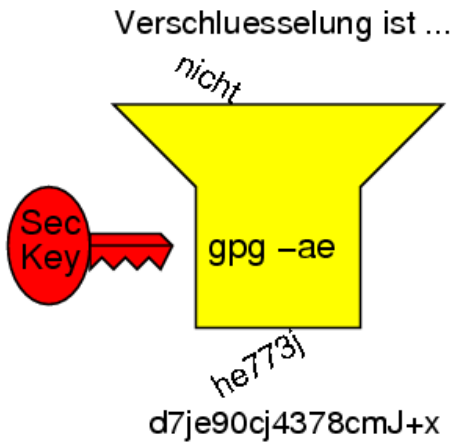
GnuPG, OpenSSL und Co

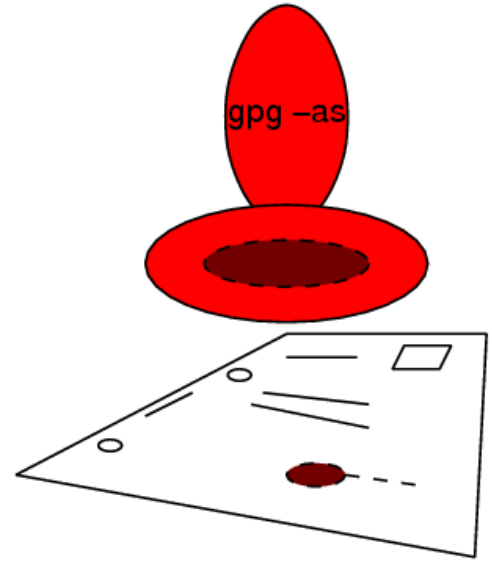
Verschlüsselung und elektronische Unterschrift

Joerg.Schulenburg-at-ovgu.de

2004-2014



TLS SSL TLS PGP hash MD5 SHA1 fingerprint RA CA PKI WoT RND private Key symmetrisch revoke ... ???



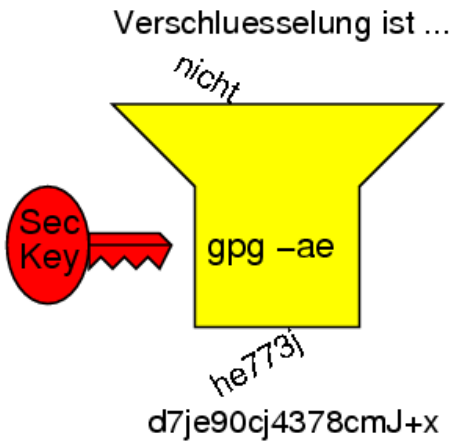
Inhalt
3 Abschnitte

- Motivation/Einführung (kurz)
- Theorie (ausführlich)
  - Verschlüsselung, Signierung, PKI, WoT, Sperrung, Schwachstellen
- Praxis (optional)
  - Protokolle: PGP, TLS, S/MIME, SSH, ...
  - Hardware: RdRand, eGK, nPA
  - Software: GnuPG, GnuTLS, OpenSSL, stunnel, OpenSSH, ...

Basiscs

Was ist Verschlüsselung? (kurz)

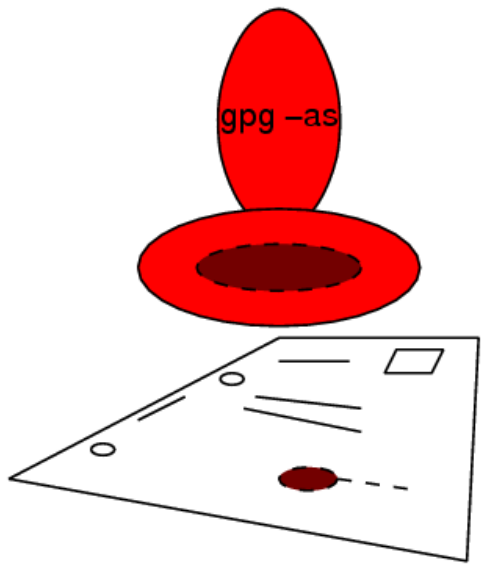
- Umwandlung von Klartext in Geheimtext
- mit dem Ziel, Klartext vor Unbefugten  zu verbergen



Basiscs

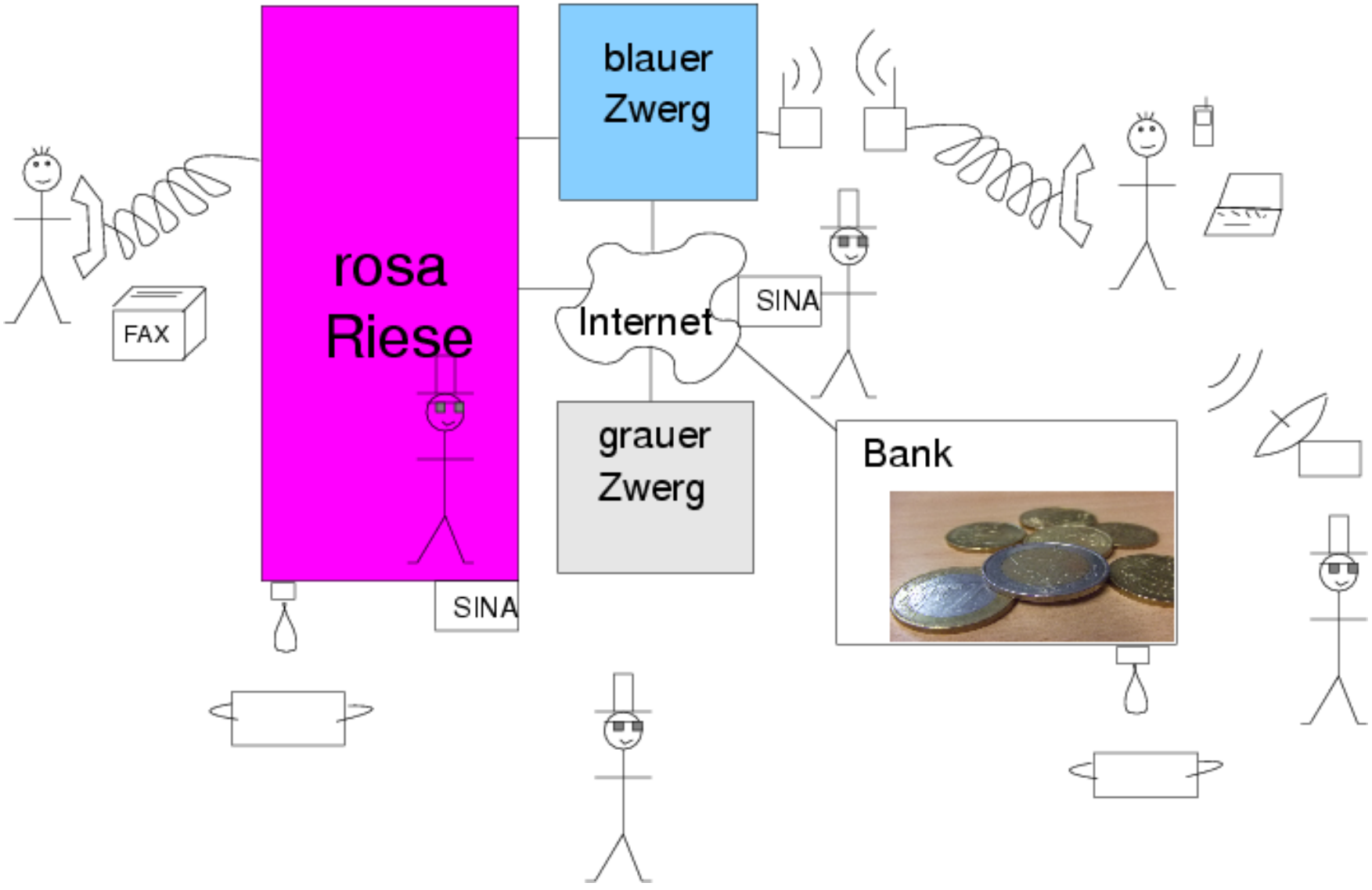
Was ist eine elektronische Signatur? (kurz)

- (elektron.) Ersatz für handgeschriebene Unterschrift (jurist.)
- lt. Wikipedia verschieden von “digitaler Signatur” (meinte ich wohl)
- digitales Anhängsel zur Prüfung von Urheberschaft und Zugehörigkeit



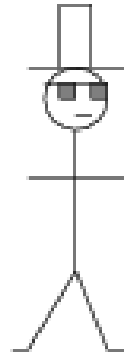
Wozu brauchen wir Crypto?

... unübersehbare Zahl von Missbrauchsrisiken



Wozu brauchen wir  Crypto?

theoretisch



- sichern gegen **mitlauschen** durch Dritte
- sichern gegen **Verfälschung**
- Verlässliche (Absender-)Identifizierung

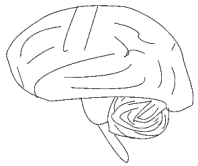
Wozu brauchen wir  Crypto?

praktisch

- WLANs: neugierige Nachbarn, Vermeidung Störerhaftung
- automat. Anmeldung ohne Passwort (ssh, ClientZert. GRID)
- bequeme (Geld-)Geschäfte via Internet (SSL, HBCI)
- (autom.) Download von signierten Programmpaketen
- vertrauliche EMAILs, ext. Backups (mit Kundendaten)
- signierte Rechnungen/Verträge per EMAIL
- Schutz bei Verlust der Hardware und mögl. Missbrauch
- Passwortersatz Userzertifikate (Browser,VPN)
- $(Ergänzungen?)

Wozu brauchen wir  Cryptowissen  ?

praktisch

- Nichts ist 100% sicher!
  (gilt für Crypto genauso wie für KKWs/AKWs)
- Wissen 
  verbessert Sicherheit mehr als ein grünes Schlosssymbol

in der Browserzeile

Ist Verschlüsselung etc. komplex?

- wenn man es richtig machen will, schon
  - schlechte ssh-Keys made by Debian systems (2008)
  - gehackte CAs und gefälschte Zertifikate (DigiNotar 2011)
  - rückgerufene Zertifikate (ungenutzt oder woher?)
  - selbstsignierte Zertifikate (Warnung des Browsers) = schlecht?
  - DFN (2007 TK,MS) und Moz.FF (2009 TK), versus Default RootCAs = gut?
  - Cross-Site-Attacks, Flash, Javascript
- Aber Wissen um Verschlüsselung und Co. bringt wie so oft Vorteile

Partner für GnuPG/SMIME?

- Banken (Idealanwender) sind lernresistent
  - SSL/RootCA Prüfsummen nur im Browser?
  - fordern Javascript für Webseiten, EMAILen im Klartext
  - signierte digitale Bankauszüge (wer kennts?)
- Telekom und Konsorten
  - Rechnungen per EMAIL (Klartext) aufgedrängelt (pos. Bsp?)
  - digital signiert? (positive Beispiele? = StratoDSL, ...)
- Compute-Server (ssh statt telnet durchgesetzt)
- ...
- Öfter mal nachfragen!

Jetzt wirds technischer ...

Theorie der Verschlüsselung

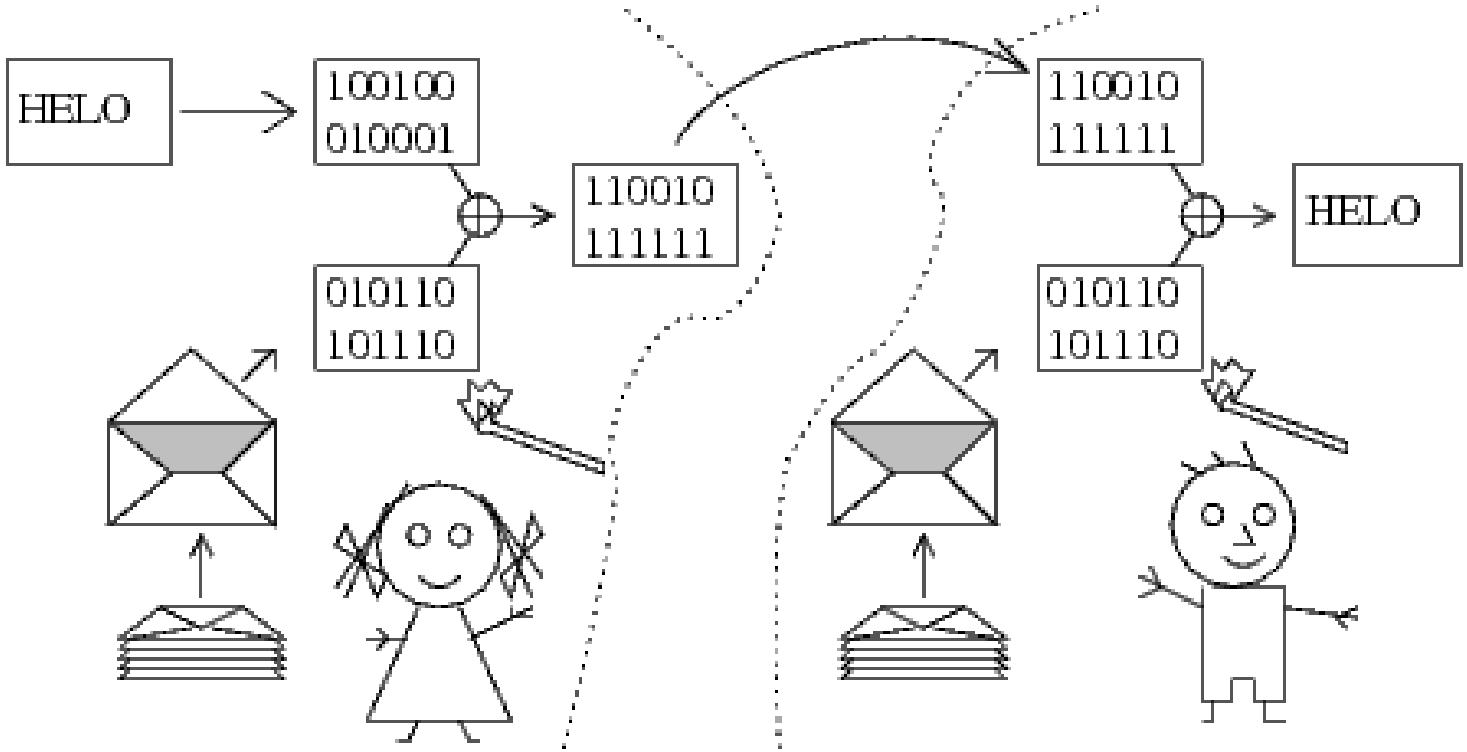
Theorie der Verschlüsselung
Zufall ist wichtig(ste Komponente)

- z.B. **gesalzene** Passwort-Hashes
- durch Salz keine Chance für Viren-Signaturen
- Komponente und Schwachstelle Nummer Eins (PRNGs)
- TRNGs fehlen oder sind langsam (D8-Würfel 3bit/Wurf)

```
gpg -a --gen-random 2 16 # 16 Bytes
openssl rand -base64 16  # [-engine padlock]
dd if=/dev/urandom bs=1 count=16 | base64 # 7MB/s
dd if=/dev/random  bs=1 count=16 | base64 # ..160kB/s
```

Wie funktioniert (theor.) Verschlüsselung?
One-Time Pad

- zufälliger Key, nur einmal zu nutzen!
- OTP ist einfach und bewiesen unbrechbar!



Wie funktioniert (theor.) Verschlüsselung?
One-Time Pad

Auch hier Fehlbedienung möglich:

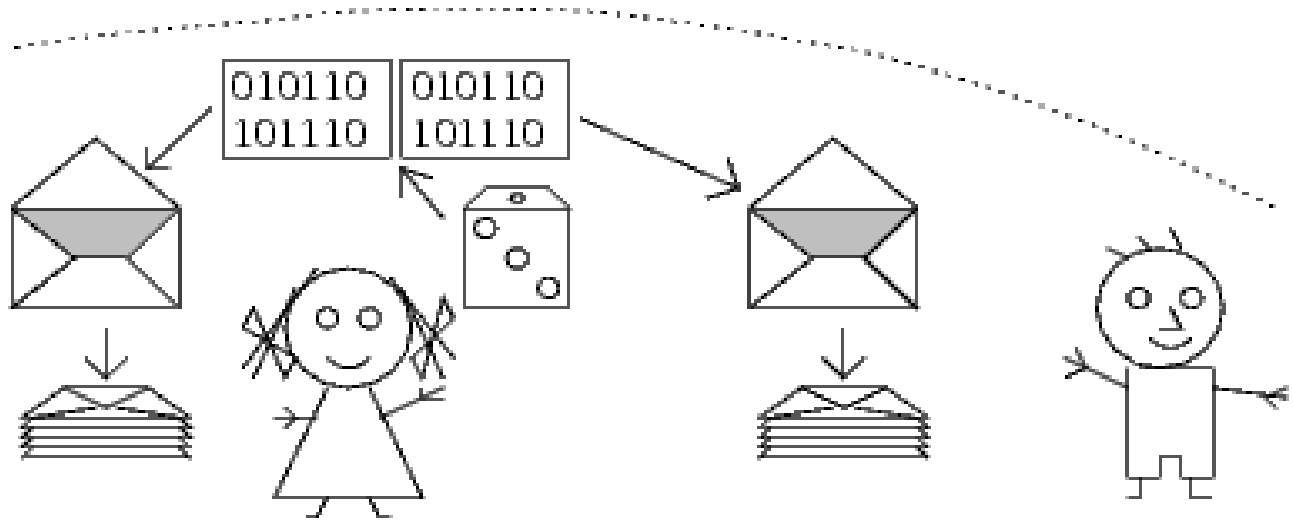
- bekannter Klartext an bestimmter Stelle verfälschbar
- z.B. xor Klartext xor Falschtext

Aber wichtigstes Problem ist das aufwändige Schlüsselmanagement

...

Wie funktioniert (theor.) Verschlüsselung?
One-Time Pad

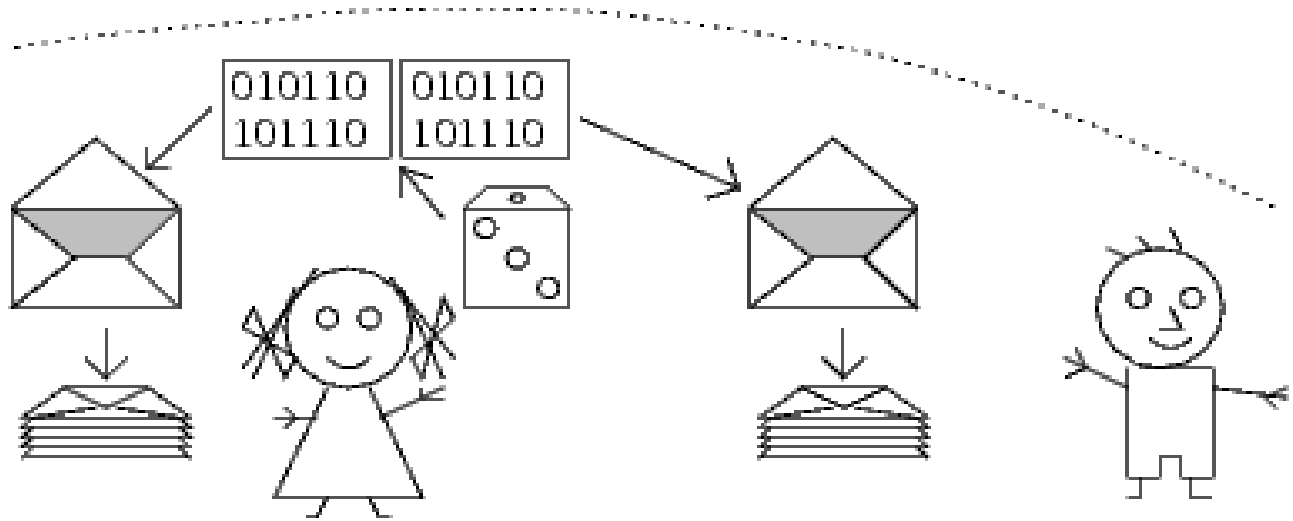
Problem: Key-Management



- echter Zufall, sicherer Transport und Aufbewahrung nötig
- Lsg: Quantenkryptographie (macht OTP praktischer, aber Skalierung(-))

Wie funktioniert (theor.) Verschlüsselung?
One-Time Pad

Problem: Key-Management



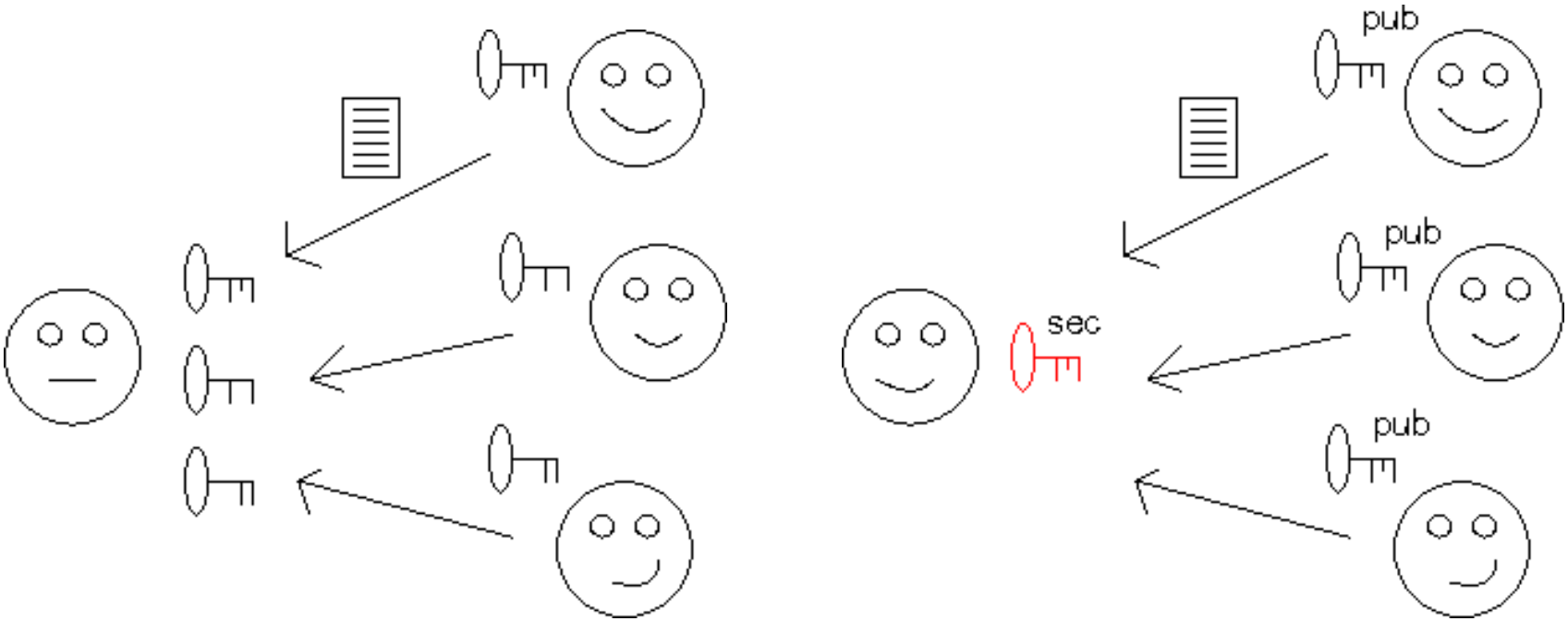
- echter Zufall, sicherer Transport und Aufbewahrung nötig
- Lsg: Quantenkryptographie (macht OTP praktischer, aber Skalierung(-))
- Kompromiss: Block-Cipher

Wie funktioniert (theor.) Verschlüsselung?
Block-Cipher

- symmetrisch (Blowfish, AES):
  - schnell
  - shared secret (paarweise, skaliert nicht)
  - Problem Schlüsselverteilung (n*(n-1))
- asymmetrisch (RSA, ELG, DSA):
  - langsam
  - "public key" = Primzahlprodukt, öffentlich (unfälschbar)
  - "secret key" = Primzahlpaar, geheim (Passphrase), secring.gpg
  - einfache Schlüsselverteilung, Rechenaufwand*1000
- hybrid (GnuPG, SMIME, ...)

Wie funktioniert (theor.) Verschlüsselung?

symmetrisch vs. asymetrisch



Wie funktioniert (theor.) Verschlüsselung?
symmetrisch

- zufälligen symmetrischen Key generieren
  und mit Passwort bzw. Passphrase verschlüsseln
  56-448 bit (= 57..150*D8)

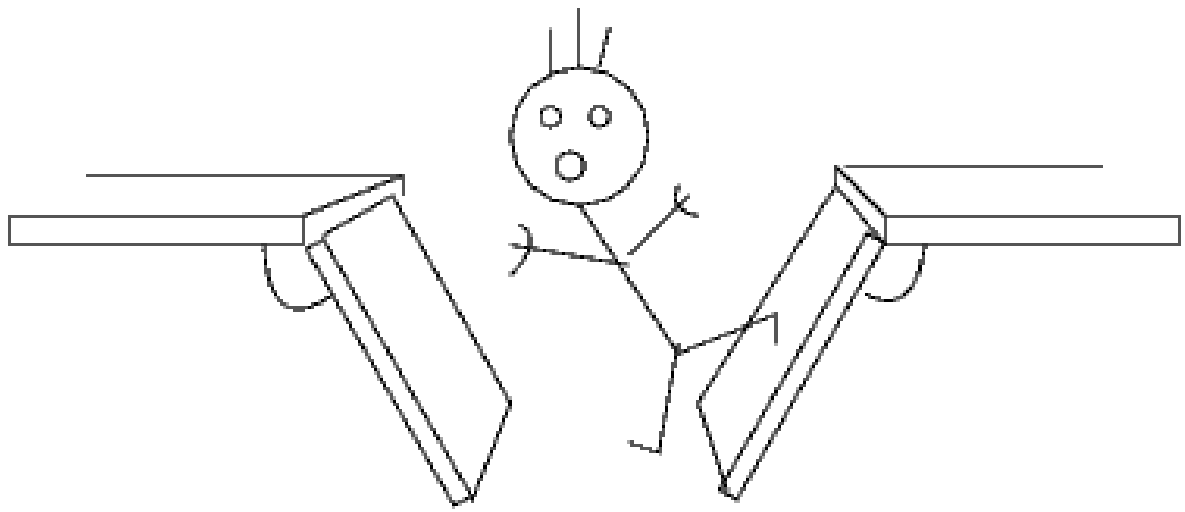
- komprimierung des Klartextes (zip, bzip2)
- symmetrisches verschlüsseln des Komprimates

z.B.: `tar -c path | gnupg -c path.tar.gpg`

Wie funktioniert (theor.) Verschlüsselung?
asymmetrisch

Falltürfunktionen (=asymmetrisch):

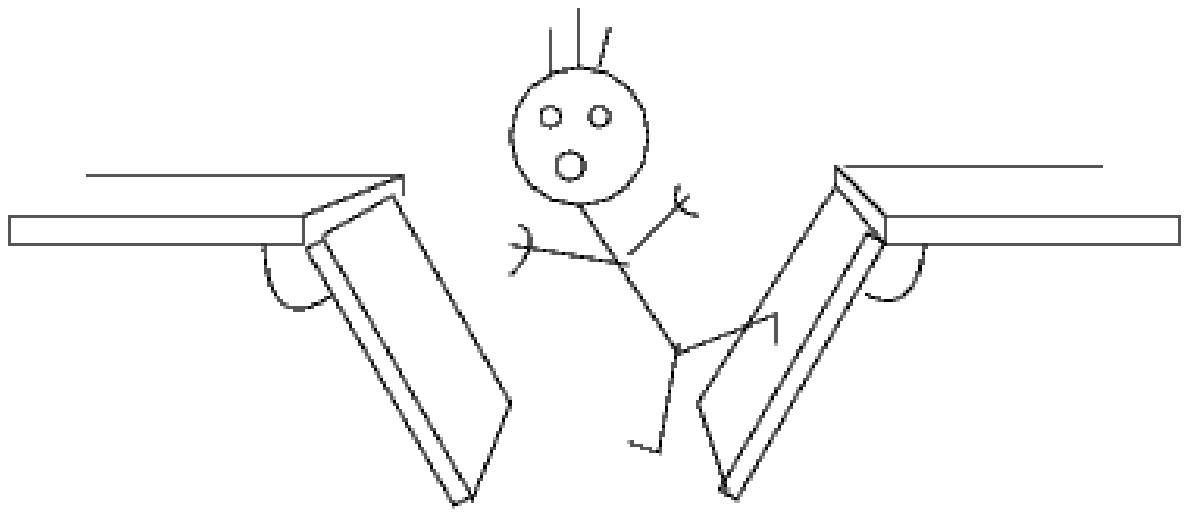
- Primzahlprodukte
- hineinfallen (privat key): 41*19=?



Wie funktioniert (theor.) Verschlüsselung?
asymmetrisch

Falltürfunktionen (=asymmetrisch):

- Primzahlprodukte
- hineinfallen (privat key): 41*19=?
- ... und wieder hinauskommen (public key): 713=?*?



Wie funktioniert (theor.) Verschlüsselung?
asymmetrisch, Primzahlen

```
gpg --gen-prime 1 16 #  2*11ms PM-600MHz
printf "%8x\n" $((0xA8FD*0xE4E9)) # 971b 2245

# brute force attack (26 zu 13bit, Faktor 96 (6.5bit))
for((x=3;$y % x;x+=2));do true;done;echo $x
y=7387       #   2ms 13bit
y=62615533   # 224ms 26bit
y=0x971b2245 #  1.4s 32bit
```

Wie funktioniert (theor.) Verschlüsselung?
asymmetrisch, Primzahlen

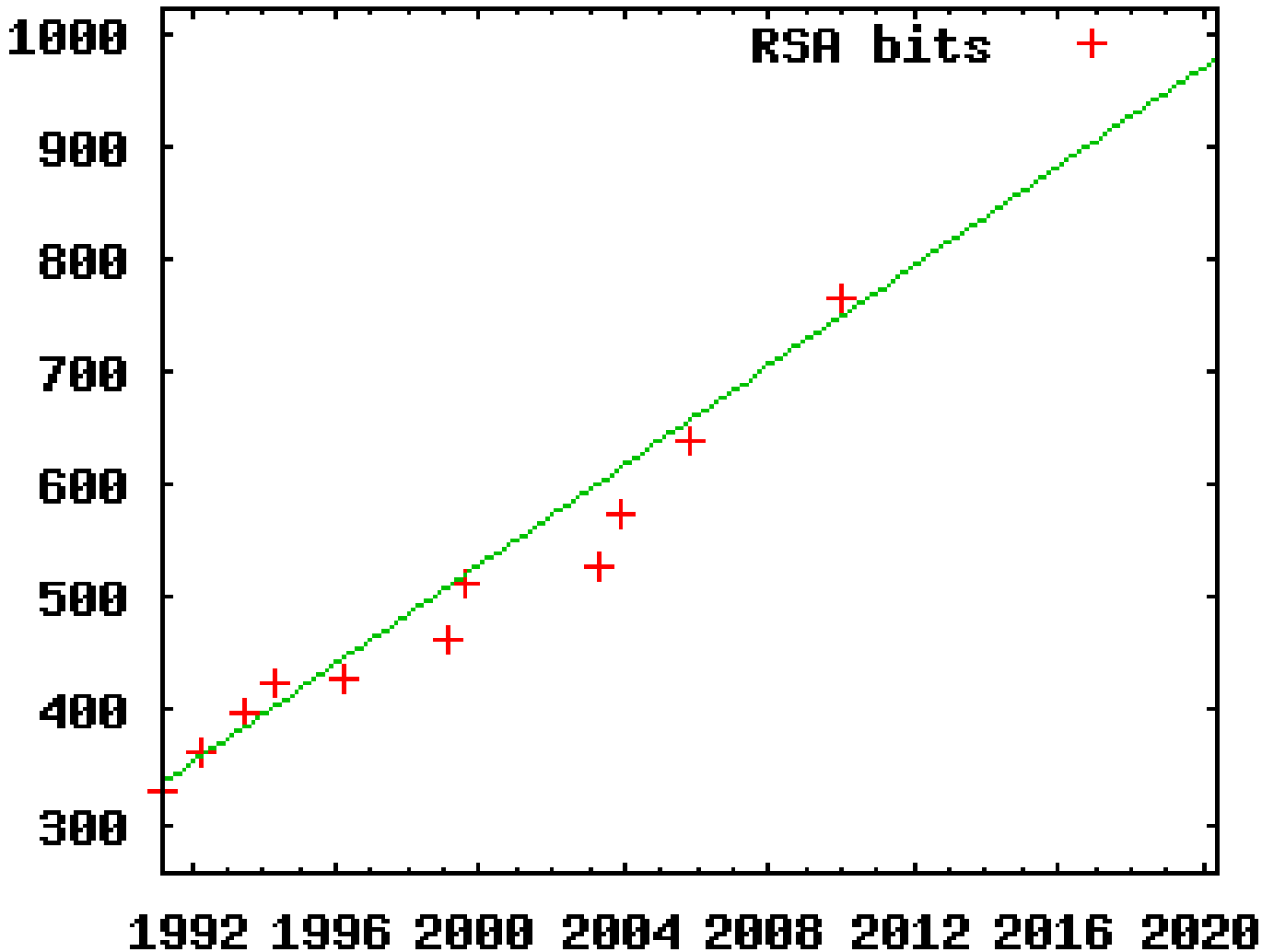
- Primzahlgenerierung:
  - Zufaellige Zahl erwürfelt 
  - Test auf Primzahl (Miller-Rabin-Test nutzt Zufall)
    
  - deshalb guter Zufall benötigt!

Wie funktioniert (theor.) Verschlüsselung?
asymmetrisch, Primzahlen, Verschleiss

- Public Keys und Hash-Algos altern!
- asym. Verschlüsselung, Signaturen, Zertifikate unterliegen daher zeitlichen Verschleiß!
- vs. Unterschriften auf Papier

Wie funktioniert (theor.) Verschlüsselung?

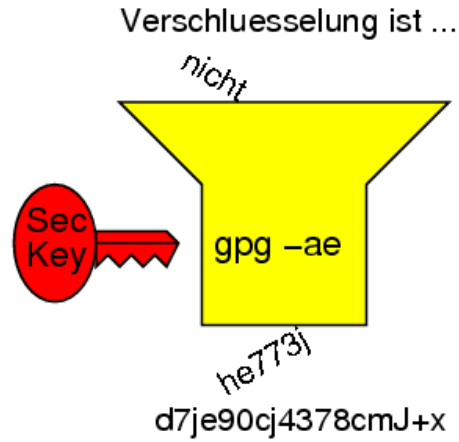
asymmetrisch, Primzahlen, RSA-Challenge



Wie funktioniert (theor.) Verschlüsselung?
asymmetrisch

- zufälligen symmetrischen Key generieren
  und mit Public-Key verschlüsseln

- Komprimierung des Klartextes (zip, bzip2)
- symmetrisches verschlüsseln des Komprimates
  (z.B.: `tar -c path | gnupg -e path.tar.gpg`)



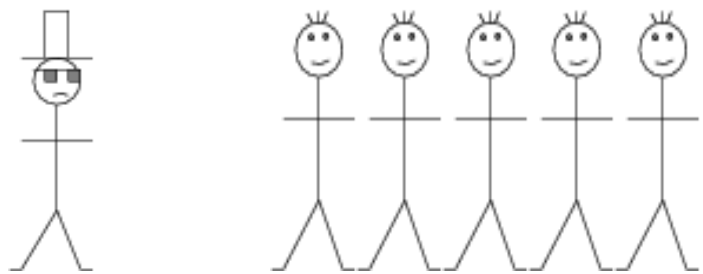
Wie funktioniert (theor.) Entschlüsselung?
asymmetrisch

- symmetrischen Key
  mit mit Secret-Key entschlüsseln

- symmetrisches entschlüsseln des Komprimates
- Dekomprimierung des Klartextes (unzip, bunzip2)
  (z.B.: `gnupg < path.tar.gpg | tar -x`)

Wie funktioniert Schlüsselverteilung?
asymmetrisch

- PUBLIC == jeder darf Key sehen
- PUBLIC == Fälschung muss unmöglich sein
- public, im Sinne von “broadcast” (Rundfunk)
  oder PGP Keyserver
- Signiert durch vertrauensvolle Leute (Web of Trust)
  oder Offizielle (CAs) und deren Prüfern (RAs)
- lokale Kopien: pubring.gpg + trustdb.gpg



Fingerabdruck?

- Hash aus Public-Key generieren
  - MD5 128bit 1991-1996 safe, 2008 full broken
  - SHA1 160bit 1995-2005 safe,
    MS-Policy2013: no SHA1-SSL after 2017 accepted
  - SHA2/x 224-512bit 2005 safe (2014)
  - SHA3/x 224-512bit 2012
- zur Prüfung der korrekten Übertragung

```
openssl dgst -md5|-md4|-md2|-sha1|-sha|-mdc2|...|-dss1 # 0.9.8e
gpg --print-mds # print message digest, version 1.4.5
# Hash: MD5 SHA1 ... SHA256 SHA384 SHA512 SHA224
```

Fingerabdruck/Hash

- zur Prüfung der korrekten Übertragung
- vs. Faulheit des Menschen (CA,RA)

```
gpg --fingerprint Joerg # 1024D (256 vs. 40 hexdigets SHA1)
 3816 B803 D578 F5AD 12FD  FE06 5D33 0C49 53BD FBE3
```

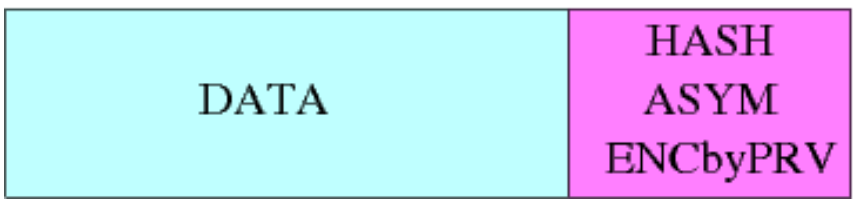
```
ssh-keygen -f test # ssh-5.1 17x9 randomart image
+--[ RSA 2048]----+
|                 |
|      .          |
|   . o .         |
|    + . .        |
|   o.=   S       |
|  ooE .          |
| .o*+o           |
|=.+oo            |
|=o.o             |
+-----------------+
```

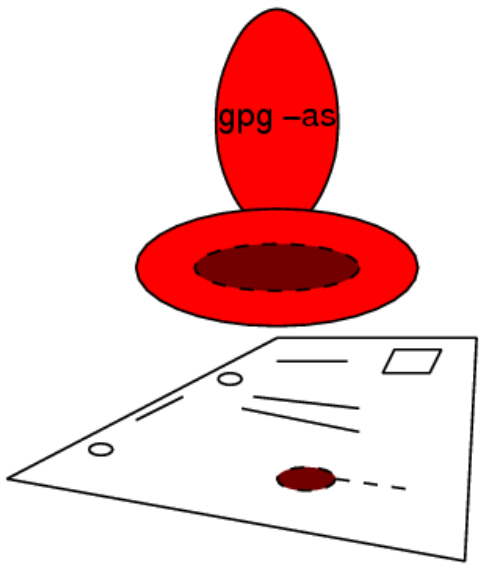
digitale Signatur

- Prüfbarkeit der Verbindung Vertragstext mit Unterzeichnern
- Vertrauen in schwerer Fälschbarkeit des Vertrages/Urkunde
  - Papier, Tinte und Handschrift (seit Jahrhunderten für Jh.)
  - schwieriger mit Stempeln, Druckern, Kopierern (aber Physik hilft)
  - Daten, Computer, Primzahlen und Wodoo-Mathematik (???)
- ohne physikalische Bindung wesentlich verschieden
- Fälschungsversuche schnell/billig, schwer nachweisbar
- Vertrauenswürdigkeit der Hardware schwer zu sichern

Wie funktioniert die digitale Signatur?

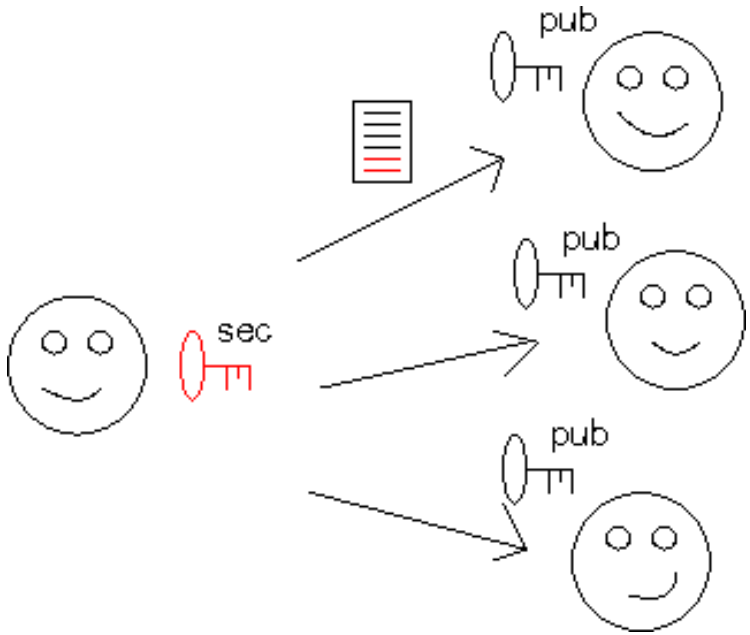
- Hash aus Klartext generieren (MD5, SHA1)
- Hash mit Secret-Key verschlüsseln





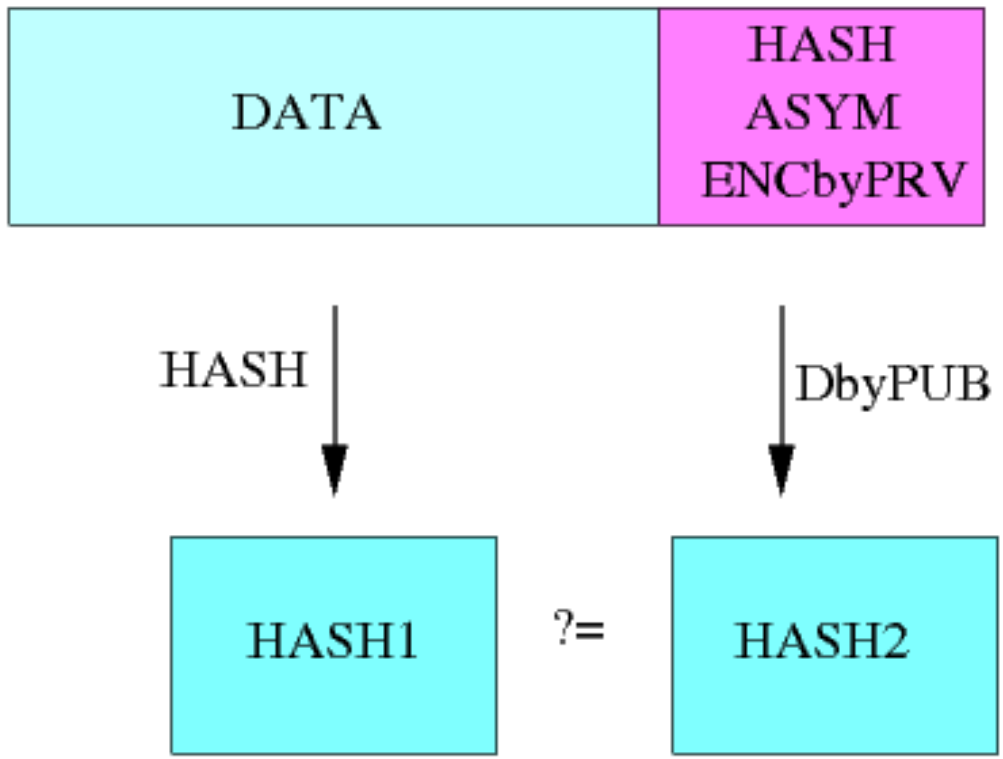
Wie funktioniert die digitale Signatur?

- jeder, der Pubkey kennt, kann prüfen



Wie funktioniert die digitale Signatur?

- Prüfung: Hash entschlüsseln mit PubKey
- ... mit selbst berechneten Hash vergleichen



Was sind Zertifikate?
PKI, Zertifikate

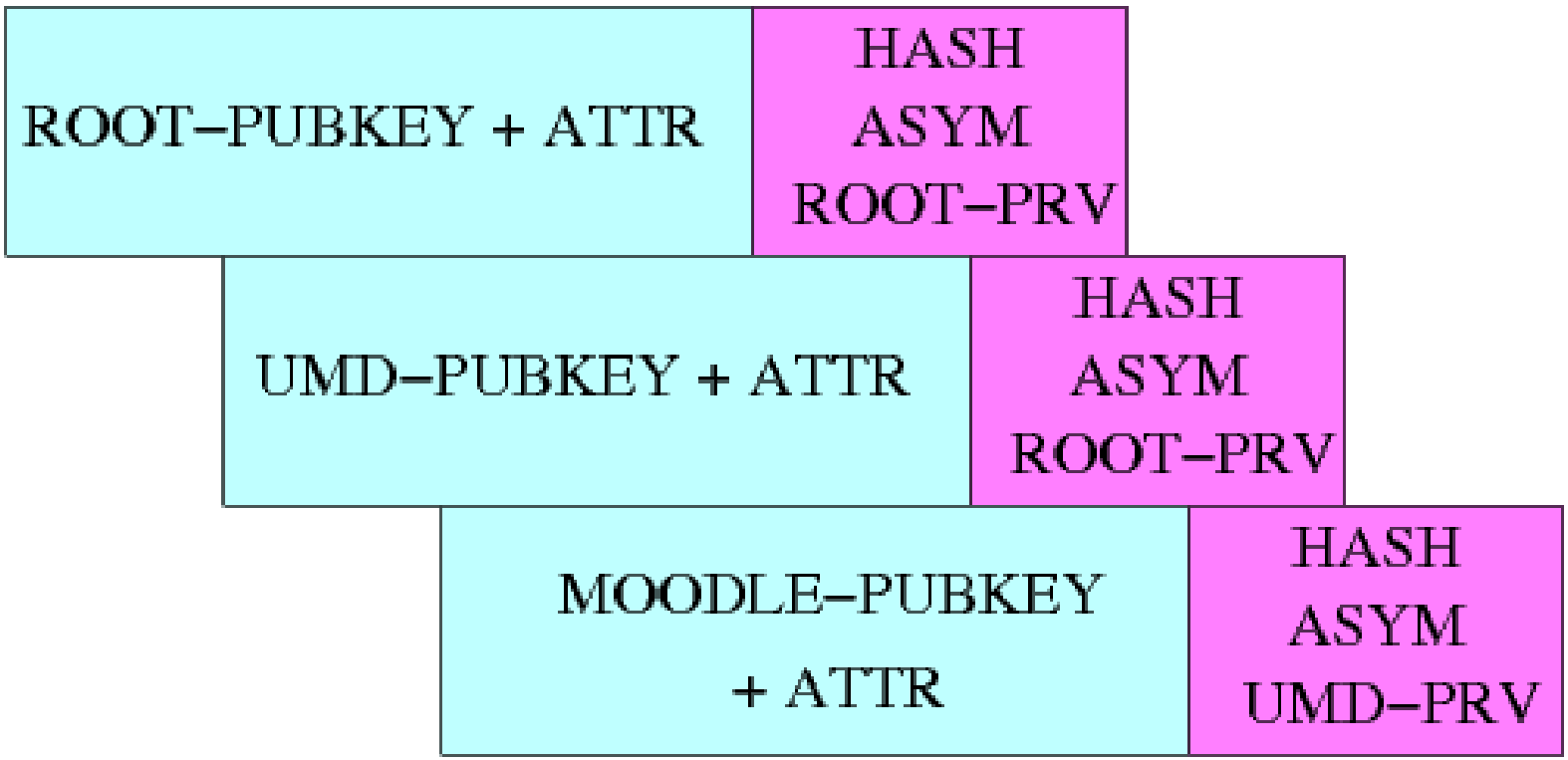
- “offizielle” signierte Public-Keys
- zentral + staatlich geadelt (aber technisch nicht besser als OpenPGP)
- evl. + WasManDamitMachenDarf-Attribute (=Text), z.B. x509

- aber: “offiziell” $\neq$ sicherer

Was sind Zertifikate?
x509 - Ketten

- RootCAs signieren Zertifikate von SubCAs
- Zertifikat enthält Attribut für “darf CA-Cs ausstellen”
- SubCAs können SubsubCAs haben bis Attribut fehlt
- Kettenaufbau in “Policies” beschrieben
- alle CAs können Zertifikate ausstellen, wenn Policy das erlaubt
- Regelverstöße möglich (i.d.R. nachweisbar, rechtliche Wirkung)

Was sind Zertifikate?
x509 - Ketten



Zertifikate
Sperrung/Widerruf

... was wenn private Key geklaut?

- Notbehelf, wenn passiert was nicht sein darf
- Rückruf/Sperrung/Widerruf/Ablauf
- Veröffentlichung einer signierten Sperrliste
- technisch ein zweiter Vertrag
- bei PGP als selbst signiertes Widerrufszertifikat
- bei TLS bei CA melden, generiert Liste (Link im Cert.)

PKI
Publik-Key-Telefonbuch

Vorteil: keine PubKey-Geheimhaltung nötig

- allgemein verfügbar, da nicht geheim
- zentrale oder dezentrale Vertrauensinstanzen (Web-Of-Trust)
- Fälschungsschutz mit Unlöschbarkeit im PGP-Servernetz

PKI

sehr nützlich, aber

auch Schattenseiten (gibts immer)

- Schlüssel mit gleichen Nummern oder Namen
- Leute oder CAs, die alles unterschreiben
- PGP-Server + SPAM für PGP-Nutzer (durch Unterschriften)
  z.B. PGP-Keyserver http://pgpkeys.pca.dfn.de/
  down seit Anfang 2011
  Grund: beleidigende User-IDs! + konkrete Beschwerden +
  20J. altes Unlöschbarkeitskonzept
- Bsp: x-hkp://gpg-keyserver.de/
  *rsch 14 Treffer! 1997..2010 (verfallen nicht)
  “politverbrecher” 1 Treffer
- generelles Problem: einzelne kippen ganze Netze (pgp,tor,...?)

Was geht nicht?

Verschlüsselung ...

- nur Inhalte sind verschlüsselt
- aber wer mailt wann und wem, bleibt sichtbar
- Verweis: Steganographie etc. (eigenes Thema)
  - ablegen verschlüsselter Texte in Boxen ...
  - Signatur mitverschlüsseln,
  - Keys ohne echte Namen/Adressen nutzen
  - Vorbild: Botnetze

Schwachstellen?

- grundsätzlich fehlender mathematischer Beweis der Sicherheit
- Hash-Kollisionen (Paargenerierungen, Zufallserweiterungen etc.)
- kurze oder schwache Schlüssel (versus Rechen-/Speicheraufwand, schlechte Zufallsgeneratoren)
- DSA (ElGamal-PK) liefert PrivKey aus Sig bei schlechten Zufall
- gefälschte Public-Keys (mangelnde Prüfung, mangelhafte Öffentlichkeit z.B.: CAs)
- Geheimhaltung des Secret Keys (Viren, Trojaner, Backdoors, NFS, CPU-Abstrahlung)
- mangelnde Implementierungen (schlechter Zufallsgenerator)
- ungesichertes Endgerät (eigenes Kapitel)
- ...

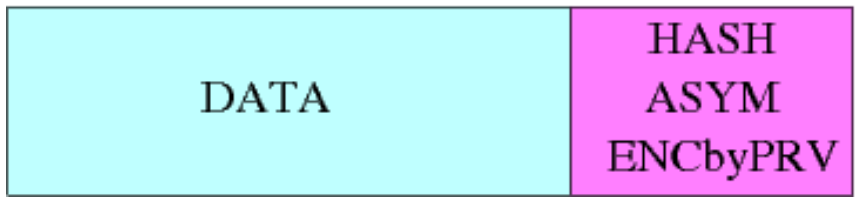
Schwachstellen?
Unsicheres Endgerät

- in der Regel PC (Trojaner, Viren, Hacker, Bugs, ...)
- aber auch FW-Update-Funktionen in Lesegeräten
- nPA + C1/C2-Leser bei unsicheren PC = Blanko-Unterschrift
- HBCI + Class3-Leser bei Gruppenaufträgen
- Nutzung von Fremd-Endgeräten verbreitet
- Aber wer schleppt sein eignenen EC-Kartenleser zum shoppen mit sich rum?

Theorie
Zusammenfassung

- Asym. (mixed) Verschlüsselung

- el. Signatur



- el. Zertifikat

OK, legen wir los!

Welche Hardware?

- jedes vertrauenswürdige Endgerät mit CPU
- mit bekannten Chips ohne verborgene Funktionalität (Keylogger, manipulierte FW von PCI-Netzwerkkarten)
- ideal separates Gerät (ohne andere Aufgaben)
- ideal sparsam und portabel (wenig CPU-Abstrahlung)
- eigene EMAIL-Anbindung oder Bluethooth/WLAN/USB
- besser alten Laptop als High-End-Game-PC, Handy, ...
- ... EC-Bezahlterminal im Shop eher kein sicheres Endgerät
- ... HBCI oder ePerso (nPA) + ClassX-Leser zu Hause oder ...
- ... der PC im Internetcaffee auch nicht

Welche Hardware?

- mit schneller echter Zufallsquelle (TRNG)
- ... eher mau: VIA C3, i810/815/840/845G
  (TRNGs auch für Monte-Carlo-Rechnungen sinnvoll,
  TM $\longrightarrow$ PTM)

$= 2 \ldots 3$ bit/s

$= 4$ bit/s

$=$ Via C3 $\approx$ 2Mb/s

```
gpg -a --gen-random 2 1
```

Welches Betriebssystem?

- jede vertrauenswürdige OS
- d.h. nur Software mit bekannter Funktionalität
- ... installiert mit signierten Paketen
- ... in Praxis: BSD, Linux, ReacOS, FreeDOS
- ohne automatische Updates!
  ... sonst Manipulationen über Distributer jederzeit möglich
  ... ideal deshalb alter Linuxlaptop nur zum verschlüsseln
- zusätzliche Kontrolle via iptables -nvL oder tcpdump
  ... check gegen nach-Hause-telefonierende Programme (Auto-Updates)

Welche Programme?

- no Closed Source! (Hintertüren, Generalschlüssel, Fehler)
- keine automatischen (unkontrollierte) Updates
- OpenSource? (Ja! Nur! OSS = Transparenz = Vertrauen)
  - PGP? (das Original? Patente!)
  - GnuPG? (ja, bewährtes universelles Jedermannsprogramm = verbreitet)
  - OpenSSL, S/MIME? (ja, aber hierarchische PKI)
  - GnuPG + OpenSSL Library basierende ...
  - und wenns nötig ist, grafische Oberfläche dazu :)
  - ...

Welche Programme?

- www.GnuPG.org (geht immer, als Backend bewährt 7MB)
- Linux: gnupg meist enthalten (für Paketmanager), GPA
- Windows: Cygwin, GPG4Win (GnuPG+GPA+OL+EX+... 38MB)

Welche Programme?

- Mailclient: Sylpheed für Linux+Windows+BSD, GnuPG integriert
  (aber Key-Management separat)
- Mailclient: Thunderbird + Plugin EnigMail (Win+Lx+BSD)
- am einfachsten mutt (textclient, auch Win+Lx+...)
- via vi `:%!gpg -aer 0x53BDFBE3`
- Plugins für Outlook-Express, LiveMail (ungetestet)

Welche Programme?

- Mailclient: Sylpheed für Linux+Windows+BSD, GnuPG integriert
  (aber Key-Management separat)
- Mailclient: Thunderbird + Plugin EnigMail (Win+Lx+BSD)
- am einfachsten mutt (textclient, auch Win+Lx+...)
- via vi `:%!gpg -aer 0x53BDFBE3`
- Plugins für Outlook-Express, LiveMail (ungetestet)
- ACHTUNG: encrypted send kann man nicht selbst lesen!

Welche Programme?
SMIME?

- Viele Mailclients haben SMIME integriert:
  (Outlook, Live-Mail, Thunderbird, ...)
- Woher Key? Wo liegt Key? ... oft mangelnde Hilfe
- Hilfe: “Wenden Sie sich an Ihren Administrator!”
- Zertifikatshändling viel unflexibler (oft ein K(r)ampf)
- ... eher für Firmenstrukturen gedacht?
- ... und auch nicht sicherer als GnuPG (andere Aspekte)
- irreführende Warnmeldungen:
- z.B.: mails mit Public-Key = “gefährlicher Inhalt” (LiveMail)
  erst mit vertrauenswuerdigen importierten root-cert OK
- z.B.: signiert mit unbekannter/abweichender Signatur ==
  “gefaehrlicher Inhalt”
- SMIME-Zertifikatsimporte erfordern wahre Klick-Orgien

GnuPG
als CLI

Motivation: fall-back, universell, Text statt viele Screenshots ;)

- Schlüsselpaar generieren (`--gen-key`, `--list-key`)
  Verzeichnis: .gnupg/secring.gpg + pubring.gpg + ...
- Public-Key ex- und importieren (`--export -a`, `--import`)
- Public-Key prüfen (`--fingerprint`)
- Keyserver: (`--search-keys`
  `--send-keys --keyserver hkp://www.keyserver.net`)
- Signatur erzeugen/prüfen (`-a --sign`, `--verify`)
- Datei ver-/entschlüsseln (`-ae`, `-d`)

GnuPG
symmetrisch

Encrypted files (per mail) and shared passphrase (personal)

- `echo test2 | gpg -c -v`
  Geben Sie die Passphrase ein:
  gpg: benutze Cipher CAST5
  15B-Header-incl-8B-Salz + pw-hash-enc-rnd-symkey + symkey-enc-compr-data
- `echo test3 | gpg -c --s2k-mode 0 --compress-algo none --passphrase xxx`
  gpg: Hinweis: Vom“simple S2K”-Modus (0) ist strikt abzuraten
  8B-header + 16B-pw-enc-rnd-symkey + enc-data+2B

OpenSSL
symmetrisch

Encrypted files (per mail) and shared passphrase (personal)

- `echo test|openssl enc -e -k xxx -cast5-cfb`
  16B-Header“Salted”-incl-8B-Salz + saltedpw-enc-data
- `echo test|openssl enc -nosalt -e -a -k xxx -cast5-cfb`
  pw-enc-data
- Nutzer muss selbst komprimieren

Zertifikatskette holen und anzeigen

```
openssl s_client -connect www.ovgu.de:443 -showcerts
Certificate chain
 0 s:/CN=www1.ovgu.de
   i:/CN=www1.ovgu.de
-----BEGIN CERTIFICATE-----
...
New, TLSv1/SSLv3, Cipher is DHE-RSA-AES256-SHA
Server public key is 1024 bit
Compression: NONE
...
```

OpenSSL

Userzertifikat (siehe ca.ovgu.de)

User weißt sich mit Zertifikat aus (kein Passwort nötig)

- `openssl s_client -connect www.ovgu.de:443 -cert mycert.pem`
- mit stunnel für eigene non-SSL-Apps aufrüsten

Und nun ...?

- ... learning by doing
- ausprobieren und verbreiten!
- viele Crypto-Erweiterungen nicht nur für EMAIL (Off-the-Record, jabber (OTR), ..., PubKey-Logins)

Beispiel aus dem Leben

grml debian apt-get

```
sudo apt-get update
 W: GPG error: http://cdn.debian.net sid Release:
 The following signatures couldn't be verified because the
 public key is not available: NO_PUBKEY AED4B06F473041FA

gpg --search-keys 0xAED4B06F473041FA
 gpg: searching for "..." from hkp server keys.gnupg.net
 (1) Debian Archive Automatic Signing Key (6.0/squeeze) ftp...
           4096 bit RSA key 473041FA, created: 2010-08-27
 Keys 1-1 of 1 for "0xAED4B06F473041FA".  Enter ... > 1
 gpg: key 473041FA: public key "Debian Archive Automatic
  Signing Key (6.0/squeeze) <ftpmaster@debian.org>" imported
 gpg: no ultimately trusted keys found

gpg --list-sig 0xAED4B06F473041FA  # 11 Signaturen
```

Beispiel aus dem Leben ...

grml debian apt-get

```
# dem System bekannte Signatur?
sudo aptitude install debian-archive-keyring # Warnt, aber ...
 * Add Debian Archive Automatic Signing Key (6.0/squeeze) (ID: 473041FA
 * Convert keyring generation to jetring.
 * Add Squeeze Stable Release Key (ID: B98321F9).  (Closes: #540890)
 * Add a DEBIAN/md5sums file to the non-udeb package.  (Closes: #534934
 * Move to debian-archive-removed-keys.gpg:
   - Debian Archive Automatic Signing Key (4.0/etch)
   - Etch Stable Release Key
   - Debian-Volatile Archive Automatic Signing Key (4.0/etch)
```

- Keine Infos über Vertrauen/Signaturen :(
- Handarbeit bzw. Verbesserungspotential
- ... unbedarfte nPA-Nutzung kann interessant werden

Tips

- mehrere zweckgebundene Keys verwenden
- tägliches nutzen (Übungskey)
- vor “Ernstfall” nochmal informieren
- nicht alle Ratschläge stur befolgen
- keiner Werbung glauben
- Zertifikat ist nicht gleich Zertifikat

Quellen:

- www.ovgu.de/jschulen/
- www.wikipedia.de (Verschlüsselung, GnuPG, PGP, ...)
- J.M. Ashley, GNU Privacy Handbuch (GPH als PDF-Datei)
- T. Bader, Geheimsache, Linux-Magazin 12/1999
- C. Kirsch, Mailchiffrierung mit GnuPG, iX 3/2004
- www.dfn-pca.de
- www.sicherheit-im-internet.de
- en.wikipedia.org/wiki/VIA_C3#cite_ref-2
  "Evaluation of Via C3 Nemehemiah Random Number Generator", 2003
- u.a.

Danke!