# Vorlesung Unix-Praktikum

## 14. Reguläre Ausdrücke, GPG

Carsten Gnörlich

Rechnerbetriebsgruppe
Technische Fakultät
Universität Bielefeld

08. Februar 2016

# Willkommen zur vierzehnten Vorlesung
## Was gab es beim letzten Mal?



- Gerätedateien: `/dev/null`, `/dev/shm`
- Ausgabekanäle und -umleitung
- Partitionen und Dateisystem anlegen
- `watch` und `tail -f`

# Willkommen zur vierzehnten Vorlesung
## Was machen wir heute?



Reguläre Ausdrücke
- Motivation
- Elemente
- ERE vs. BRE

E-Mail
- Protokolle
- Schwachstellen
- Verschlüsselung

# Wildcards zur Beschreibung von Suchmustern

## Im Kontext von Dateien / Verzeichnissen



- `ls uebung*.txt` paßt auf:

  `uebung1.txt`
  `uebung12.txt`
  `uebungsaufgabe.txt`

- `ls uebung1?.txt` paßt auf:

  `uebung11.txt`
  `uebung19.txt`

  aber nicht auf:

  `uebung1.txt`
  `uebung101.txt`

# Grenzen der bisherigen Ausdrucksmittel

Viele Suchmuster können wir noch nicht abbilden, z.B:



- "alle Dateien mit *uebung<nummer>.txt*", also:

  `uebung1.txt`
  `uebung2.txt`
  `uebung10.txt`
  `uebung107.txt`

  aber nicht: `uebungen.txt`

- "alle Dateien *uebung11.txt* bis *uebung13.txt*", also:

  `uebung11.txt`
  `uebung12.txt`
  `uebung13.txt`

  aber z.B. nicht: `uebung14.txt`

# Erweiterte reguläre Ausdrücke
## Lösung: ERE (Extended Regular Expressions)



Standardisiert und verwendet u.a. in:

- `bash`
- `egrep`, `grep`, `sed`, `awk`
- Apache, PHP, Javascript
- Javascript
- MS Visual Studio, MS Frontpage
- Editoren, u.a. `vi` und `emacs`
- u.v.m

# Erweiterte reguläre Ausdrücke
## Rekursiver Aufbau



Erweiterte reguläre Ausdrücke sind rekursiv aufgebaut:

| ERE | Bedeutung |
|---|---|
| x | ein Zeichen "x" ist ein ERE. |
| RS | wenn R und S ERE sind, dann ist auch RS einer. |
| . | jedes beliebige Zeichen (außer \n / Zeilenschaltung) |
| [abc] | Zeichenklasse (Erläuterung: gleich!) |
| (R) | R selbst (einfach nur Klammerung) |
| R\|S | R oder S |
| R* | R kein- oder mehrmal |
| R+ | R ein- oder mehrmal |
| ... und weitere Elemente... | |

- Zeile 1+2 sind die Grundlage
  (ausreichend für konstante Zeichenfolgen wie `liste.dat`)
- alles Übrige kann man beliebig einbauen und kombinieren

# Erweiterte reguläre Ausdrücke
## Auswertung des Suchmusters



- Muster soll *exakt* passen (Bsp: Dateien)

```
> ls aufgabe.txt
-rw-r--r-- 1 cg cg 5566 26.Jan 14:53 aufgabe.txt
```

- oder: größter passender Teilstring wird gesucht

```
> echo "Blumenwiesen"| sed -e "s/wie/va/"
Blumenvasen

> grep ipsum lorem.txt
Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
Proin ipsum Nulla at leo.
...
```

# Erweiterte reguläre Ausdrücke
## Platzhalter für beliebige Zeichen



Der Punkt . ist Platzhalter für *genau ein* beliebiges Zeichen:

| *Muster:* | `aufgabe.` |
|---|---|
| paßt auf: | `aufgabe1` |
| | `aufgabe2` |
| | `aufgabeX` |
| paßt nicht auf: | `aufgabe` |

# Erweiterte reguläre Ausdrücke

## Platzhalter für beliebige Zeichen



Suche nach dem Punkt selbst: `\.`

| *Muster:* | `aufgabe.\.txt` |
|---|---|
| paßt auf: | `aufgabe1.txt` |
| | `aufgabeX.txt` |
| paßt nicht auf: | `aufgabetxt` |
| | `aufgabe10.txt` |
| | `aufgabe1.doc` |

# Erweiterte reguläre Ausdrücke
## Zeichenklassen



*Eines* der Zeichen zwischen [...] muß passen:

| *Muster:* | `aufgabe[123]\.txt` |
|---|---|
| paßt auf: | `aufgabe1.txt` |
| | `aufgabe2.txt` |
| | `aufgabe3.txt` |
| paßt nicht auf: | `aufgabe4.txt` |
| | `aufgabe123.txt` |

# Erweiterte reguläre Ausdrücke

## Zeichenklassen: Bereiche entlang der lexikographischen Ordnung



Bereiche aus Ziffern und Buchstaben sind möglich:

| *Muster:* | `aufgabe[1-3c-e]\.txt` |
|---|---|
| paßt auf: | `aufgabe1.txt` |
| | `aufgabe2.txt` |
| | `aufgabe3.txt` |
| | `aufgabec.txt` |
| | `aufgabed.txt` |
| | `aufgabee.txt` |
| paßt nicht auf: | `aufgabe4.txt` |
| | `aufgabef.txt` |
| | `aufgabeA.txt` |

Minuszeichen voranstellen wenn es Teil der Klasse werden soll:

`[-a-z0-9]`

# Erweiterte reguläre Ausdrücke

## Zeichenklassen: Weiterer Anwendungsfall

Lösung für eine der einführenden Aufgabenstellungen:

| *Muster:* | `aufgabe1[123]\.txt` |
|---|---|
| paßt auf: | `aufgabe11.txt` |
| | `aufgabe12.txt` |
| | `aufgabe13.txt` |
| paßt nicht auf: | `aufgabe1.txt` |
| | `aufgabe10.txt` |
| | `aufgabe14.txt` |

# Erweiterte reguläre Ausdrücke

## Regulären Ausdruck keinmal oder mehrmals wiederholen



abcR* Ausdruck R darf keinmal oder mehrmals vorkommen

| *Muster:* | `aufgabex*\.txt` |
|---|---|
| paßt auf: | `aufgabe.txt` |
| | `aufgabex.txt` |
| | `aufgabexxx.txt` |
| | `aufgabexxxxxxxx.txt` |
| paßt nicht auf: | `aufgabey.txt` |
| | `aufgabexy.txt` |
| | `aufgabexxxxy.txt` |

# Erweiterte reguläre Ausdrücke

## Geklammerten regulären Ausdruck keinmal oder mehrmals wiederholen



`(RST)*` Ausdr. RST dürfen keinmal oder mehrmals vorkommen

| *Muster:* | `fliege(weit)*weg` |
|---|---|
| paßt auf: | `fliegeweg` |
| | `fliegeweitweg` |
| | `fliegeweitweitweg` |
| | u.s.w. |

Hinweis: bei egrep, grep -E `(RST)*`
bei grep, sed `\(RST\)*`

→ Erklärung siehe Folie 22 ff.

# Erweiterte reguläre Ausdrücke

## * und [...] werden häufig kombiniert



Vergleiche das einführende Beispiel:

| *Muster:* | `aufgabe[0-9]*\.txt` |
|---|---|
| paßt auf: | `aufgabe1.txt` |
| | `aufgabe2.txt` |
| | `aufgabe10.txt` |
| | `aufgabe102.txt` |
| | `aufgabe.txt` |

- Wie kann man `aufgabe.txt` (ohne Ziffern) noch ausfiltern?

# Erweiterte reguläre Ausdrücke

## Regulären Ausdruck ein- oder mehrmals wiederholen



abcR+ Ausdruck R darf einmal oder mehrmals vorkommen

| *Muster:* | `aufgabe[0-9]+\.txt` |
|---|---|
| paßt auf: | `aufgabe1.txt` |
| | `aufgabe2.txt` |
| | `aufgabe10.txt` |
| | `aufgabe102.txt` |
| paßt nicht auf: | `aufgabe.txt` |

# Erweiterte reguläre Ausdrücke

## Regulären Ausdruck kein- oder einmal wiederholen



abcR? Ausdruck R darf keinmal oder genau einmal vorkommen

| *Muster:* | `aufgabe[0-9]?\.txt` |
|---|---|
| paßt auf: | `aufgabe1.txt` |
| | `aufgabe2.txt` |
| | `aufgabe.txt` |
| paßt nicht auf: | `aufgabe10.txt` |

# Erweiterte reguläre Ausdrücke

## Regulären Ausdruck genau n-mal wiederholen



abcR`{n}` Ausdruck R darf genau n-mal vorkommen

| *Muster:* | `aaab{3}ccc` |
|---|---|
| paßt auf: | `aaabbbccc` |
| paßt nicht auf: | `aaabbbbccc` |
| | `aaabbccc` |

# Erweiterte reguläre Ausdrücke

## Regulären Ausdruck genau n- bis m-mal wiederholen



abcR`{n,m}` Ausdruck R darf n- bis m-mal vorkommen

| *Muster:* | `aaab{3,5}ccc` |
|---|---|
| paßt auf: | `aaabbbccc` |
| paßt auf: | `aaabbbbccc` |
| paßt auf: | `aaabbbbbccc` |
| paßt nicht auf: | `aaabbbbbbccc` |
| | `aaabbccc` |

# Erweiterte reguläre Ausdrücke

## Regulären Ausdruck genau n- bis m-mal wiederholen



Auch hier gilt Klammerung/Kombinierbarkeit mit ERE:

| *Muster:* | `jaba(daba){1,3}du` |
|---|---|
| paßt auf: | `jabadabadu` |
| | `jabadabadabadu` |
| | `jabadabadabadabadu` |
| paßt nicht auf: | `jabadabadabadabadabadu` |
| | `jabadu` |

# Erweiterte reguläre Ausdrücke

## Unterschiede zu “Basic regular Expressions” (BRE)



- egrep, grep -E: implementieren ERE wie beschrieben:

| *Muster:* | `fliege(weit)*weg` |
|---|---|
| paßt auf: | `fliegeweitweitweg` |

| *Muster:* | `fliege\(weit)\weg` |
|---|---|
| paßt auf: | `fliege(weit)weg` |

- grep, sed impl. Verhalten mit BRE genau anders herum:

| *Muster:* | `fliege\(weit)\*weg` |
|---|---|
| paßt auf: | `fliegeweitweitweg` |

| *Muster:* | `fliege(weit)weg` |
|---|---|
| paßt auf: | `fliege(weit)weg` |

# Erweiterte reguläre Ausdrücke

## Vom Unterschied ERE/BRE betroffene Zeichen



| ERE (egrep) | BRE (sed) |
|---|---|
| `(R)` | `\(R\)` |
| `R+` | `R\+` |
| `R?` | `R\?` |
| `{R}` | `{R}` |

Zitat aus `> man 7 regex`:

*"Zwei Arten von regulären Ausdrucken sind Pfusch."*

# E-Mail: Schwachstellen und Verschlüsselung
## Historie



E-Mail ist einer der ältesten Netzdienste

- Entwicklung war nicht geplant; es "ergab sich so"
  weil Nutzer über das Netzwerk kommunizieren wollten
- 1982 RFC 822 "Urvater" des E-Mail-Protokolls

(RFC = "Request for Comments"
= Standards für Internetprotokolle)

# E-Mail: Schwachstellen und Verschlüsselung

## E-Mail-Protokolle

### Abholen/Lesen

- POP3 (veraltet)
- IMAP (Ports 143 und 993)

### Absenden, Weiterleiten

- SMTP: Simple Mail Transfer Protocol
- textbasiertes Protokoll auf Port 25

- textbasierte Protokolle kann man mit `telnet` oder `netcat` "sprechen"

# E-Mail: Schwachstellen und Verschlüsselung

## E-Mail über SMTP versenden

```
> telnet mailserver 25
HELO techfak.uni-bielefeld.de
MAIL FROM: <cg@techfak.uni-bielefeld.de>
RCPT TO: <cg@techfak.uni-bielefeld.de>
DATA
From: Carsten <cg@techfak.uni-bielefeld.de>
To: cg@techfak.uni-bielefeld.de
Subject: SMTP
Eine oder mehrere Zeilen E-Mail-Inhalt
.
QUIT
```

# E-Mail: Schwachstellen und Verschlüsselung
## Nachteile von SMTP



Hauptnachteil:

- Keine Authentifizierung des Absenders!

Weitere Nachteile:

- Zustellung kann beliebig lange dauern / E-Mail kann lautlos verloren gehen
- Keine Empfangsbestätigung

  (Disposition-Notification-To von Outlook wird von den meisten Clients ignoriert / kann man wegklicken!)

# E-Mail: Schwachstellen und Verschlüsselung

## Verschlüsselung mit asymmetrischen Schlüsselpaaren



Unterzeichnen von E-Mails:

- Authentizität des Absenders ist gegeben;
  E-Mail im Klartext lesbar.

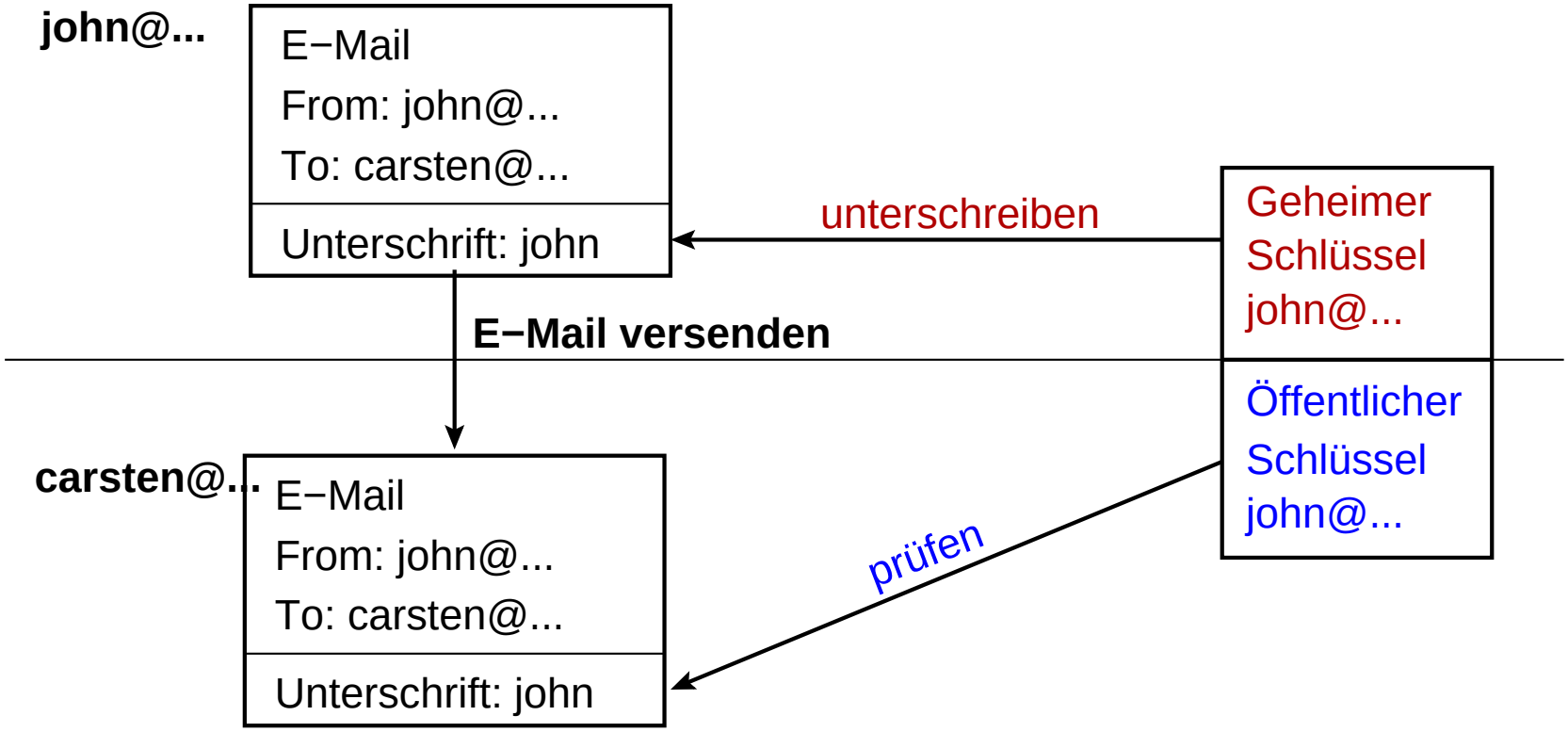
Verschlüsseln von E-Mails:

- Authentizität des Absenders ist gegeben;
  und nur der Empfänger kann die E-Mail lesen.

# E-Mail: Schwachstellen und Verschlüsselung

## Kryptographische Unterschrift






"Prüfen" = Kommt die Mail vom Besitzer des geheimen Schlüssels?

# E-Mail: Schwachstellen und Verschlüsselung

## Aufgaben für den Absender (john)



| | |
|---|---|
| 1. Schlüsselpaar generieren | einmal |
| 2. Schlüssel mit E−Mail−Identität verbinden (= E−Mail−Programm konfigurieren) | |
| 3. Öffentlichen Schlüssel hochladen | |
| 4. Unterschriebene E−Mails versenden | beliebig oft |

# E-Mail: Schwachstellen und Verschlüsselung

## Schlüsselpaar erzeugen

gpg (GNU Privacy Guard)

- john erzeugt sich ein Schlüsselpaar:

```
gpg --gen-key
```

- Im folgenden Dialog Voreinstellungen annehmen;
  Name und E-Mail-Adresse wie im E-Mail-Profil angeben.
- zum Abschluß bei der Frage "Ändern: ...
  (F)ertig/(B)eenden?" mit F antworten
- Paßphrase zum Schutz des Schlüssels eingeben

(ausführliches Beispiel: s. nächste Folien)

# Genauer Ablauf der Schlüsselerzeugung

## Auswahl des Kryptographie-Verfahrens (Voreinstellung nehmen)



```
gpg --gen-key
gpg (GnuPG) 1.4.11; Copyright (C) 2010 Free Software Foundation, I
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: Verzeichnis ‘/homes/tstgzi/.gnupg’ erzeugt
gpg: Neue Konfigurationsdatei ‘/homes/tstgzi/.gnupg/gpg.conf’ erst
gpg: WARNUNG: Optionen in ‘/homes/tstgzi/.gnupg/gpg.conf’ sind wäh
gpg: Schlüsselbund ‘/homes/tstgzi/.gnupg/secring.gpg’ erstellt
gpg: Schlüsselbund ‘/homes/tstgzi/.gnupg/pubring.gpg’ erstellt
Bitte wählen Sie, welche Art von Schlüssel Sie möchten:
   (1) RSA und RSA (voreingestellt)
   (2) DSA und Elgamal
   (3) DSA (nur unterschreiben/beglaubigen)
   (4) RSA (nur signieren/beglaubigen)
Ihre Auswahl? 1
```

# Genauer Ablauf der Schlüsselerzeugung

## Schlüssellänge und Gültigkeit (Voreinstellungen nehmen)



```
RSA-Schlüssel können zwischen 1024 und 4096 Bit lang sein.
Welche Schlüssellänge wünschen Sie? (2048) 2048
Die verlangte Schlüssellänge beträgt 2048 Bit
Bitte wählen Sie, wie lange der Schlüssel gültig bleiben soll.
         0 = Schlüssel verfällt nie
       <n>  = Schlüssel verfällt nach n Tagen
       <n>w = Schlüssel verfällt nach n Wochen
       <n>m = Schlüssel verfällt nach n Monaten
       <n>y = Schlüssel verfällt nach n Jahren
Wie lange bleibt der Schlüssel gültig? (0) 0
Schlüssel verfällt nie
Ist dies richtig? (j/N) j
```

# Genauer Ablauf der Schlüsselerzeugung

## Name, E-Mail-Adresse, Paßphrase eintragen



```
Sie benötigen eine User-ID, um Ihren Schlüssel eindeutig zu machen; das
Programm baut diese User-ID aus Ihrem echten Namen, einem Kommentar und
Ihrer Email-Adresse in dieser Form auf:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Ihr Name ("Vorname Nachname"): Max Mustermann
Email-Adresse: mmuster@techfak.uni-bielefeld.de
Kommentar:
Sie haben diese User-ID gewählt:
    "Max Mustermann <mmuster@techfak.uni-bielefeld.de>"

Ändern: (N)ame, (K)ommentar, (E)-Mail oder (F)ertig/(B)eenden? F
Sie benötigen eine Passphrase, um den geheimen Schlüssel zu schützen.

Wir müssen eine ganze Menge Zufallswerte erzeugen.  Sie können dies
unterstützen, indem Sie z.B. in einem anderen Fenster/Konsole irgendetwas
tippen, die Maus verwenden oder irgendwelche anderen Programme benutzen.

Es sind nicht genügend Zufallswerte vorhanden.  Bitte führen Sie andere
Arbeiten durch, damit das Betriebssystem weitere Entropie sammeln kann!
(Es werden noch 284 Byte benötigt.)
+++++
....+++++
Wir müssen eine ganze Menge Zufallswerte erzeugen.  Sie können dies
unterstützen, indem Sie z.B. in einem anderen Fenster/Konsole irgendetwas
tippen, die Maus verwenden oder irgendwelche anderen Programme benutzen.
........+++++
......+++++
```

# Genauer Ablauf der Schlüsselerzeugung
## Abschluß der Schlüsselgenerierung



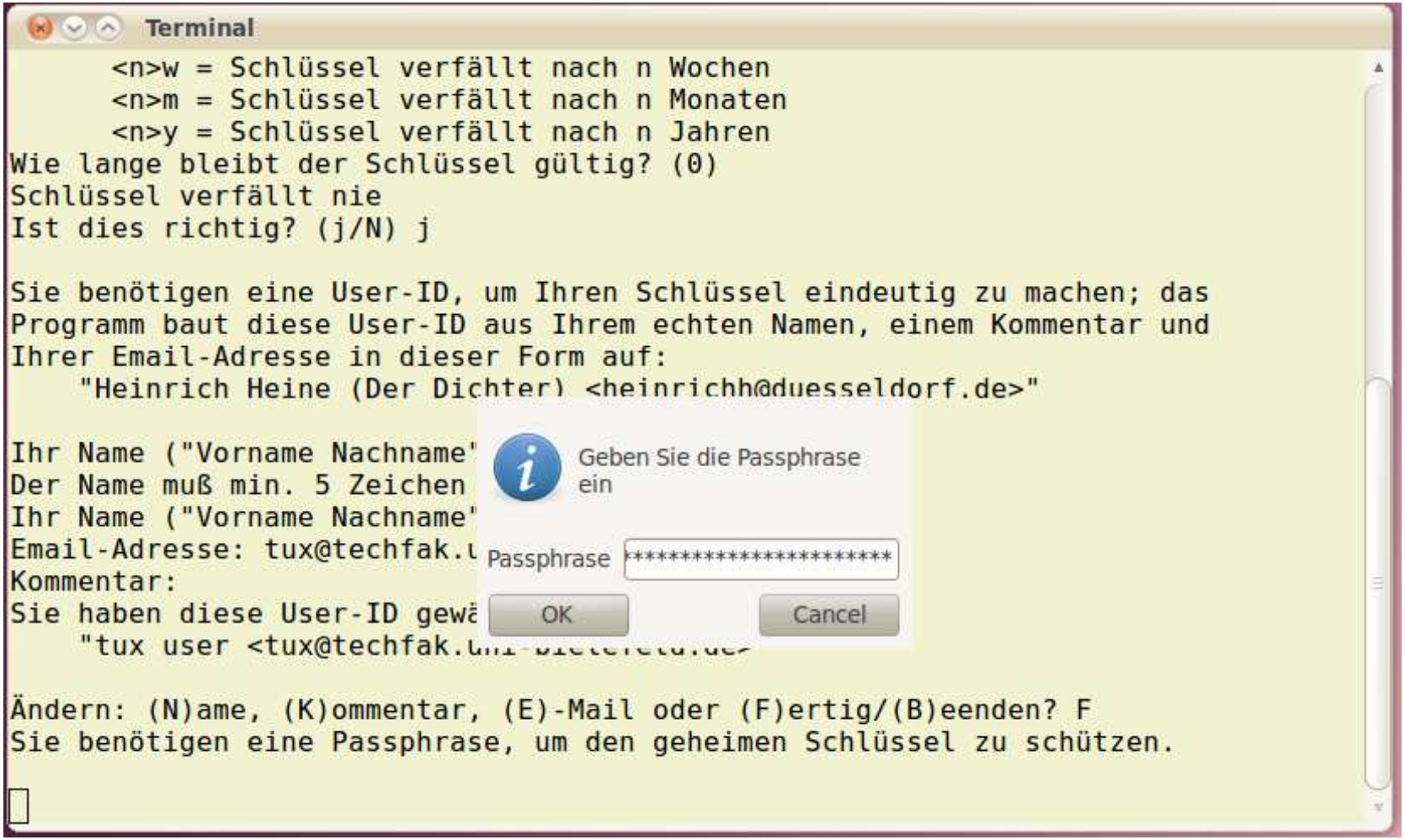
```
gpg: /homes/tstgzi/.gnupg/trustdb.gpg: trust-db erzeugt
gpg: Schlüssel 1D1C8D17 ist als uneingeschränkt vertrauenswürdig gekennzeichnet
Öffentlichen und geheimen Schlüssel erzeugt und signiert.

gpg: "Trust-DB" wird überprüft
gpg: 3 marginal-needed, 1 complete-needed, PGP Vertrauensmodell
gpg: Tiefe: 0  gültig:   1  unterschrieben:   0  Vertrauen: 0-, 0q, 0n, 0m, 0f, 1u
pub   2048R/1D1C8D17 2012-10-16
      Schl.-Fingerabdruck = 08C9 8F78 79ED E56C B60C  BAC6 DF53 6E98 1D1C 8D17
uid                  Max Mustermann <tstgzi@techfak.uni-bielefeld.de>
sub   2048R/405A97B2 2012-10-16
```

# E-Mail: Schwachstellen und Verschlüsselung

## Eingabe der Passphrase



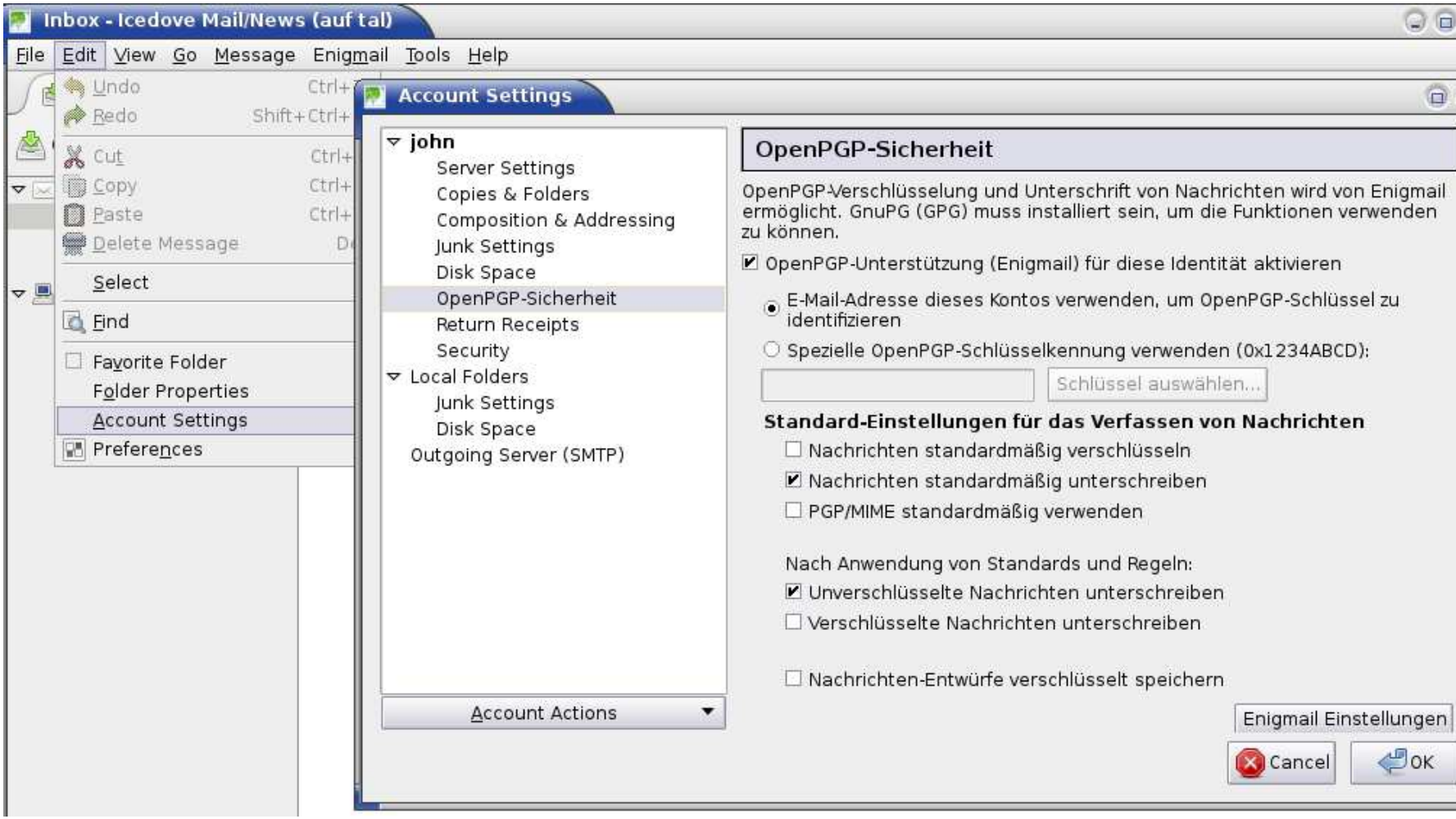
- entweder direkt in der Kommandozeile
- oder über PIN-Fenster (wenn gpg-agent läuft)



# E-Mail: Schwachstellen und Verschlüsselung

## Schlüssel in Thunderbird mit E-Mail-Identität verbinden



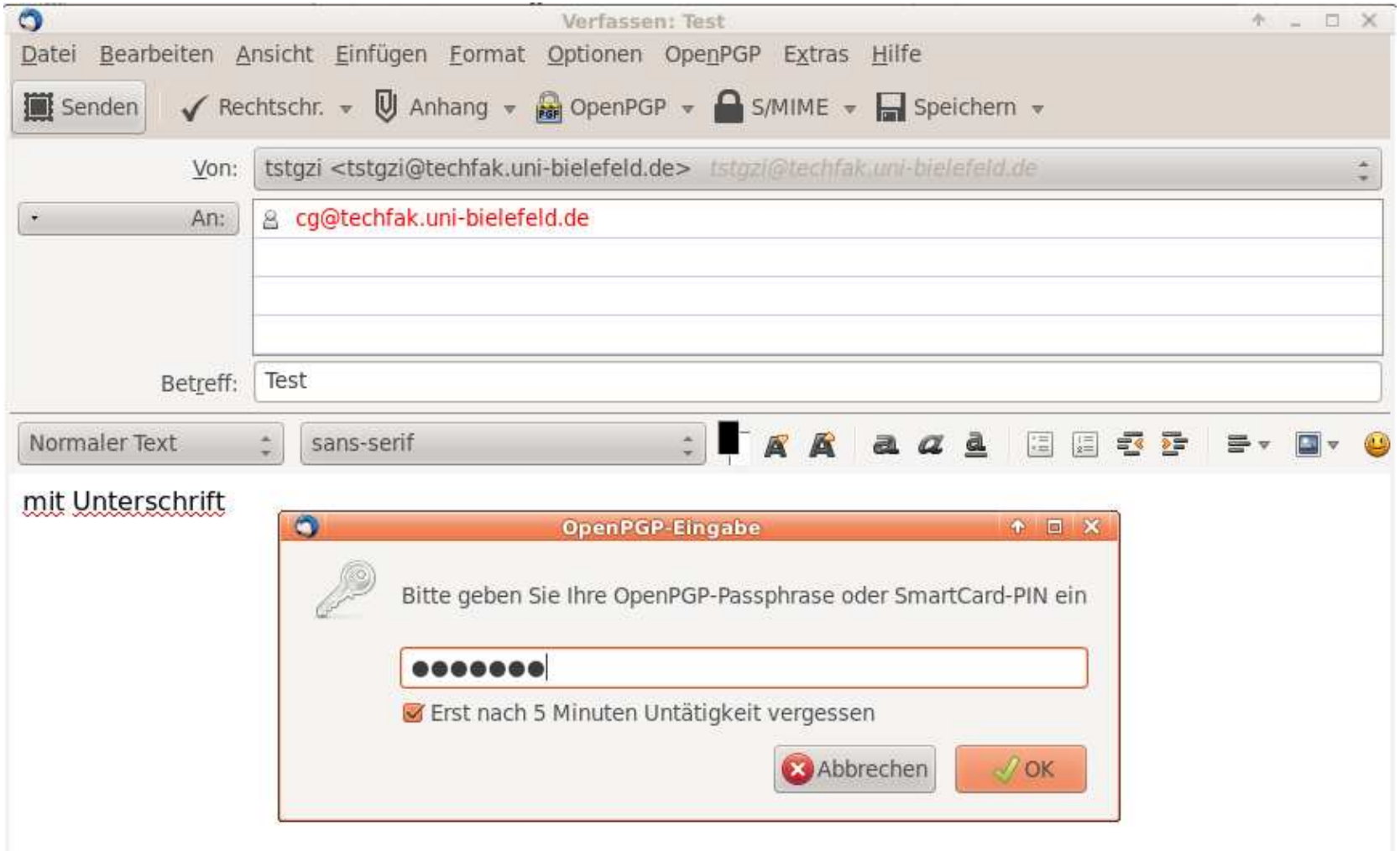


# E-Mail: Schwachstellen und Verschlüsselung

## Unterschriebene E-Mail versenden



- Abschicken der E-Mail erfordert ab jetzt GPG-Passphrase:



# E-Mail: Schwachstellen und Verschlüsselung
## Öffentlichen Schlüssel bereitstellen



john "exportiert" seinen öffentlichen Schlüssel:

```
> gpg --armor --export john@techfak.uni-bielefeld.de
```

und lädt ihn auf seinen Webserver, etc.

# E-Mail: Schwachstellen und Verschlüsselung

## Aufgaben für den Empfänger



1. Öffentlichen Schlüssel von john einholen
2. Fingerabdruck des Schlüssels prüfen
   (Gehört der Schlüssel wirklich john?)
3. Unterschriebene E-Mail öffnen

# E-Mail: Schwachstellen und Verschlüsselung

## Öffentlichen Schlüssel einholen



1. Datei johns_key.txt herunterladen
2. Johns Schlüssel aufnehmen:

```
gpg --import johns_key.txt
```

# E-Mail: Schwachstellen und Verschlüsselung

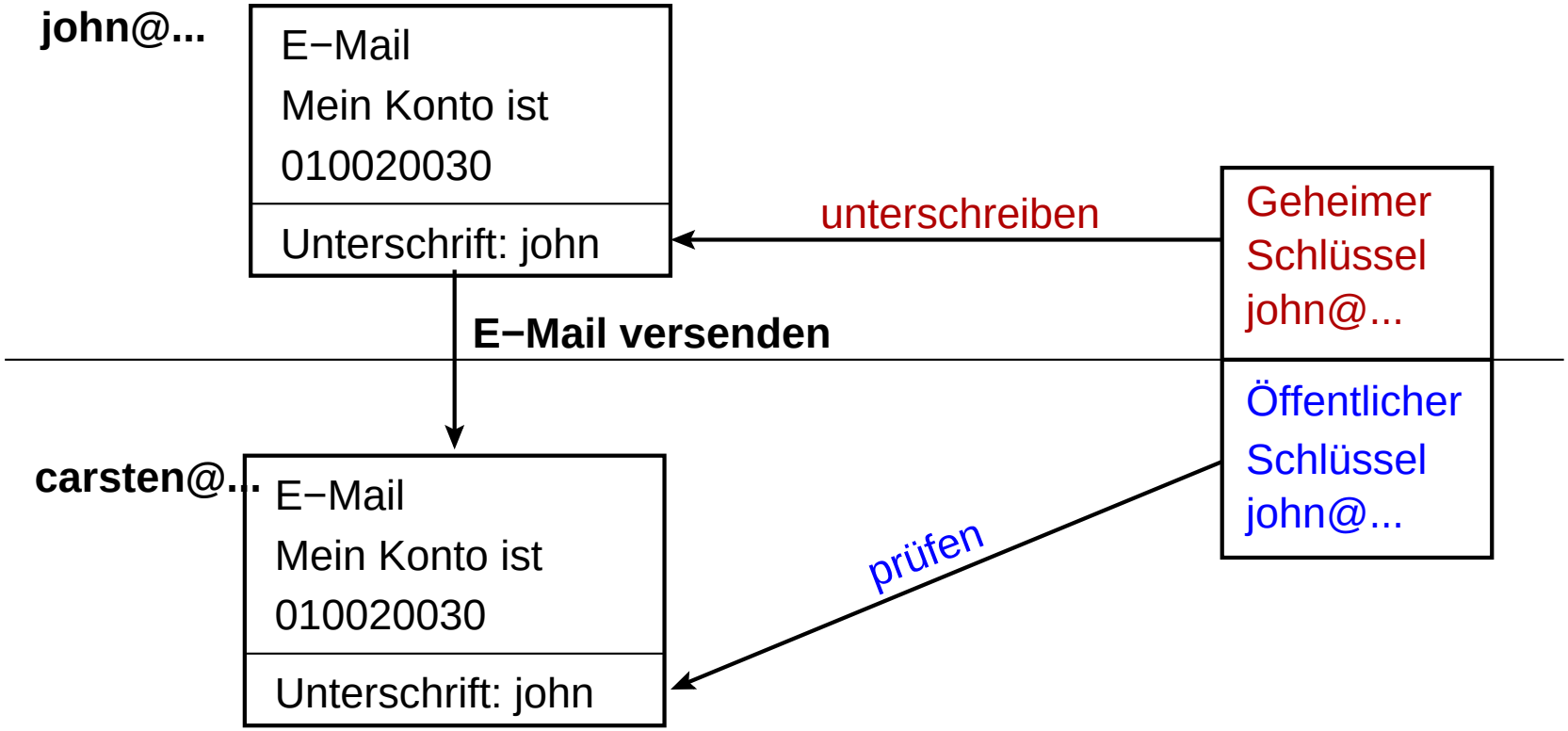
## Vorsicht: Ist das wirklich Johns Schlüssel?



# E-Mail: Schwachstellen und Verschlüsselung

## Folgendes Täuschungs-Szenario - Original-Mail



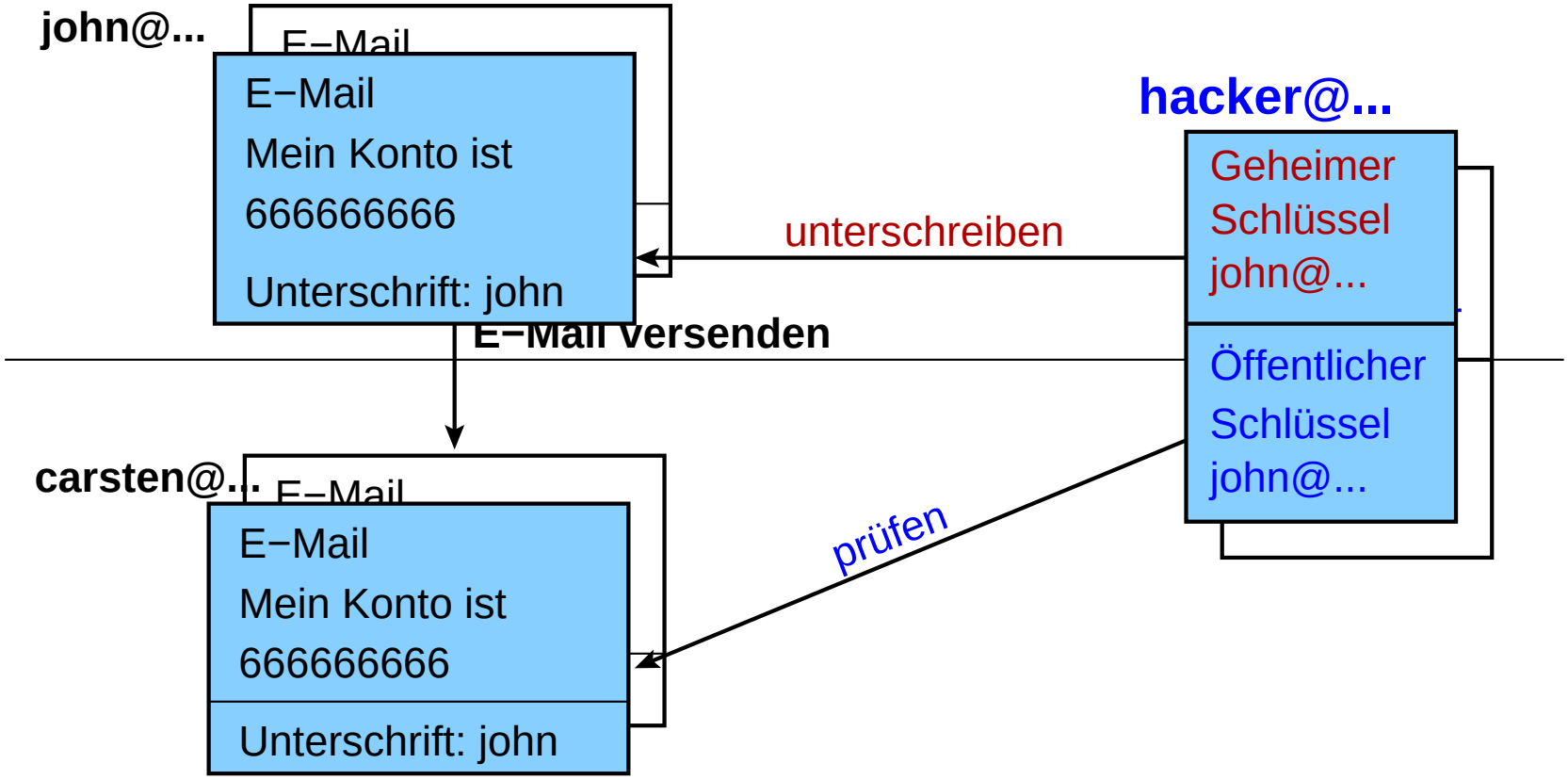



"Prüfen" = Kommt die Mail vom Besitzer des geheimen Schlüssels?

# E-Mail: Schwachstellen und Verschlüsselung

## Folgendes Täuschungs-Szenario - Gehackte Mail






"Prüfen" = Kommt die Mail vom Besitzer des geheimen Schlüssels?

# E-Mail: Schwachstellen und Verschlüsselung

## Überprüfen des Fingerabdruckes

Jeder Schlüssel hat einen eindeutigen, nicht fälschbaren Fingerabdruck:

```
> gpg --fingerprint tux@techfak.uni-bielefeld.de
pub 2048R/C677F222 2011-10-21
Schl.-Fingerabdruck = 19A1 FAFC E538 4E02 D04F
6651 B444 C7E3 C677 F222
uid tux user <tux@techfak.uni-bielefeld.de>
sub 2048R/74A2C40B 2011-10-21
```

# E-Mail: Schwachstellen und Verschlüsselung
## Überprüfen des Fingerabdruckes



Fingerabdruck von John persönlich bestätigen lassen:

- am Telefon vorlesen lassen
- von persönlich erhaltener Visitenkarte ablesen
- an ganz vielen Stellen im Netz nachlesen

# E-Mail: Schwachstellen und Verschlüsselung
## Überprüfen des Fingerabdruckes



Empfänger markiert überprüften Schlüssel als gültig:

(Voraussetzung: Empfänger (cg) hat auch ein Schlüsselpaar)

```
> gpg --sign-key john@techfak.uni-bielefeld.de
```

# E-Mail: Schwachstellen und Verschlüsselung

## Vertrauenswürdige E-Mail



# E-Mail: Schwachstellen und Verschlüsselung

## Verschlüsselung ist ein sehr komplexes Thema



zum weiteren Einlesen:

- Wikipedia, Stichwort GnuPG
- http://www.gnupg.org

was man auch wissen / ausprobieren sollte:

- Funktionsweise des “Web of Trust”
- komplettes Verschlüsseln von E-Mails

# Ende der heutigen Vorlesung



**Vielen Dank fürs Zuhören!**

**Viel Erfolg mit Eurem Studium!**