

Crypto Campaign

E-Mail-Verschlüsselung mit GnuPG

Lennart Braun
<3braun@informatik...>

8. November 2016

Warum verschlüsseln?

Asymmetrische Kryptographie

RSA

GnuPG

Warum sollte man seine E-Mails verschlüsseln?

- ▶ E-Mails liegen normalerweise im Plaintext auf Servern, die kann jeder mit Zugriff lesen.
 - ▶ neugierige Administratoren
 - ▶ Geheimdienste
 - ▶ Algorithmen
- ▶ zum Schutze anderer
 - ▶ Schutzbedürftige fallen weniger auf, wenn mehr Leute verschlüsseln.
- ▶ Weil es geht.

Warum sollte man seine E-Mails verschlüsseln?

Ich bin kein Terrorist, ich habe doch nichts zu verbergen!

Arguing that you don't care about the right to privacy because you have nothing to hide is no different than saying you don't care about free speech because you have nothing to say.

Edward Snowden

Warum sollte man seine E-Mails verschlüsseln?

Es gibt doch TLS ...

- ▶ TLS bietet nur Transportverschlüsselung
- ▶ Auf den Servern liegt die E-Mail wieder im Klartext vor
- ▶ Man kann sich nicht darauf verlassen, dass alle Mailserver TLS sprechen

Wir wollen Ende-zu-Ende-Verschlüsselung!

- ▶ Nur Absender und Empfänger sollen den Inhalt lesen können.

Was möchten wir haben?

Vertraulichkeit

- ▶ Nur Sender und Empfänger sollen die Nachricht lesen können.

Integrität

- ▶ Niemand soll unbemerkt die Nachricht verändern können.

Zurechenbarkeit

- ▶ Wir möchten sicher sein, dass die Nachricht von dem richtigen Absender stammt.

Was möchten wir haben?

Vertraulichkeit

- ▶ Nur Sender und Empfänger sollen die Nachricht lesen können.

Integrität

- ▶ Niemand soll unbemerkt die Nachricht verändern können.

Zurechenbarkeit

- ▶ Wir möchten sicher sein, dass die Nachricht von dem richtigen Absender stammt.

Wichtig!

Die hier vorgestellten Verfahren schützen nur den Inhalt der Nachricht. Die Metadaten bleiben weiterhin sichtbar.

Asymmetrische Kryptographie

Symmetrische Kryptographie:

- ▶ Sender und Empfänger besitzen einen geheimen Schlüssel
- ▶ Nachteil: $\frac{n(n-1)}{2}$ Schlüssel für n Personen
- ▶ Vorteil: schnell

Asymmetrische Kryptographie

Symmetrische Kryptographie:

- ▶ Sender und Empfänger besitzen einen geheimen Schlüssel
- ▶ Nachteil: $\frac{n(n-1)}{2}$ Schlüssel für n Personen
- ▶ Vorteil: schnell

Asymmetrische Kryptographie:

- ▶ Jeder besitzt ein Schlüsselpaar bestehend aus öffentlichem und privatem Schlüssel
- ▶ Nur n Schlüsselpaare für n Personen
- ▶ Nachteil: weniger schnell

⇒ hybride Verschlüsselung

Asymmetrische Verschlüsselung



Bob hat ein Schlüsselpaar bestehend aus einem öffentlichen Schlüssel (gelb) und einen privaten Schlüssel (grün) erstellt.

Asymmetrische Verschlüsselung



Alice bezieht Bobs öffentlichen Schlüssel.

Asymmetrische Verschlüsselung



Alice erstellt ein Dokument, das verschlüsselt werden soll.

Asymmetrische Verschlüsselung



Alice wendet Bobs öffentlichen Schlüssel auf das Dokument an.

Asymmetrische Verschlüsselung



Alice erhält dadurch das verschlüsselte Dokument.

Asymmetrische Verschlüsselung



Alice schickt das verschlüsselte Dokument an Bob.

Asymmetrische Verschlüsselung



Bob erhält das verschlüsselte Dokument.

Asymmetrische Verschlüsselung



Bob wendet seinen privaten Schlüssel auf das verschlüsselte Dokument an.

Asymmetrische Verschlüsselung



Bob erhält nun das unverschlüsselte Originaldokument.

Digitale Signaturen



Bob hat ein Schlüsselpaar bestehend aus einem öffentlichen Schlüssel (gelb) und einen privaten Schlüssel (grün) erstellt.

Digitale Signaturen



Alice bezieht Bobs öffentlichen Schlüssel.

Digitale Signaturen



Bob erstellt ein Dokument, das signiert werden soll.

Digitale Signaturen



Bob wendet seinen privaten Schlüssel auf das Dokument an.

Digitale Signaturen



Bob erhält dadurch das signierte Dokument.

Digitale Signaturen



Bob schickt das signierte Dokument an Alice.

Digitale Signaturen



Alice erhält das signierte Dokument.

Digitale Signaturen



Alice wendet Bobs öffentlichen Schlüssel auf das signierte Dokument an.

Digitale Signaturen



Alice hat sichergestellt, dass dieses Dokument von Bob stammt und nicht verfälscht wurde.

Wie baut man solche Verfahren?

Einwegfunktionen

- ▶ $f: X \rightarrow Y$
- ▶ Berechnung von $y = f(x)$ ist effizient möglich (d. h. in probabilistischer Polynomialzeit)
- ▶ die Umkehrfunktion $x = f^{-1}(y)$ jedoch nicht

Wie baut man solche Verfahren?

Einwegfunktionen

- ▶ $f: X \rightarrow Y$
- ▶ Berechnung von $y = f(x)$ ist effizient möglich (d. h. in probabilistischer Polynomialzeit)
- ▶ die Umkehrfunktion $x = f^{-1}(y)$ jedoch nicht

Trapdoor-Einwegfunktionen

- ▶ es gibt eine zusätzliche Information t
- ▶ mit dieser ist eine effiziente Berechnung von f^{-1} möglich

Wie baut man solche Verfahren?

Einwegfunktionen

- ▶ $f: X \rightarrow Y$
- ▶ Berechnung von $y = f(x)$ ist effizient möglich (d. h. in probabilistischer Polynomialzeit)
- ▶ die Umkehrfunktion $x = f^{-1}(y)$ jedoch nicht

Trapdoor-Einwegfunktionen

- ▶ es gibt eine zusätzliche Information t
- ▶ mit dieser ist eine effiziente Berechnung von f^{-1} möglich

Beispiel

- ▶ Multiplikation von großen Primzahlen ist einfach.
- ▶ Man geht davon aus, dass Faktorisierung schwierig ist.
- ▶ bester bekannter Algorithmus: General number field sieve
Laufzeit: $\mathcal{O}(\exp(c \cdot (\log n)^{\frac{1}{3}} \cdot (\log \log n)^{\frac{2}{3}}))$ (subexponentiell)

Etwas Mathe

Teilen mit Rest

Seien $a, n \in \mathbb{Z}$ und $n > 0$. Dann gibt es eindeutige $q, r \in \mathbb{Z}$, so dass $a = n \cdot q + r$ und $0 \leq r < n$.

Man schreibt

$$a \bmod n = r \quad \text{oder} \quad a \equiv r \pmod{n}.$$

Etwas Mathe

Teilen mit Rest

Seien $a, n \in \mathbb{Z}$ und $n > 0$. Dann gibt es eindeutige $q, r \in \mathbb{Z}$, so dass $a = n \cdot q + r$ und $0 \leq r < n$.

Man schreibt

$$a \bmod n = r \quad \text{oder} \quad a \equiv r \pmod{n}.$$

Sei $\mathbb{Z}_n$ die Menge der Kongruenzklassen modulo n . $\mathbb{Z}_n$ ist ein Ring:

- ▶ es gibt 0 und 1
- ▶ es gibt Addition und Multiplikation
- ▶ Addition ist invertierbar

Etwas Mathe

Theorem (Kleiner Satz von Fermat)

Für jede Primzahl p und jedes $\alpha \in \mathbb{Z}_p$ gilt

$$\alpha^p = \alpha.$$

Für $\alpha \neq 0$ gilt damit auch $\alpha^{p-1} = 1$.

Etwas Mathe

Theorem (Kleiner Satz von Fermat)

Für jede Primzahl p und jedes $\alpha \in \mathbb{Z}_p$ gilt

$$\alpha^p = \alpha.$$

Für $\alpha \neq 0$ gilt damit auch $\alpha^{p-1} = 1$.

Theorem (Chinesischer Restsatz)

Seien $n_1, \dots, n_k \in \mathbb{N}$ paarweise relativ prim und $n = \prod_{i=1}^k n_i$.
Dann ist die Abbildung

$$\begin{aligned} \tau: \mathbb{Z}_n &\rightarrow \mathbb{Z}_{n_1} \times \dots \times \mathbb{Z}_{n_k} \\ x \bmod n &\mapsto (x \bmod n_1, \dots, x \bmod n_k) \end{aligned}$$

ein Ringisomorphismus.

RSA

Schlüsselerzeugung

- ▶ Wähle zwei große Primzahlen $p \neq q$.
- ▶ Setze $n = p \cdot q$.
- ▶ Berechne $\varphi(n) = (p - 1) \cdot (q - 1)$
- ▶ Wähle $e > 1$ mit $\gcd(e, \varphi(n)) = 1$
(z. B. $e = 2^{16} + 1 = 65537$)
- ▶ Finde d mit $e \cdot d \equiv 1 \pmod{\varphi(n)}$
- ▶ Public Key $K_{pub} = (n, e)$
- ▶ Secret Key $K_{sec} = (p, q, d)$

Was macht man mit diesen Zahlen?

- ▶ Man bildet die Nachricht auf ein Element $m \in \mathbb{Z}_n$ ab
 - ▶ Verwendung von Padding notwendig (z. B. OAEP)
- ▶ Verschlüsselung:
 - ▶ Verschlüsseln: $c \leftarrow m^e \pmod{n}$
 - ▶ Entschlüsseln: $m \leftarrow c^d \pmod{n}$
- ▶ Digitale Signatur:
 - ▶ Signieren: $s \leftarrow m^d \pmod{n}$
 - ▶ Verifizieren: $s^e \stackrel{?}{=} m \pmod{n}$

Wieso funktioniert das?

RSA

Korrektheit

Theorem (Korrektheit von RSA)

Für alle $m \in \mathbb{Z}_n$ gilt

$$(m^e)^d = m \pmod{n}.$$

RSA

Korrektheit

Theorem (Korrektheit von RSA)

Für alle $m \in \mathbb{Z}_n$ gilt

$$(m^e)^d = m \pmod{n}.$$

Beweis.

Sei $m \in \mathbb{Z}_n$ (o. B. d. A. $m \neq 0$) und $c = m^e \pmod{n}$, dann gilt

$$\begin{aligned} c^d &= (m^e)^d = m^{ed} = m^{k \cdot (p-1) \cdot (q-1) + 1} \\ &= (m^{p-1})^{k \cdot (q-1)} \cdot m \stackrel{(\star)}{=} 1^{k \cdot (q-1)} \cdot m = m \pmod{p}. \end{aligned}$$

($\star$) Kleiner Satz von Fermat

Obiges gilt analog für q und mit dem Chinesischen Restsatz folgt

$$(m^e)^d = m \pmod{n}.$$



RSA

Sicherheit

Probleme:

RSAP Gegeben Ciphertext m^e und Public Key (n, e) ,
berechne m .

*RSAP** Gegeben Public Key (n, e) ,
bestimme den Secret Key d .

FACTORIZING Gegeben $n = pq$, bestimme p und q

RSA

Sicherheit

Probleme:

RSAP Gegeben Ciphertext m^e und Public Key (n, e) ,
berechne m .

*RSAP** Gegeben Public Key (n, e) ,
bestimme den Secret Key d .

FACTORING Gegeben $n = pq$, bestimme p und q

Beziehungen:

$$RSAP \leq_p RSAP^* =_p FACTORING$$

Es gibt keinen Beweis, dass *RSAP* schwierig ist.

Werbung

Ihr interessiert euch für Kryptographie?

- ▶ Bei den CInsects gibt es Menschen, die sich mit sowas beschäftigen.
- ▶ Angriffe auf RSA sind ein häufiges Thema.
- ▶ mehr Informationen:
 - ▶ <https://cinsects.de>
 - ▶ cinsects@informatik...



GnuPG

Wie geht das in der Praxis?

- ▶ Pretty Good Privacy von Phil Zimmerman (1990)
- ▶ Internet Standard: OpenPGP (RFC4880)
- ▶ freie Implementation: Gnu Privacy Guard (GnuPG / GPG, <https://gnupg.org/>)

GnuPG

Wie geht das in der Praxis?

- ▶ Pretty Good Privacy von Phil Zimmerman (1990)
- ▶ Internet Standard: OpenPGP (RFC4880)
- ▶ freie Implementation: Gnu Privacy Guard (GnuPG / GPG, <https://gnupg.org/>)

Und wie kann ich das benutzen?

- ▶ Kommandozeile
 - ▶ `$ gpg ...`
- ▶ Erweiterung für Thunderbird: Enigmail
<https://www.enigmail.net>
 - ▶ Installation:
https://enigmail.wiki/Installation_of_Enigmail

Schlüsselerzeugung

Was zu tun ist:

- ▶ sich eine starke Passphrase überlegen
 - ▶ diese schützt den Schlüssel auf eurer Festplatte
 - ▶ z. B. mit Diceware
- ▶ einen Schlüsseltyp wählen
 - ▶ RSA mit einer Schlüssellänge von 4096 bit
(alternativ Curve25519/EdDSA25519)
- ▶ ein Ablaufdatum setzen
 - ▶ Schlüssel wird nach bestimmter Zeit ungültig
 - ▶ kann (auch im Nachhinein) verlängert werden
- ▶ ein Revocation Certificate erstellen
 - ▶ erlaubt Zurückziehen des Schlüssels bei Kompromittierung

Keyserver

Woher bekomme ich den öffentlichen Schlüssel einer Person?

- ▶ von ihr persönlich (im Optimalfall)
- ▶ von einem Keyserver

Keyserver

Woher bekomme ich den öffentlichen Schlüssel einer Person?

- ▶ von ihr persönlich (im Optimalfall)
- ▶ von einem Keyserver

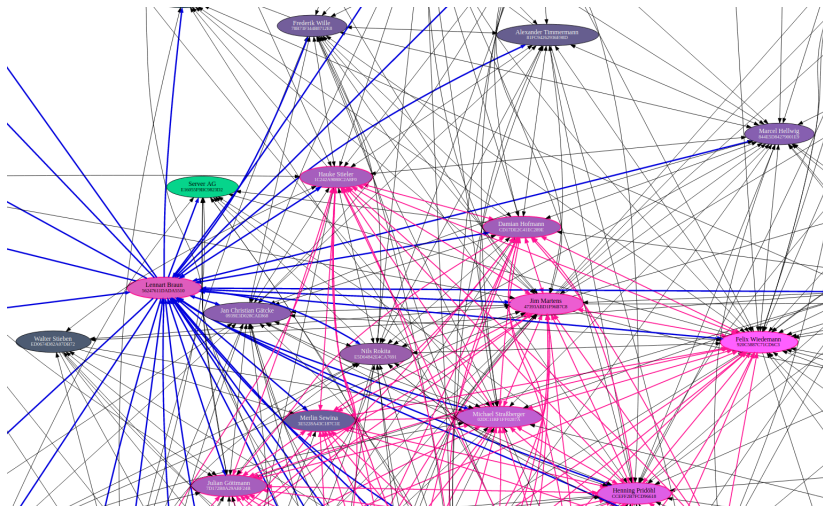
Keyserver ...

- ▶ ...sind eine Art Telefonbuch für PGP-Schlüssel
- ▶ ...erlauben das Suchen und Einstellen von Schlüsseln
- ▶ Nachteil: die E-Mail-Adressen sind öffentlich einsehbar (kann zu mehr Spam führen)
- ▶ Es gibt einen Keyserver auf Mafiasi.de:
<https://mafiasi.de/pks/>

Web of Trust

- ▶ Alice möchte Charlie eine verschlüsselte Nachricht schicken, hat aber seinen Public Key noch nicht.
- ▶ Sie findet einen Schlüssel mit Charlies E-Mail-Adresse auf einem Keyserver, ist sich jedoch nicht sicher, dass dieser auch wirklich Charlie gehört.
- ▶ Bob kennt sowohl Alice als auch Charlie und hat Charlies Public Key signiert.
- ▶ Bob bezeugt damit, dass der Schlüssel Charlie gehört.
- ▶ Wenn Alice Bob vertraut, *kann* sie sich auf ihn verlassen diesen Schlüssel verwenden.

Web of Trust



<https://mafiasi.de/pks/graph>

Vertrauen

Vertrauen ist schwierig, es gibt verschiedene Arten:

Vertrauen darauf, ...

- ▶ ... dass der Schlüssel auch zu der richtigen Person gehört.
- ▶ ... dass der Schlüssel nicht kompromittiert ist.
- ▶ ... dass eine Person die Identität des Inhaber gründlich prüft, bevor sie einen Schlüssel unterschreibt.
- ▶ Vertrauen zu der Person generell

Verwendung mit Enigmail

Hier folgt der praktische Teil.

Installation

GnuPG

- ▶ Linux: `/usr/bin/gnupg`
bzw. über den Package Manager der Distribution
- ▶ Windows: GPG4Win (<https://www.gpg4win.org/>)
- ▶ Mac: GPGTools (<https://gpgtools.org/>)

Enigmail

- ▶ <https://www.enigmail.net>
- ▶ Installation:
https://enigmail.wiki/Installation_of_Enigmail

GnuPG

Konfigurationsdatei

Unix: \$HOME/.gnupg/gpg.conf

Windows: %APPDATA%\gnupg\gpg.conf

Ausschnitt:

```
keyid-format long
with-fingerprint
personal-cipher-preferences AES256 AES192 AES CAST5
personal-digest-preferences SHA512 SHA384 SHA256
    ↪ SHA224
personal-compress-preferences ZLIB BZIP2 ZIP
#weak-digest SHA1
cert-digest-algo SHA512
default-preference-list SHA512 SHA384 SHA256 SHA224
    ↪ AES256 AES192 AES CAST5 ZLIB BZIP2 ZIP
    ↪ Uncompressed
```

Wir wollen kein SHA1!