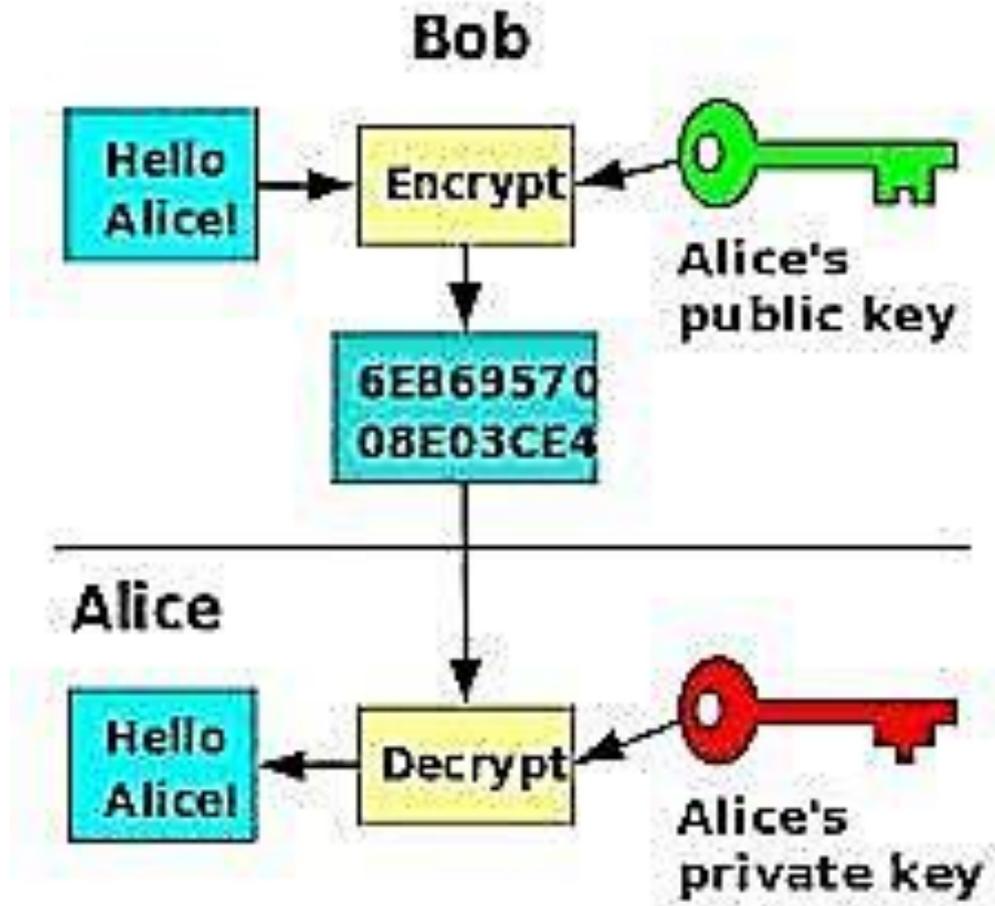
- ▶ öffentlicher Schlüssel „public key“
  - ▶ Dient lediglich zum Verschlüsseln
  - ▶ Kann ohne Risiko weitergegeben werden
- ▶ privater Schlüssel „private key“
  - ▶ Dient nur zum Entschlüsseln, darf niemals weitergegeben werden
- ▶ Algorithmus z.B. RSA (Rivest, Shamir, Adleman)
  - ▶ wesentlich langsamer als symmetrische Verfahren
  - ▶ basieren meist auf Eigenschaften großer Primzahlen und Primfaktoren

Private key      Public key

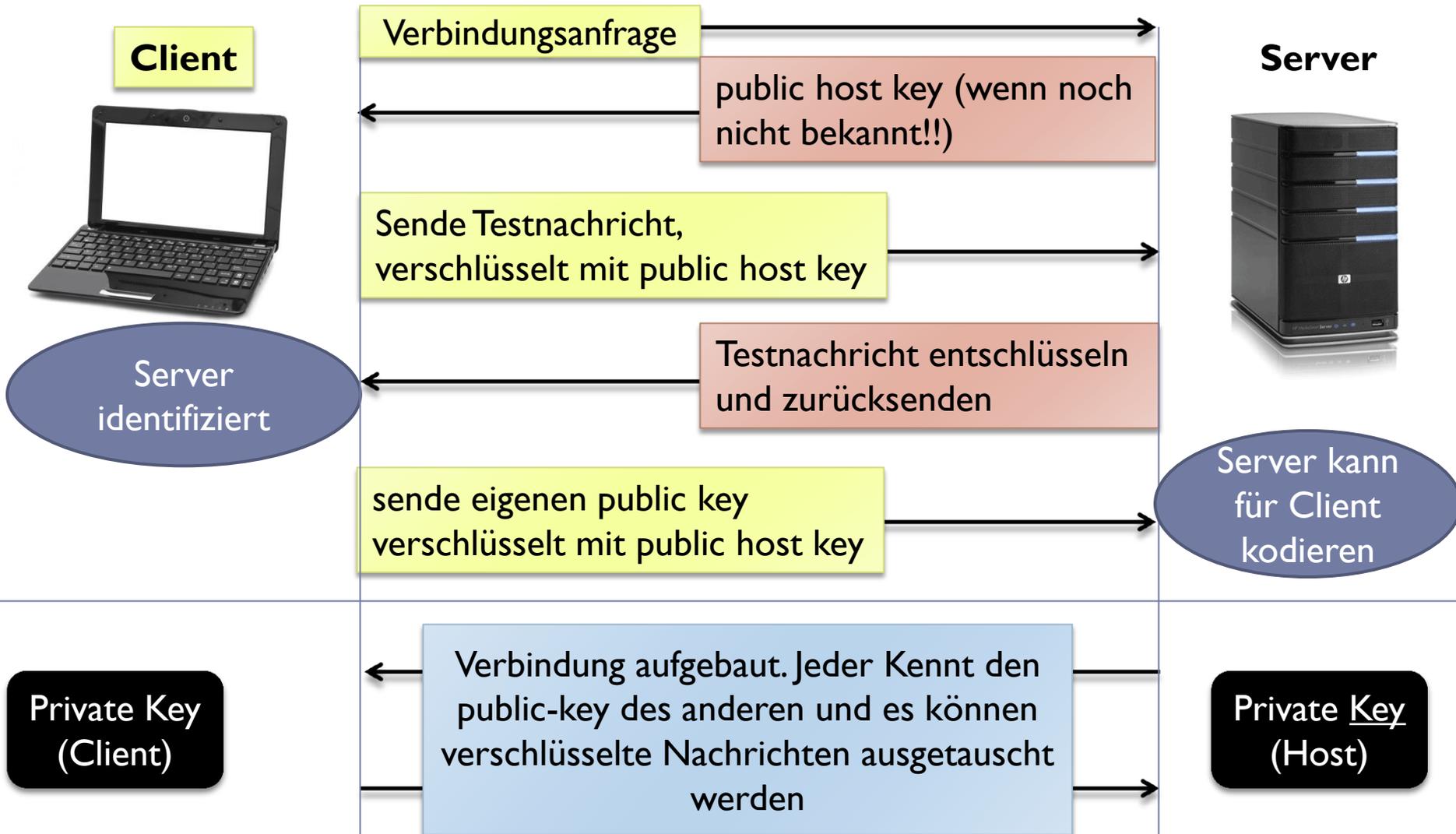


# Sicheres Kommunizieren mit Public & Private Key

---



# SSH Verbindungsaufbau (vereinfacht)



# SSL

---

- ▶ **SSL: Secure-Socket-Layer**
  - ▶ Verschlüsselungsverfahren ähnlich dem SSH-Verfahren
  - ▶ bietet eine abhörsichere Netzwerkschicht
  
- ▶ **HTTP over SSL (HTTPS)**
  - ▶ Erkennbar an URLs: `https://...`
  - ▶ Kommunikation mit Webserver ist nun nicht mehr im Netzwerk abhörbar
  - ▶ Pflicht z.B. bei Banktransaktionen

# Vielen Dank!

---

- ▶ Bei Fragen einfach eine Mail an:
  - ▶ [arvid@cs.uni-potsdam.de](mailto:arvid@cs.uni-potsdam.de)
- ▶ Nächste Woche beginnt die Vorlesung EPR
  - ▶ Immer 1x wöchentlich + Übung bis Anfang Februar 2014