IT-Sicherheit

Prof. Dipl.-Ing. Klaus Knopper

Stand: 14. Januar 2018

Master-Vorlesung an der Hochschule Kaiserslautern
im Wintersemester 2017/18

Inhaltsverzeichnis

1 Organisatorisches

Organisatorisches

☞ Modul „Recht der Informationstechnologie und IT-Sicherheit“, Teil „IT Sicherheit“, Vorlesung mit Übungen jeweils Dienstags 16:00 Uhr in A209.1

Di. 10.10.2017	Organisatorisches, Einführung
Di. ...	...
Noch nicht im Campusboard	Klausur

☞ **http://knopper.net/bw/itsec**

Folie 1

Kursziel

- Grundlagen der zu schützenden Güter und Schutzmechanismen kennen lernen,
- Schwachstellen und Angriffspunkte sowie Ansätze zur Verteidigung kennen lernen (praktischer Teil),
- IT-Sicherheit als integraler Bestandteil eines Gesamtkonzeptes verstehen,
- „Best Practice“ Beispiele, Normen und Anleitungen zur IT-Sicherheit kennen.

Folie 2

Zur Benutzung von Folien/Skript

- Der Foliensatz wird nach Bedarf während des Semesters erstellt (Work in Progress). Daher bitte Vorsicht beim Ausdrucken.
- Verweise auf sinnvolle ☞ Sekundärliteratur sind entsprechend gekennzeichnet und i.d.R. direkt anklickbar.
- Prüfungsrelevant (Klausur) sind grundsätzlich alle in der Vorlesung und in den Übungen behandelten Themen.

Folie 3

In der Skript-Version befinden sich neben den Folien aus der Vorlesung auch noch weitere Kommentare, Fußnoten und Links, die zur Wiederholung und Klausurvorbereitung nützlich sein können.

2 Einleitung

2.1 Einordnung IT-Sicherheit

IT-Sec. und der Business Value of IT (1)

$$\text{BVIT} = \frac{\text{Business Performance}}{\text{IT Investment}}$$

IT Investment: Eine Investition in die **Fähigkeit, ein Geschäft zu führen**
Der **Geschäftserfolg** (Business success, business value, linke Seite der Gleichung) hängt wesentlich davon ab, den Zählerwert in der Wertegleichung (Business Performance) zu erhöhen, nicht (nur) den Wert des Nenners zu reduzieren.

Folie 4

IT-Sec. und der Business Value of IT (2)

Was bedeutet dies für die „IT Security“, die ja als Investition im Nenner steht und sich somit grundsätzlich erst mal „negativ“ auf das Ergebnis auswirkt?

☞ Die Business Performance hängt vom Funktionieren IT ab. Ein Ausfall durch unzureichendes IT Investment in die Infrastruktur kann also den Wert des Zählers in verheerender Weise zerstören.

Folie 5

Die Art der Investition in IT-Sicherheit muss bereits in der Planung der Infrastruktur berücksichtigt werden. Eine nachträgliche „Aufbesserung“ ist i.d.R. mit deutlich höheren Kosten verbunden als eine strukturelle Maßnahme von Anfang an.

2.2 Ziele der Informationssicherheit

Ziele

- ☞ Vertraulichkeit
- ☞ Integrität
- ☞ Verfügbarkeit

Folie 6

Normen und Zertifizierungen

☞ IT-Sicherheitsmanagement: internationale ☞ ISO/IEC-27000-Reihe

Im deutschsprachigen Raum: ☞ IT-Grundschutz

Evaluierung und Zertifizierung von IT-Produkten und -systemen: ☞ ISO/IEC 15408 (Common Criteria)

Folie 7

IT-Sicherheitsstandards und Handbücher

Hier ist zwischen technischen Maßnahmen (Verschlüsselung, Firewall, Zugangskontrolle, physikalische Netzwerk- und Systemtrennung etc.) und organisatorischen Maßnahmen (Policy, Konzept) zu unterscheiden, die ineinander greifen müssen, um zum Erfolg zu führen.

Norm	Inhalte
ISO/IEC 15408	Common Criteria
ISO/IEC 27001	Informationssicherheits-Managementsysteme (ISMS)
ISO/IEC 27002	Informationssicherheits-Managementsysteme (ISMS)
BSI-Standard 100-1	Managementsysteme für Informationssicherheit
BSI-Standard 100-2	IT Grundschutz Vorgehensweise (BSI2)
BSI-Standard 100-3	Risikoanalyse auf Basis von IT-Grundschutz (BSI3)

Abbildung 1: Liste von Normen und Empfehlungen zur Informationssicherheit

Auszug aus BITKOM: Kompass der IT-Sicherheitsstandards
http://www.kompass-sicherheitsstandards.de/

Fragen: Wird bei diesen Standards tatsächlich ein „hoher Grad von Sicherheit" zertifziert oder gefordert? Was genau wird bei „Common Criteria" („Allgemeine Kriterien für die Bewertung der Sicherheit von Informationstechnologie") eigentlich betrachtet? (auch ☞ Kritik lesen!)

2.2.1 Vertraulichkeit

Vertraulichkeit

☞ Vertraulichkeit ist die Eigenschaft einer Nachricht, nur für einen beschränkten Empfängerkreis vorgesehen zu sein. Weitergabe und Veröffentlichung sind nicht erwünscht. Vertraulichkeit wird durch Rechtsnormen geschützt, sie kann auch durch technische Mittel gefördert oder erzwungen werden.

Maßnahmen: ☞ Verschlüsselung, ☞ Digitale Rechteverwaltung (z.B. DRM)

Folie 8

2.2.2 Verfügbarkeit

Verfügbarkeit

Die Verfügbarkeit eines technischen Systems ist die **Wahrscheinlichkeit** oder das Maß, dass das System bestimmte Anforderungen zu einem bestimmten Zeitpunkt bzw. innerhalb eines vereinbarten Zeitrahmens erfüllt.

Als Kennzahl:

$$\text{Verfügbarkeit} = \frac{\text{Gesamtzeit} - \text{Ausfallzeit}}{\text{Gesamtzeit}}$$

Auch: Uptime $=$ Gesamtzeit $-$ Ausfallzeit oder „Mean Time between Failure„ (☞ MTBF)

Maßnahmen: Redundanz, Backup, Archivierung, Hot-Standby, ...

Problem: Haltbarkeit aktueller physischer Datenträger!

Folie 9

2.2.3 Integrität

Integrität

s.a. Wikipedia

Alte Definition: „Verhinderung unautorisierter Modifikation von Information“

Bundesamt für Sicherheit in der Informationstechnik (BSI): „Korrektheit (Unversehrtheit) von Daten und der korrekten Funktionsweise von Systemen“ (weiter gefasst)

Kriterien: **Korrekter Inhalt**, **Unmodifizierter Zustand** bzw. die **Möglichkeit, Modifikationen zu erkennen und zuordnen zu können**

Maßnahmen: ☞ **Prüfsummen**, ☞ **Digitale Signatur**

Folie 10

3 Verschlüsselung

Verschlüsselung

Ziel: Nur **berechtigten Personen** den Zugriff auf die Daten ermöglichen (Vertraulichkeit).

Mittel: Anwendung (möglichst sicherer) mathematisch-algorithmischer Verfahren.

- Symmetrische Verschlüsselung [6]
- Asymmetrische Verschlüsselung [8]

Folie 11

Während in der **symmetrischen Verschlüsselung** der gleiche Code („symmetrischer Schlüssel“) zum Verschlüsseln und Entschlüsseln einer Nachricht verwendet wird, ist beim asymmetrischen Verfahren zwei **unetrschiedliche, aber zusammen gehörende** Schlüssel zum Ver- und Entschlüsseln notwendig. Beide Verfahren werden in diesem Skript exemplarisch gezeigt.

3.1 Symmetrische Verschlüsselung

Symmetrische Verschlüsselung

Der Caesar-Algorithmus ist eines der einfachsten (und unsicheren) Verfahren, um Verschlüsselung zu zeigen.

```
if(eingabe >= 'A' && eingabe <= 'M')
  ausgabe = eingabe + 13;
else if(eingabe >= 'N' && eingabe <= 'Z')
  ausgabe = eingabe - 13;
else
  ausgabe = eingabe;
```

Folie 12

Aus dem Eingabetext **"HALLO WELT"** wird **"UNYYB JRYG"** und umgekehrt, der Algorithmus ist also umkehrbar. Der „Schlüssel" ist die Zahl 13, um die das Eingabezeichen nach links oder rechts im Alphabet verschoben wird.

XOR

Ein einfacher, auf Computern besonder effizient einsetzbarer Verschlüsselungs-Algorithmus, ist das bitweise **Exklusiv Oder**, dessen Anwendung ebenfalls umkehrbar ist:

a	**b**	**a** $\oplus$ **b**
0	0	0
1	0	1
0	1	1
1	1	0

Folie 13

Block Cipher (1)

Da die bitweise Anwendung immer in Blöcken mit „sinnvoller Größe“ durchgeführt wird, (ggf. Größe eines Prozessor-Registers), spricht man von einer Blockchiffre (Block Cipher).
Beispiel (Pseudocode) XOR:

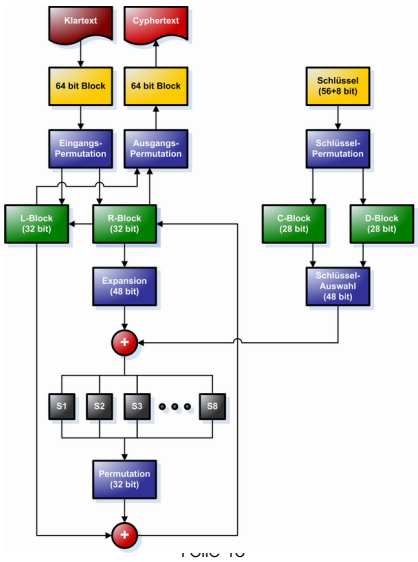
```
u_32   schluessel=0xabcd;
u_32[] data = "Hallo, Welt ...";
for(int i=0; i < sizeof(data) / sizeof(u_32); i++)
 data[i] =  data[i] ^ schluessel;
```

Folie 14

Im Gegensatz zum Caesar-Algorithmus ist XOR ein mathematisch beweisbar sicheres Verfahren. Die Schlüssellänge ist entscheidend, um das „Erraten“ des Schlüssels durch Ausprobieren zu erschweren. Für die XOR-Verknüpfung lässt sich der Schlüssel leicht berechnen, wenn der verschlüsselte *und* der unverschlüsselte Datensatz bekannt sind (plain text Attacke).

Das Problem der Zurückrechnung des Schlüssels auf dem verschlüsselten und unverschlüsselten Datensatz kann durch Blockchiffren umgangen werden, die positionsabhängig Daten „durcheinanderwürfeln“, wodurch bei jeder noch so kleinen Änderung am Originaltext ein völlig anderer Ausgabetext entsteht. Eine nur teilweise Dechiffrierung bzw. das Erraten eines Teilschlüssels ist somit schwieriger.

DES



Da DES im Design eine sehr geringe Schlüssellänge besitzt, ist es mit genügend Rechen-Resourcen relativ leicht zu brechen. AES gilt mit 128 oder mehr Bits Schlüssellänge als nach heutigen Standards sicher.

AES

Der Advanced Encryption Standard wurde als DES-Nachfolger in einer lizenzkostenfreien Implementierung von Joan Daemen und Vincent Rijmen erfunden (auch „Rijndael-Algorithmus“ genannt). Er ist heute Standard in den meisten Crypto-Produkten und wird auch bei der Datenübertragung im Internet (https, WPA etc.) eingesetzt.

Folie 16

3.2 Asymmetrische Verschlüsselung

Problem bei der symm. Verschlüsselung

Während die Verschlüsselung an sich bei modernen Verfahren wie AES als sicher gelten kann, ist die sichere Aufbewahrung und Verteilung der Schlüssel ein großes Problem.

Bei bekannt werden des symmetrischen Schlüssels können alle damit verschlüsselten Dokumente entschlüsselt werden.

Zwei Personen, die verschlüsselt kommunizieren wollen, stehen somit zunächst vor dem Problem, den Schlüssel auf sichere Weise zu erhalten.

Folie 17

Zwei komplementäre Schlüssel

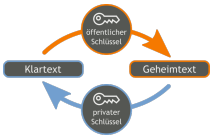
Bei der asymmetrischen Verschlüsselung werden zwei sich gegenseitig ergänzende (komplementäre) Schlüssel generiert. Daten, die mit einem der Schlüssel verschlüsselt werden, können NICHT mit dem gleichen Schlüssel wieder entschlüsselt werden, sondern nur mit dem anderen.

Einer der Schlüssel wird geschützt aufbewahrt (z.B. in einer symmetrisch verschlüsselten Datei) und ist nur dem Besitzer bekannt. ☞ **Secret Key** (mitunter auch als „Private Key“ bezeichnet).

Der anderen Schlüssel wird an die Kommunikationspartner verteilt, und muss nicht besonders geschützt werden. ☞ **Public Key** (Öffentlicher Schlüssel).

Folie 18

Ablauf der sicheren Kommunikation

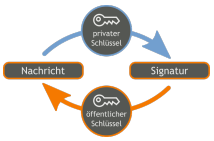


1. Eine Nachricht, die sicher übertragen werden soll, wird mit dem Öffentlichen Schlüssel des beabsichtigten Empfängers (der dem Sender bekannt ist) verschlüsselt.
2. Die verschlüsselte Nachricht wird über einen, möglicherweise „unsicheren“ Kanal (jeder kann sie kopieren) an den Empfänger übertragen. Mit Hilfe des Öffentlichen Schlüssels kann sie jedoch nicht wieder lesbar gemacht werden.
3. Der Empfänger, welcher den passenden Privaten Schlüssel besitzt, kann die Nachricht lesen.

Folie 19

... wandert später ggf. in den Abschnitt „Schutz vor Manipulation“. In Kürze:

Ablauf der Verifikation von Daten



1. Der Urheber eines digitalen Dokumentes bildet eine Prüfsumme (Hash) über das Dokument.
2. Er verschlüsselt diese Prüfsumme, diesmal aber mit seinem Geheimen Schlüssel, der nur ihm selbst zugänglich ist.
3. Das Dokument wird zusammen mit der verschlüsselten Prüfsumme an die beabsichtigten Empfänger verschickt (unverschlüsselt oder nicht, spielt dabei keine Rolle).
4. Die Empfänger können mit Hilfe des ihnen bekannten Öffentlichen Schlüssels des Dokumenten-Urhebers die Prüfsumme dechiffrieren, und überprüfen, ob sie noch zum Dokument passt. Ist das nicht der Fall, wurde das Dokument manipuliert. Ansonsten ist es authentisch.

Folie 20

Erzeugung und Benutzung

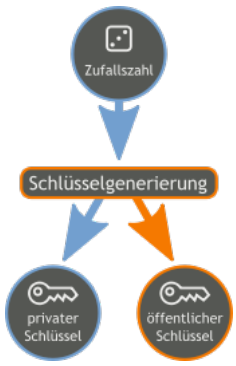
... eines Asymmetrischen Schlüsselpaars: Siehe Skript.

Die Anwendung der Modulo-Operation (Teilungsrest) innerhalb des Algorithmus sorgt dafür, dass bestimmte Schritte nicht umkehrbar sind! Mit z.B. dem Teilungsrest „9“ bei „X Modulo 10“ kann, offensichtlich, nicht die Zahl X berechnet werden. Hierdurch ist es unmöglich, zu einem Öffentlichen Schlüssel den Privaten Schlüssel exakt zu berechnen.

Es gibt, aus dem gleichen Grund, mathematisch gesehen, unendlich viele Private Schlüssel, die genauso komplementär zu einem Öffentlichen Schlüssel sind wie der ursprünglich generierte, allerdings sind sie bei der erforderlichen hohen Schlüssellänge praktisch unmöglich durch „Ausprobieren“ zu finden.

Folie 21

3.3 Beispiel: Generierung und Benutzung eines asymmetrischen Schlüsselpaars nach dem RSA-Verfahren



Wir verwenden in diesem Beispiel, damit es noch ohne Computer mit Kopfrechnen funktioniert, sehr kurze Zahlen. In der Praxis können dies Werte mit mehreren hundert Stellen sein.

1. Wähle zwei zufällige (normalerweise sehr, sehr große) Primzahlen p und q, wobei $p \neq q$,
2. Berechne das RSA-Modul $N = p \cdot q$,
3. Berechne die Eulersche Funktion $\varphi(N) = (p-1) \cdot (q-1)$,
4. Wähle (zufällig) eine zu $\varphi(N)$ teilerfremde Zahl e (d.h. es gibt keine gemeinsamen ganzzahligen Teiler), die größer als 1 und kleiner als $\varphi(N)$ ist, dies ist der sog. **Verschlüsselungsexponent**,
5. Berechne den **Entschlüsselungsexponenten** d mit der Formel $e \cdot d + k \cdot \varphi(N) = 1$

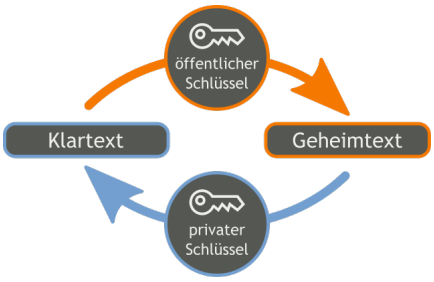
Das Schlüsselpaar ist nun e („encrypt") und d („decrypt"), die beide mit dem gleichen RSA-Modul N verwendet werden.

Die im Algorithmus zu Beginn festgelegten, gewählten oder berechneten Werte p, q, k werden nicht mehr benötigt.

Ein Beispiel mit konkreten Zahlen:[1]

1. Wähle Primzahlen: $p = 11$ und $q = 13$
2. Das RSA-Modul ergibt sich zu $N = p \cdot q = 143$
3. Eulersche Funktion: $\varphi(N) = (p-1)(q-1) = 120$
4. Wähle e teilerfremd zu 120: $e = 23$. Damit bilden $e = 23$, $N = 143$ den Öffentlichen Schlüssel.
5. $e \cdot d + k \cdot \varphi(N) = 1$ hat eine Lösung mit $k = -9$ und $d = 47$, somit ist $d = 47$ der geheime Entschlüsselungsexponent (Secret Key).

[1] s.a. Wikipedia



Verschlüsselung

Die verschlüsselten Daten c werden aus dem Originaltext m mit Hilfe des Verschlüsselungsexponenten e und des Moduls N berechnet nach der Formel

$$c = m^e \, mod \, N$$

also

$$2 = 7^{23} \, mod \, 143$$

Entschlüsselung

Der entschlüsselte Text m ergibt sich mit dem verschlüsselten Text c, dem geheimen Entschlüsselungsexponenten d und dem Modul N nach der Formel

$$m = c^d \, mod \, N$$

also

$$7 = 2^{47} \, mod \, 143$$

Angriffsvektor bei Asymm. Verschlüsselung

Der Öffentliche Schlüssel kann zwar problemlos über ein unsicheres Netzwerk übertragfen werden, da man mit ihm die verschlüsselten Daten nicht wieder entschlüsseln kann, jedoch kann ein Angreifer versuchen, mit falscher Identität einen Öffentlichen Schlüssel für jemand anders zu propagieren, zu dem er selbst jedoch den passenden geheimen Schlüssel besitzt.

Lösungsansatz: Schlüssel von einer „vertrauenswürdigen Instanz“ signieren lassen, deren Öffentlicher Schlüssel unwiderlegbar authentisch ist. (Ist das so? ☞ Liste von vertrauenswürdigen Signierern in Firefox)

Folie 22

SSL-Zertifikat

...ist nichts anderes als eine Datei mit

1. Öffentlicher Schlüssel,
2. weitere Informationen wie Name, Mailadresse, Webseite, Postadresse, Geodaten, ...
3. digital signiert von einer „vertrauenswürdigen Instanz“ (oder, im einfachsten Fall, vom Besitzer selbst mit Hilfe seines NICHT im ARCHIV ENTHALTETEN Privaten Schlüssels).

Eine Veränderung von Daten innerhalb des Zertifikats führt dazu, dass sich ihre Prüfsumme ändert und damit die digitale Signatur hinfällig wird.

Das Zertifikat wird, wie der Offentliche Schlüssel, an alle Kontakte verteilt.

Zu jedem Zertifikat muss der Eigentümer den Privaten Schlüssel separat gut gesichert aufbewahren.

Folie 23

Softwaretechnik

- openssl (Secure Socket Layer)
  - Webserver (https)
  - Mail-Server (smtps, pop3s, imaps)
  - Mail-Clients (S/MIME)
  - Datei- und Archiv-Manager mit Verschlüsselung
  - Access Points (WPA, WPA Enterprise)
  - div. Server: SAMBA, LDAP, MYSQL, SSH
- PGP (Pretty Good Privacy)
  - Mail-Programme (PGP-MIME)
  - Dateien verschlüsseln

Folie 24

Best Practice

Der Industriestandard ist ☞ SSL (Secure Socket Layer) bzw. TLS (Transport Layer Security) mit ☞ OpenSSL als Open Source Standard Implementierung.

Die OpenSSL Libraries, die in fast allen Krypto-Produkten verwendet werden, werden ständig aktualisiert und fehlerbereinigt. Einige Versionen haben allerdings designbedingte Fehler und werden daher nicht mehr eingesetzt (d.h. z.B. im Webserver abgeschaltet, Verbindungen mit angreifbaren Verschlüsselungsalgorithmen oder Fehlern im Schlüsselaustausch werden abgewiesen).

Aufgrund der Komplexität und der vielfältigen Angriffsszenarien im Kryptobereich gilt es als fahrlässig, eine eigene, proprietäre Verschlüsselung in kritischen Bereichen zu implementieren und einzusetzen!

Intermezzo: Aus aktuellem Anlass...

Angriff auf WPA2 (WLAN): Was ist dran? Risiken? Was muss man tun?

Tabelle der angenommenen und tatsächlichen Angriffsszenarien!

S.a. separates Handout zu „WPA2 Krack Attack“.

3.4 Übungen zu Verschlüsselung und Signatur

Übungen zu Verschlüsselung und Signatur

Ziele:

1. Fähigkeit, eine persönliche Public-/Secret Key Infrastruktur einzurichten.
2. Fertigkeit erlernen, auf Datei-Ebene Verschlüsselung und Signatur „Low Level“ durchzuführen.
3. ☞ Übertragung auf Software-Produkte wie Mailprogramm, Office, Browser, Zugangs-Software.

Mittel:

1. PGP (Pretty Good Privacy) bzw. GnuPG (Open Source Version),
2. OpenSSL (Secure Socket Layer Frontend-Programme)

Folie 27

Hinweis: Wenn Sie die Windows-Version der Kommandozeilen-basierten Tools verwenden, müssen Sie unter Windows ggf. die Binärverzeichnisse, in denen die Programme installiert sind (**`gpg.exe`** und **`openssl.exe`**) in den Suchpfad aufnehmen, oder in das jeweilige Verzeichnis mit **`cd Verzeichnisname`** wechseln, damit die Programe gefunden werden.

Üblicherweise werden die Programme jedoch von den graphischen Frontends aufgerufen, und arbeiten im Hintergrund. Wir rufen Sie hier zu Übungszwecken mit den passenden Kommandozeilenparametern auf, um zu lernen, wie sie arbeiten.

Unter Linux und Mac sind die Programme standardmäßig im Suchpfad.

3.4.1 GnuPG

GnuPG

Übungen: GnuPG

Website / Download: **https://www.gnupg.org/**

Aufgaben: Im Vorlesungs-Skript!

Folie 28

Hinweis: Eckige Klammer (...) bedeutet „optionaler Parameter", spitze Klammer <...> bedeutet „erforderlicher Parameter". — kennzeichnet eine Alternative (EBNF-Format).

Persönliches PGP-Schlüsselpaar erzeugen

gpg --gen-key

Hier werden als persönliche Daten der Name und die Mailadresse abgefragt.

Achtung: Sollten Sie schon ein PGP-Schlüsselpaar besitzen, sichern Sie die Dateien **secring.pgp** und **pubring.pgp** sicherheitshalber vorher, damit die vorhandenen Schlüssel nicht ersetzt oder überschrieben werden!

PGP-Schlüssel auflisten

gpg --list-keys [name]

Mit dem „Fingerabdruck" (Prüfsumme) lassen sich telefonisch oder persönlich die öffentlichen Schlüssel verifizieren, d.h. der Eigentümer des geheimen Schlüssel gibt den Fingerprint seines Schlüssels nach einem Beweis seiner Identität (Personalausweis, Reisepass etc.) persönlich bekannt.

gpg -v --fingerprint [name]

Öffentlichen Schlüssel veröffentlichen oder suchen

Es gibt mehrere weltweit zugängliche Datenbanken für PGP-Schlüssel, über die man auch nach dem Schlüssel eines Gesprächspartners suchen kann.

gpg --keyserver=x-hkp://pgp.mit.edu -a --send-keys <**ID|name**>

Einen Public Key hochladen.

gpg --keyserver=x-hkp://pgp.mit.edu -a --search-keys <**ID|name**>

Einen Public Key per ID oder Mailadresse suchen und ggf. in die eigene Datenbank übernehmen.

Der Keyserver und andere Parameter können auch in die Konfigurationsdatei **gnupg.conf** übernommen werden, er muss dann nicht jedesmal angegeben werden.

Öffentliche Schlüssel exportieren und importieren

Neben der Datenbank-Methode unterstützt GnuPG auch das exportieren und importieren von Dateien zum Austausch öffentlicher Schlüssel.

gpg -a -o pubkey.asc --export [ID|name]

Öffentlichen Schlüssel exportieren und in die Datei **pubkey.asc** speichern.

gpg -a -o secret_key.asc --export-secret-keys

Wenn der geheime Schlüssel von einem zum anderen Gerät transportiert werden soll, ist besondere Vorsicht geboten. Das Kommando exportiert ihn in eine Datei, die nach dem Import sofort wieder gelöscht werden sollte.

gpg --import <**keys.asc**>

Importiert Schlüssel, geheim oder öffentlich, aus der angegebenen Datei in die eigene Schlüsseldatenbank.

Signieren von Schlüsseln

gpg --sign-key <**ID|name**>

Um die Identität eines Schlüsselinhabers selbst zu bestätigen, kann mit dem eigenen Secret Key eine Unterschrift erzeugt werden. Vorher sollte natürlich der Inhaber zweifelsfrei feststehen (Ausweis) und der Fingerabdruck seines öffentlichen Schlüssens überprüft worden sein.

Schlüsseldatenbank editieren

gpg --edit-keys <**ID|name**>

Interaktives Verifizieren, Löschen, Signieren etc. von Schlüsseln in der Datenbank, auch Vertrauenslevel ändern.

Dateien verschlüsseln

gpg --encrypt -a -r <**ID-Empfänger**> <**datei**>

Die angegebene Datei verschlüsseln mit dem öffentlichen Schlüssel des beabsichtigten Empfängers. Um eine Datei zu verschlüsseln, die man selbst wieder entschlüsseln können soll, als **ID-Empfänger** unbedingt die eigene ID angeben, für die man den geheimen Schlüssel besitzt! Die Zieldatei wird (wg. **-a**) in eine ASCII-codierte Datei namens **datei.asc** gespeichert, die Originaldatei wird also nicht überschrieben.

Dateien entschlüsseln

gpg --decrypt -o datei.neu <**datei.asc**>

Entschlüsselt **datei.asc** mit Hilfe des geheimen Schlüssels, der zum öffentlichen Schlüssel passt, mit dem sie verschlüsselt wurde. Das Ergebnis wird in **datei.neu** gespeichert.

Dateien signieren

gpg --detach-sign -a <**datei**>

Erzeugt eins „abgetrennte Signatur“ **datei.asc**, also eine unterschriebene Prüfsumme, der angegebenen Datei, mit Hilfe des eigenen geheimen Schlüssels. Alternativ kann mit **--clearsign** eine Datei erzeugt werden, die sowohl die Originaldatei in ASCII-kodierter Form, als auch die Signatur enthält.

Datei mit Signatur verifizieren

gpg --verify <**datei.asc**> **[datei]**

Überprüft die angegebene Datei auf Veränderungen und korrekte (zu einem bestätigten öffentlichen Schlüssel) passende Unterschrift.

3.4.2 OpenSSL

Übungen: OpenSSL

Website: **https://www.openssl.org/**
Binary Downloads:
https://wiki.openssl.org/index.php/Binaries

Aufgaben: Im Vorlesungs-Skript!

Folie 29

Hinweise:

- Eckige Klammer (...) bedeutet „optionaler Parameter“, spitze Klammer <...> bedeutet „erforderlicher Parameter“. — kennzeichnet eine Alternative (EBNF-Format).
- Normalerweise wird zunächst ein „Zertifizierungs“-Schlüsselpar erzeugt, eine sogenatte „Certification Authority“ (aka CA). Hiermit werden dann die persönlichen Zertifikate, Server-Zertifikate oder weitere Zertifizierer signiert. Zur Verifikation benötigen die Clients (Browser, Mailprogramme etc.) dann die Kette der Zertifikate der Signierer. Eine solche Infrastruktur mit separaten Zertifizierungsstellen wird auch als PKI (Public Key Infrastructure) bezeichnet. Rechenzentren, die Schlüssel zentral verwalten, verwenden dies beispielsweise, und besitzen selbst ein Zertifikat, das von einer der Wurzel-Zertifizierungsstellen signiert ist, deren Zertifikat in den meisten SSL-fähigen Software-Produkten integriert ist. Wir verwenden hier der Einfachheit halber in den Übungen, analog PGP, selbstsignierte Zertifikate, die entsprechende Erweiterungen besitzen, um auch für andere Zwecke benutzt zu werden. Ähnlich wie bei PGP müssen diese erst vom Nutzer als „authentisch“ bestätigt bzw. als „Vertificate Chain“ importiert werden, bevor sie als „sicher“ gelten und für die Verifikation und Verschlüsselung geeignet sind.
-

Persönliches OpenSSL-Schlüsselpaar erzeugen und mit 10 Jahren Gültigkeit signieren (self-signed x509 certificate)

Infos:
☞ X509 Standard
Die ☞ ASN.1 Struktur der Zertifikate erinnert etwas an den Aufbau einer ☞ LDAP-Datenbank.

```
openssl req -new -x509 -nodes -days 3650 -out ssl.crt -keyout ssl.key
```

Die Informationen (Adresse, Name, E-Mail) werden in der Standard-Installation von openssl interaktiv abgefragt. Als ☞ Dateiendungen für das mit Option **-out** erzeugte Zertifikat sind neben dem gezeigten **.crt** auch **.pem** und andere üblich. Der Inhalt der Datei ist ☞ Base64-kodiert, damit wird sie beim versehentlichen Laden und Speichern mit einem Texteditor nicht zerstört, und kann auch über Transportwege mit weniger als 8Bit langen Zeichen zerstörungsfrei übertragen werden.

Das Dateiformat des mit Option **-keyout** erzeugten, privaten Schlüssels ist ebenfalls Base64-kodiert, im Beispiel wird sicherheitshalber eine nicht-standardisierte Endung **.key** verwendet, um den Key nicht versehentlich zu exportieren.

Die Option **-nodes** („No DES Encryption") sorgt dafür, dass die Datei, in der sich der private Schlüssel befindet, nicht noch symmetrisch verschlüsselt und mit Passwort geschützt wird, was eigentlich sinnvoll wäre, sich aber für die Benutzung mit automatisch startenden Diensten wie Web-Servern nicht gut verträgt.

X509-Zertifikate im Textformat dekodieren/Anzeigen

```
openssl x509 -text -in ssl.crt
```

Zeigt den Inhalt des Zertifikates mit Prüfsumme und Meta-Informationen in der Konsole an. Die Ausgabe enthält am Ende auch die Base64-kodierte Version.

```
openssl pkcs12 -export -out datei.p12 -inkey ssl.key -in ssl.crt -name "Mein Name"
```

Speichert das Schlüsselpaar in einem ☞ PKCS12-Container mit der Endung **.p12**, der von den meisten Mailprogrammen importiert werden kann. Es wird aus Sicherheitsgründen ein Import-Passwort abgefragt, das später beim Installieren im Mailprogramm wieder verlangt wird. Zusätzlich (und VORHER!) muss ggf. die Datei **ssl.crt** auch noch als „Zertifizierungsstelle" importiert werden, damit der öffentliche Schlüssel (der eigentlich in unserem Beispiel mit dem „Zertifizierer" identisch ist) als gültig erkannt wird.

Webserver OpenSSL-Schlüsselpaar erzeugen und mit 10 Jahren Gültigkeit signieren

Damit ein selbstsigniertes SSL-Zertifikat mit zugehörigem Secret Key von modernen Browsern akzeptiert wird, sind Erweiterungen in der Konfigurationsdatei **openssl.cnf** notwendig! Während früher nur ein einziger FQDN (Fully Qualified Domain Name) als „Common Name" möglich war, so können jetzt mit der Erweiterung **subjectAltName** mehrere DNS-Einträge für den gleichen Server gesetzt werden.

Im Abschnitt **[req]** der Konfigurationsdatei **openssl.cnf** ist die Zeile

```
req_extensions = v3_req
```

zu ergänzen. Im (evtl. neuen!) Abschnitt **[v3_req]** werden folgende Zeile eingefügt, und der Abschnitt **[alt_names]** eingefügt:

```
[ v3_req ]
basicConstraints = CA:TRUE
subjectAltName = @alt_names

[ alt_names ]
DNS.1 = meinserver.net
DNS.2 = www.meinserver.net
DNS.3 = mail.meinserver.net
DNS.4 = meinserver.shop
DNS.5 = weiterer.name.de
...
```

Nun kann das Kommando zur Generierung des Server-Schlüsselpaars aufgerufen werden, bei dem die Option **`-extensions v3_req`** anzugeben ist.

```
openssl req -new -x509 -extensions v3_req -nodes -days 3650 -out server.crt -keyout server.key
```

Die Informationen (Adresse, Name, E-Mail) werden in der Standard-Installation von openssl interaktiv abgefragt. Wichtig: Der „Common Name" ist nun nicht mehr der Name einer Person, sonder der „Haupt-Name" des Webservers, der ebenfalls in den Einträgen für das **`subjectAltName`**-Feld auftauchen sollte.

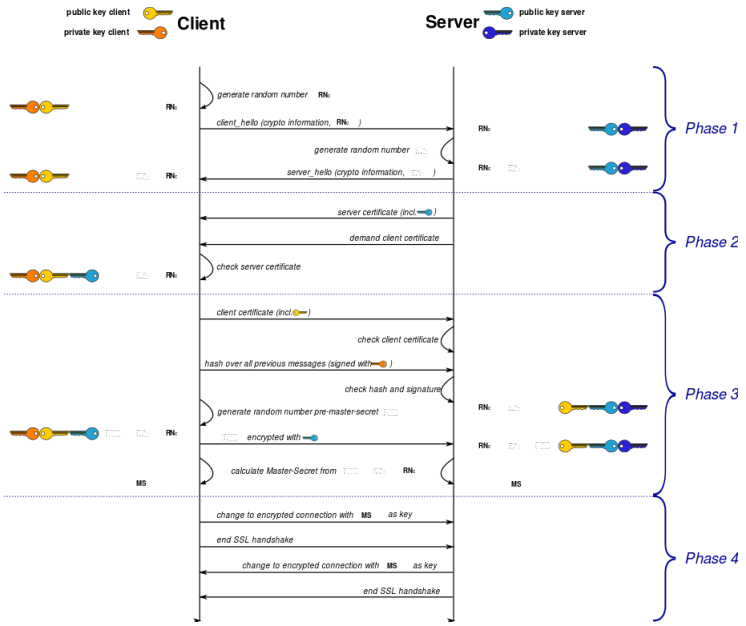
Verschlüsseln mit OpenSSL

```
openssl smime -encrypt -in datei -binary -outform DEM -out datei.enc ssl.crt
```

Entschlüsseln mit OpenSSL

```
openssl smime -decrypt -in datei.enc -binary -inform DEM -inkey ssl.key -out datei
```

Hinweis: Normalerweise werden Daten NICHT direkt asymmetrisch verschlüsselt, da dieses Verfahren algorithmisch sehr aufwändig und resourcenintensiv ist (Speicher, Rechenzeit). Stattdessen wird ein Symmetrischer Schlüssel für einen wählbaren Block Cipher Algorithmus mit Hilfe eines Zufallswertes (Salt) erzeugt, die Datei wird mit diesem verschlüsselt, und der verwendete symmetrische Schlüssel wird mit dem Public Key des Empfängers verschlüsselt an die verschlüsselte Datei angehängt bzw. in Phase 4 des ☞ Handshake beim https-Verbindungsaufbau ausgehandelt. Hiermit werden die Vorteile der schnellen symmetrischen Blockchiffren mit denen des sicheren asymmetrischen Schlüssel-Handlings verknüpft.



Aus der Skizze wird auch deutlich, dass https (tls) eine Authentiofizierung des Client beim Server über SSL-Zertifikate unterstützen. Hierfür muss der Client einen signierten Public Key präsentieren, deren Signatur vom Server als Authentifizierung akzeptiert wird (dieser kennt entweder den Fingerprint des Client-Zertifikates, oder den der unterschreibenden Certificate Authority, was das Hinzufügen weiterer Client-Zertifikate sehr leicht macht). Dieses Verfahren kann anstelle einer Passwort-Abfrage verwendet werden, der Besucher der Webseite muss dann das entsprechende Client-Zertifikat, das ihm vom Server-Betreiber übermittelt wird, in seinen Browser importieren.

Ähnlich funktioniert die Authentifizierung bei der elektronischen Abgabe der Steuererklärung per ☞ ELSTER.

Signieren mit OpenSSL

```
openssl dgst -sha256 -sign ssl.key -out datei.sig datei
```

Signatur prüfen mit OpenSSL

```
openssl dgst -sha256 -verify ssl.crt -signature datei.sig datei
```

SSL-Client-Verbindung aufbauen mit OpenSSL

```
openssl s_client -connect www.paypal.com:443
```

SSL-Server aufbauen mit OpenSSL

```
openssl s_server -key server.key -cert server.cert -accept 443 -www
```

Das Kommando startet einen Server-Dienst, der das WWW-TLS-Handshake unterstützt, auf dem Port 443 (Standard für https) des eigenen Computers. Es werden die zuvor generierten (`openssl req ...` Public-/Private-Key Paare verwendet. Mit `-www` (kleingeschrieben) wird der https-Handshake bis zum Aufbau der bidirektional verschlüsselten Verbindung durchgeführt, ohne weitere Funktionalität.

Mit der Option `-WWW` (großgeschrieben statt klein im obigen Beispiel) kann aber auch ein „richtiger" Webserver simuliert werden, der alle Dateien im aktuellen Verzeichnis per https an den Client übertragen kann.

Mit der Option **-msg** werden auf der Serverseite zusätzlich TLS Protokoll-Details als Hexdump ausgegeben.

Da Ports unter 1000 unter Unix privilegiert sind, muss openssl dort ggf. mit vorangestelltem **sudo** gestartet werden, oder ein höherer Port (z.B. 4430) angegeben werden, der dann im Browser entsprechend mit der URL

```
https://localhost:4430
```

kontaktiert wird.

Weitere gute OpenSSL-Kommandozeilen...

Z.B. **https://www.sslshopper.com/article-most-common-openssl-commands.html**

3.4.3 Passwortloses sicheres Login mit SSH-Keys

Die ☞ Secure Shell, welche oft auch als Proxy für Dienste wie *Secure FTP*, *git* oder *svn* verwendet wird, unterstützt parallel zum bekannten Passwort-basierten Login auch die Authentifizierung per *SSH Public Key*. Hierzu muss zunächst wieder ein Public-/Private Key Paar erzeugt werden. Auf der Unix-Kommandozeile:

```
ssh-keygen -t rsa
```

Der erzeugte Private Key kann optional mit einem Passwort verschlüsselt gespeichert werden, um ihn vor Diebstahl zu schützen.

Auf der Seite des SSH-Servers ist der clientseitig generierte Public Key nun in die Datei **authorized_keys**, die sich unter Unix im Verzeichnis **$HOME/.ssh** befindet, einzutragen.

```
ssh user@server-adresse mkdir -p -m 700 .ssh
cat .ssh/id_rsa.pub | ssh user@server-adresse tee -a .ssh/authorized_keys
```

Beim nächsten Aufruf von

```
ssh user@server-adresse
```

sollte nun nicht mehr das Unix-Passwort des Benutzers auf dem Zielrechner abgefragt werden, sondern allenfalls das Passwort, mit dem der Private Key auf dem Client-System verschlüsselt wurde.

3.4.4 Fazit

Fazit PGP und OpenSSL

Zwei Programme für die gleiche Funktionalität?

Obwohl PGP historisch gesehen die „ältere“ Software ist, hat sich SSL v.a. bei kommerziellen Produkten durchgesetzt. Dies mag auch mit der Tatsache zu tun haben, dass „Secure Socket Layer“ sich speziell für Web-Anwendungen und auch Streaming von Inhalten durchgesetzt hat, und das x509-Format für die Zertifikate auch nützliche und erweiterbare Meta-Informationen enthält, die in PGP Public Keys nicht vorgesehen sind. Mit S/MIME existiert ein auf SSL basierender Standard für Mail-Attachments, während PGP-MIME nicht standardisiert (wenn auch bei vielem Mailern implementiert) ist.

Speziell für die Verschlüsselung und Signatur von E-Mail sind PGP und S/MIME noch beinahe paritätisch vertreten, und viele Programme unterstützen, ggf. über Plugins, beide Methoden.

Folie 30

3.5 Datenträger-Verschlüsselung

„Festplatten“-Verschlüsselung

... soll v.a. sensitive Daten bei Diebstahl des Gerätes bzw. Datenspeichers (Smartphone mit privaten/Firmen-Daten, Notebook, auch PCs) vor Ausspähen schützen (Thema Vertraulichkeit).

- Datei-, Partitions- oder Datenträger-weise Verschlüsselung,
- i.d.R. symmetrische Blockcipher,
- der Schlüssel kann ein Passwort sein (Länge vs. Komfort, Sicherheit vs. Merkbarkeit), oder ein Binärschlüssel mit Zufallsanteil bei der Generierung, der selbst wieder durch ein Passwort geschützt in einem separaten Datenbereich gespeichert wird,
- verschiedene Software verfügbar, teils plattformneutral, teils OS-spezifisch.

Folie 31

Beispiel „Android"-Verschlüsselung (1)

- „Full Disk Encryption" gehört seit Android 5 zur (optional in den Einstellungen unter „Sicherheit" aktivierbaren) Standard-Ausstattung,
- bedeutet: „die **`/data`**-Partition wird verschlüsselt",
- symmetrische, AES-basierte Blockverschlüsselung mit 128Bit,
- der „Device Encryption Key" (DEK) wird in einem separaten Bereich auf dem Gerät mit einer PIN oder Passwort verschlüsselt gespeichert („KeyMaster"),
- bei „Hardware Encryption" übernimmt ein Teil des Chipsatzes („TrustZone", von der der KeyMaster ein Teil ist) die Ver- und Entschlüsselungsalgorithmen quasi als „Blackbox", von außen über eine API von signierten Applikationen gesteuert,
- Angriffspunkt: Vom Hersteller signierte Apps können die „TrustZone" zum Auslesen des DEK, Ver- und Entschlüsseln von Daten bewegen. Bei einigen Chipsätzen ist diese Schnittstelle ☞ unzureichend geschützt.

Folie 32

S.a. **`https://www.heise.de/security/meldung/Heftiger-Schlag-fuer-Android-Verschluesselung-3`**

Da der verschlüsselte DEK bei Hardware-unterstützter Verschlüsselung an das Gerät gebunden ist, ist eine „Offline Decryption" erst möglich, wenn der Key unverschlüsselt vorliegt. Bei Zerstörung des DEK sind die Daten nur noch per „Brute Force", also erraten des sehr langen symmetrischen Schlüssels, dechiffrierbar, was äußerst unwahrscheinlich ist. Eine mit Hardware-Unterstützung verschlüsselte SD-Karte ist so nicht mehr entschlüsselbar, wenn das Gerät, auf dem sie erzeugt wurde, defekt ist.

Beispiel „Android"-Verschlüsselung (2)

- Mit dem Android Debugger (ADB) können die Daten vom dem Gerät über USB-Kabel oder Netzwerk gesichert werden, so lange die verschlüsselte(n) Partition(en) entsperrt sind. Das Backup sollte natürlich auch wieder durch eine geeignete Maßnahme (z.B. **`pgp`** oder AES) geschützt werden.
- Im ausgeschalteten Zustand oder bei Sperre des Gerätes muss der DEK erst wieder durch Entschlüsseln aktiviert werden.
- Ein „Umverschlüsseln" oder eine komplette Entschlüsselung mit Zurückspeichern wird softwareseitig bislang nicht unterstützt. Es ist aber relativ leicht, die PIN oder das Passwort, mit der/dem der DEK selbst verschlüsselt wird, auszutauschen (unter Kenntnis der alten PIN).

Folie 33

Beispiel „Festplatten"-Verschlüsselung (1)

Übersicht: **https://de.wikipedia.org/wiki/Festplattenverschlüsselung**
Produkte (OS-spezifisch):

- Windows: ☞ EFS, ☞ BitLocker,
- Mac: ☞ FileVault
- Linux: dm-crypt (auch die bei Android verwendete Variante, das **/data**-Block Device (Flash-Partition) transparent zu ver-/entschlüsseln)

Folie 34

Beispiel „Festplatten"-Verschlüsselung (2)

Produkte (Multi-/Cross-plattform):

- Container-Formate mit Verschlüsselungs-Unterstützung, z.B. ☞ 7Zip,
- ☞ CrossCrypt: Zugriff auf verschlüsselte Linux Loop-AES-Partitionen unter Windows,
- VeraCrypt (Nachfolger von ☞ TrueCrypt
- Weitere, teils proprietäre Produkte: **https://de.wikipedia.org/wiki/Festplattenverschl%C3%BCsselung#Software**

Folie 35

Beispiel „Festplatten"-Verschlüsselung (3)

In Ländern, in denen Kryptographie verboten ist und um - falschen oder berechtigten - Verdächtigungen zu entkommen, ist bei Verschlüsselungsmechanismen der Begriff der „plausible deniability„ (Glaubhafte Abstreitbarkeit ein Sicherheits- und Qualitätskriterium. Dies bedeutet in Bezug auf die Festplattenverschlüsselung, dass ohne Kenntnis des Schlüssels nicht bewiesen werden kann, ob und welche Daten auf einer verschlüsselten Partition vorhanden sind, und nicht einmal, ob überhaupt verschlüsselt wurde. Der Besitzer kann also „glaubhaft bestreiten", im Besitz sensitiver Daten zu sein, was bei entsprechend geeigneten Mechanismen auch nicht widerlegt werden kann.

Folie 36

Bei dem unter Linux gebrächlichen **dm-crypt**-Mechanismus ist keine „Kennung" auf der Partition sichtbar, die nahelegt, es könnten sich verschlüsselte Daten darauf befinden. Mit einem Hex-Editor betrachtet, sieht eine solche Partition bzw. Container scheinbar wie eine Ansammlung völlig zufälliger, ungeordnter Daten aus.

Ein Beispiel unter Linux:

```
# Anlegen eines 1GB großen Containers mit Zufallsdaten
dd if=/dev/urandom of=crypt.img bs=1M count=1000

# Mapping herstellen, Passwort-Hash als Schlüssel
# für aes-cbc-essiv:sha256 block cipher
sudo cryptsetup open crypt.img geheim --type plain
 Passphrase eingeben:
```

Das EINMALIG (!) eingegebene Passwort bildet (mit sha256 gehasht) den symmetrischen Schlüssel für die AES-Blockchiffrierung. Im Verzeichnis **/dev/mapper** entsteht ein neues Block-Device mit dem Namen **geheim**, das die „entschlüsselte" Version der Daten enthält.

```
# Formatieren mit FAT (als Beispiel)
sudo mkdosfs /dev/mapper/geheim

# Mounten
sudo mkdir -p /geheim
sudo mount -o umask=000 /dev/mapper/geheim /geheim
```

Im Ordner /geheim können nun unverschlüsselt Dateien geschrieben und gelesen werden, die verschlüsselt im Container crypt.img landen.

Mit dem Hex-Editor kann im gemounteten Zustand der Inhalt des FAT32-Dateisystems im Klartext angezeigt werden:

```
00000000   EB 58 90 6D  6B 66 73 2E  66 61 74 00  02 08 20 00  02 00 00 00  .X.mkfs.fat... .....
00000014   00 F8 00 00  3F 00 FF 00  00 00 00 00  00 40 1F 00  CD 07 00 00  ....?........@......
00000028   00 00 00 00  02 00 00 00  01 00 06 00  00 00 00 00  00 00 00 00  ....................
0000003C   00 00 00 00  80 01 29 08  89 2B BC 4E  4F 20 4E 41  4D 45 20 20  ......)..+.NO NAME
00000050   20 20 46 41  54 33 32 20  20 20 0E 1F  BE 77 7C AC  22 C0 74 0B    FAT32   ...w|."t.
00000064   56 B4 0E BB  07 00 CD 10  5E EB F0 32  E4 CD 16 CD  19 EB FE 54  V.......^..2.......T
00000078   68 69 73 20  69 73 20 6E  6F 74 20 61  20 62 6F 6F  74 61 62 6C  his is not a bootabl
0000008C   65 20 64 69  73 6B 2E 20  20 50 6C 65  61 73 65 20  69 6E 73 65  e disk.  Please inse
000000A0   72 74 20 61  20 62 6F 6F  74 61 62 6C  65 20 66 6C  6F 70 70 79  rt a bootable floppy
000000B4   20 61 6E 64  0D 0A 70 72  65 73 73 20  61 6E 79 20  6B 65 79 20   and..press any key
000000C8   74 6F 20 74  72 79 20 61  67 61 69 6E  20 2E 2E 2E  20 0D 0A 00  to try again ... ...
```

... während im verschlüsselten Container weiterhin nur „Zufallsdaten" sichtbar sind:

```
00000000   54 EF C4 20  B6 1D C9 9A  64 61 8D A9  7D 6A 4C D8  6D 1E 9E 03  T.. ....da..}jL.m...
00000014   5C 24 80 05  D8 B0 81 F1  77 69 83 65  94 FF D3 44  89 97 4A 6F  \$......wi.e...D..Jo
00000028   4A 29 76 FC  F2 C3 9E E0  73 2E 17 66  09 6C 20 83  11 1E E8 0A  J)v.....s..f.l .....
0000003C   9E 81 FA B8  50 71 7D E0  8D 7F 40 3E  0B 93 93 FB  BC 28 DD 7C  ....Pq}...@>.....(.|
00000050   8A D0 98 3C  B3 05 94 42  9C 95 58 53  4A BF 26 F5  6A E8 C7 F4  ...<...B..XSJ.&.j...
00000064   CA 23 BF 14  96 6E 48 F3  B8 8C A1 30  9E 7A 2F 74  83 83 EC ED  .#...nH....0.z/t....
00000078   89 4A 32 25  B1 D8 CF FC  E0 2D 36 ED  77 46 E7 4F  21 5F ED B6  .J2%.....-6.wF.O!_..
0000008C   23 33 6B 9E  56 5C F1 A7  44 2D 41 1D  94 06 E1 A7  A6 43 C6 51  #3k.V\..D-A......C.Q
000000A0   C1 5C 12 EF  51 D4 D4 C5  7A 0A 71 E3  3C 3C 39 12  5C 73 22 E0  .\..Q...z.q.<<9.\s".
000000B4   CE 57 FC F6  72 A7 6E 08  7B F8 E6 7C  F2 44 7F B4  B6 B2 5E E6  .W..r.n.{..|.D....^.
000000C8   C2 3C 0D D2  DC F1 B5 28  22 65 FA B6  55 B0 70 E0  EE 8D 6C B7  .<.....("e..U.p...l.
```

Nach Beenden der transparenten Ver-/Entschlüsselung im Verzeichnis **`/geheim`**

```
sudo umount /geheim
sudo cryptsetup close geheim
```

sind die Daten im verschlüsselten Container sicher und lassen sich ohne Kenntnis des Passwortes auch nicht entschlüsseln.

Um den Container wieder „aufzuschließen", müssen die o.g. Kommandos (bis auf das Formatieren natürlich!) wiederholt werden.

`cryptsetup` unterstützt mit der Option **`--type tcrypt`** beim Öffnen übrigens auch das Verarbeiten von **TrueCrypt**- und **VeraCrypt**-Containern. Allerdings ist ein vorhandener Header im Container Voraussetzung. Das Anlegen neuer True- oder VeraCrypt-Container ist mit dm-crypt alleine nicht möglich.

Bestimmte andere Formate wie ☞ LUKS (Linux Unified Key Setup), die mit einer Kennung dem Betriebssystem die Verschlüsselung einer Partition signalisieren und so ein „halbautomatisches" Einbinden ermöglichen, erfüllen das Kriterium der *Glaubhaften Abstreitbarkeit* grundsätzlich nicht.

Weitere Systeme, die (entsprechende Vorkehrungen vorausgesetzt) das „Glaubhafte Abstreitbarkeit" Kriterium erfüllen, sind u.a.

- ☞ Tor (The Onion Router), ☞ Freenet,
- Dateiverschlüsselungssoftware ☞ FreeOTFE, ☞ VeraCrypt

Angriffspunkte „Festplatten"-Verschlüsselung

Angriff	Verteidigung
Ausspähen des symmetrischen Schlüssels	Nur „im Kopf" speichern oder Schlüssel sicher verwahren (z.B. verschlüsselter Container keepassX)
Erraten durch „Brute Force" (Ausprobieren)	Hohe Schlüssellängen, keine einfachen Passwörter mit „Wörterbücher"-Anteilen, keine „gelben Zettel" am Monitor aufkleben.
Abgreifen der Daten vor Verschlüsselung bzw. nach Entschlüsselung	Vermeiden von längerfristigen unverschlüsselten „Zwischenspeichern", möglichst direkt verschlüsselt speichern und erst beim Lesen entschlüsseln

Folie 37

Ein beliebtes Hilfsmittel von Angreifern ist der Einbau von System-Programmen, die sich zwischen die Verschlüsselungsroutine und die Datenträger setzten, und die die unverschlüsselten Daten abgreifen und übertragen. Solche Programme sind oft auch in Trojanern enthalten, die auch durch Geheimdienste und in der Strafverfolgung eingesetzt werden, um Verdächtige zu überwachen. Zur Installation ist mitunter die Hilfe des unbedachten Anwenders vonöten, der durch entsprechende Meldungen im Browser veranlasst wird, ein vermeintliches „Sicherheits-Update" herunterzuladen und als Administrator zu installieren.

Umgekehrt (Vorgriff auf das Thema „Verfügbarkeit") können Trojaner von starker Festplattenverschlüsselung Gebrauch machen, um die Anwender-Daten mit einem eigenen, dem Benutzer unbekannten, Schlüssel zu verschlüsseln, und diesen Schlüssel erst nach Zahlung eines „Lösegelds" dem Benutzer bekannt zu machen. Dabei ist es allerdings höchst unwahrscheinlich, dass der Anwender nach Zahlung tatsächlich wieder Zugriff auf die Daten erhält, da dies dem Angreifer keine nennenswerten Vorteile bringt, sondern ihn eher der Gefahr aussetzt, durch den Übermittlungskanal für den Schlüssel entdeckt zu werden. Dieser Attacke kann leicht begegnet werden durch häufige Datensicherungen auf einen (mit bekanntem Schlüssel verschlüsselten) externen Datenträger, allerdings mit dem Zustand VOR der Attacke.

Datei- vs. Geräteverschlüsselung

Bei dateiweiser Verschlüsselung kann für jeden Ordner oder jede Datei ein separater Schlüssel verwendet werden. Problem: Wie und wo werden diese Schlüssel gespeichert? Angriffspunkte: „Klartext-Attacke“, durch Dateinamen und andere Metadaten wie Größe und Position im Dateisystembaum kann auf den Inhalt einer Datei geschlossen werden, was „Brute Force“-Attacken erleichtert. Zum Zugriff auf eine Datei wird bei einigen Algorithmen oft eine unverschlüsselte Version der Datei im Speicher oder temporären Bereichen der Festplatte gehalten, um den Aufwand für die transparente Dechiffrierung gering zu halten, dort wären die Daten auslesbar.

Bei Geräteweiser (=Datenträger bzw. Partition oder Container-Image) Verschlüsselung wird der gesamte verschlüsselte Datenbereich blockweise verschlüsselt, und muss „als Ganzes“ aufgeschlossen werden. Es wird immer der gerade gelesene Block dechiffriert, und nur so viel unverschlüsselt im Speicher gehalten, wie von der Anwendung benötigt wird. Die Komplexität der Algorithmen ist geringer als bei der dateiweisen Verschlüsselung. Allerdings sind bei Verlust des Schlüssels weder die Dateisystem-Struktur, noch die Dateien selbst wiederherstellbar. Nicht einmal die Existenz oder Löschung von Dateien kann noch festgestellt werden.

Folie 38

Löschen von Daten?

Problem:

- System calls wie **`unlink(char *filename)`**... entfernen nur Einträge aus dem Inhaltsverzeichnis des Dateisystems. Gleiches gilt für „formatieren“.
- Datensegmente einer Datei bleiben i.d.R. erhalten (ggf. über den Datenträger „verstreut“,
- Auch ältere Versionen der Dateien bleiben erhalten,
- „Überschreiben“ wird vom Betriebssystem meist so gehandhabt, dass eine NEUE Datei angelegt wird, und die alte lediglich als „gelöscht“ markiert wird (d.h. der Platz ist wieder nutzbar, sobalt notwendig).

Lösungsansatz?

Folie 39

Sicheres Löschen von Daten

- Dateisysteme, die ein „in-place“ überschreiben der physikalischen Sektoren unterstützen,
- Datei erst „normal“ löschen, dann den „freien Platz“ des Datenträgers komplett überschreiben (durch eine riesige Datei),
- Datenträger von vornherein verschlüsseln, und den symmetrischen Schlüssel verwerfen, wenn der Datenträger außer Betrieb genommen wird (Achtung: Brute Force offline-Entschlüsselung ggf. eine Angriffsmöglichkeit)

Folie 40

Forensik

= gelöschte oder versteckte Daten (teilweise) wiederherstellen.
... Ergänzen. ...

Folie 41

4 Sicherheit im Netzwerk

Sicherheit im Netzwerk

Angriff auf den Datentransport und Verteidigungsmechanismen.

- Wiederholung Grundlagen ISO/OSI-Modell und TCP/IP (V4 und V6), Intra- und Internet, mit Fokus auf Schwachstellen,
- Netzwerk-Architektur und Netzwerktopologie unter Sicherheits-Aspekten,
- beispielhafte Maßnahmen wie Firewall (iptables), VLAN, 802.1x, ipsec, ...

Folie 42

4.1 Grundlagen TCP/IP unter Berücksichtigung von Angriffspunkten

ISO/OSI

☞ Das 7-Schichten Modell beim Datentransport über Netzwerke
NB: Auf den ☞ OSI-Layer 8 wird speziell bei Sicherheitsthemen auch oft eingegengen, man könnte dies als Zusammenfassung der „nicht-technischen Problemstellungen“ im Modell interpretieren. ;-)

Folie 43

TCP/IP

- TCP/IP
  - Entspricht nur teilweise dem ISO/OSI-Modell,

TCP/IP

Live-Folien während der Vorlesung: Insert here.
...

Netzwerke / Netzdienste erkennen (Scanning)

Beispiel Linux / Mac:

- Lokal: **`netstat -tulpen`**
- Remote: **`nmap -P0 -O -sT [-sU] ip-adresse(n)_opfer/netzmaske`**

NB: **`-P0`** bei **`nmap`** verhindert, dass ein ICMP Ping Request versandt wird, hiermit werden auch Adressen gescannt, die „nicht pingbar“ sind. **`-sU`** würde auch UDP-Ports scannen, was SEHR lange dauern kann. Mit **`-rtt_[max,initial]_timeout sekunden...`** kann länger auf Antwort gewartet werden, wenn ein Firewall oder Gateway aus Sicherheitsgründen eine „Verzögerung“ eingebaut hat.

Folie 46

Netzwerk-Pakete „sniffen“

Als ☞ Sniffing oder ☞ Snooping wird das Verfahren bezeichnet, Netzwerk-Pakete abzufangen, zu speichern und zu analysieren, die NICHT für die eigene IP-Adresse als Empfänger beabsichtigt sind.
Passive Sniffer (kaum erkennbar für den Rest des Netzwerkes):

- **`iptraf`**, **`etherape`** Traffic-Analyzer (Portbasiert)
- **`tcpdump`**: Primitiv aber effizienter IP-Logger

Hierfür wird der „Promiscuous Modus“ der jeweiligen Netzwerkkarte aktiviert, der auch Pakete empfangen kann, die nicht für die eigene IP gedacht sind.
Passiv: Es wird nur EMPFANGEN (gelauscht = geschnüffelt = sniffing)

Folie 47

Angriffe auf die Netzwerk-Übertragung (1)

Direkter Einbruch in die INnrastruktur:

- Kompromittierung der Infrastruktur: Accesspoint-Übernahme durch herstellerseitig installierte Services bzw. Hintertüren (☞ „Backdoor“ (Fern-Administration über unzureichend geschützte Service-Ports bei Netzwerk-Komponenten),
- Aufstellen eigener Accesspoints mit bekannter SSID aber ohne Authentifizierung/Verschlüsselung, bzw. Mobilfunk-Basen (☞ IMSI-Catcher),

Folie 48

Aktive Sniffer

Häufig eingesetzte „Man in the Middle“Attacken (auch **aktive** Sniffer, die Routing oder Pakete verändern):

- Umleiten von Paketen im LAN an Hardware-Adressen eigener Rechner per Arp-Poisoning ☞ **`ettercap`**, auch „feindliche Übernahme“ von Verbindungen möglich,
- Mitschneiden des Netzwerk-Datenverkehrs, auch verschlüsselt, zur späteren „Offline“-Auswertung ☞ **`wireshark`**.

Folie 49

S.a.

- ☞ „Backdoor“-Definition auf Wikipedia
- **`https://www.heise.de/security/meldung/Router-Backdoor-Cisco-Netgear-und-Linksys-vers`**
  Router-Backdoor in Komponenten von Cisco, Netgear und Linksys
- **`https://de.wikipedia.org/wiki/IMSI-Catcher`**
- **`https://de.wikipedia.org/wiki/Ettercap`**
- **`https://de.wikipedia.org/wiki/Wireshark`**

Das judikative Bestreben, die länderspezifische Verbreitung solcher Tools durch Illegalisierung zu verbieten, ist rein logisch gesehen nicht dazu geeignet, den unbedarften Anwender vor Kriminellen zu schützen. Ausgebildete Netzwerk-Administratoren und Sicherheits-Experten können die gleichen Tools jedoch sinnvoll einsetzen, um Angriffspunkte in der eigenen Netzwerk-Infrastruktur zu erkennen und Möglichkeiten zur Fehlerbehebung zu finden.

S.a. ☞ Diskussion über Sinn und Anwendbarkeit des sog. „Hacker-Paragraphen" § 202c StGB

Erkennung von Netzwerk-Manipulationen?

- Doppelte Hardware- oder IP-Adressen, häufige Verbindungsabbrüche (**`tcpdump`**-Analyse,
- Protokollanalyse: Wird eine unverschlüsselte Verbindung erzwungen? (Mobilfunk: **`SnoopSnitch`**)

Folie 50

Cross Site Scripting Angriffe (XSS)

(s.a. ☞ Wikipedia

- (Scheinbare oder echte) Manipulation einer an sich harmlosen Webseite, für den Anwender im Browser nicht erkennbar (Code injection),
- mit oder ohne Ändern von Daten auf dem Server zu diesem Zweck,
- Eigentlich kein direkter Angriff auf den Server, aber Ausnutzen von dessen Vertrauenswürdigkeit, möglicherweise überschreiben Serverseitiger Daten.
- „Unsichere Webformulare" bzw. Content-Management.Systeme.

Folie 51

Live-Beispiel

... einer an sich harmlosen und einfachen PHP Web-Applikation aus der Softwaretechnik-Vorlesung.

Folie 52

Vollständiger Code des „hochgradig gefährlichen“ Gästebuchs:

```
<HTML>
 <HEAD>
  <meta http-equiv="Content-Type" content="text/html; charset=utf8">
  <TITLE>Ein Gästebuch in PHP</TITLE>
 </HEAD>
 <BODY>

  <H1>Ein Gästebuch in PHP</H1>

  <H2>Neuer Eintrag:</H2>

  <!-- Das Formular zum eingeben einer Nachricht, mit Knopf zum Absenden -->
  <!-- Es ruft sich selbst wieder auf, daher kein "ACTION=...".          -->

  <FORM method=POST>
   Ihre Nachricht:<br>
   <textarea name="beitrag" rows=5 cols=40></textarea><br>
   <input type=submit value="Beitrag absenden">
  </FORM>

  <!-- Wenn das Formular bereits abgeschickt wurde, soll der Eintrag ins
       Gästebuch permanent gespeichert werden. -->
  <!-- Hierzu muss eine Datei geöffnet werden. Da der WWW-Server mit
       reduzierten Rechten läuft, sollte diese in einem für "Gäste"
       schreibbaren Bereich liegen, z.B. unter Windows in
       C:\Windows\TEMP\gaestebuch.txt -->

  <?php

    // Wenn ein Beitrag abgeschickt wurde -> Speichern in der Datei
    if($_POST[beitrag] != ""){
     // Die Datei muss für den Webserver schreibbar sein!
     // Windows:
```

```
    // $datei = fopen("C:\Windows\TEMP\gaestebuch.txt", "a");
    // Oder: eine schreibbare Datei im aktuellen Verzeichnis verwenden
    // Linux:
    // $datei = fopen("/tmp/gaestebuch.txt", "a");
    // z.B. .gaestebuch.txt
    $datei = fopen(".gaestebuch.txt", "a"); // "a" für "append" = "anhängen"
    // Trennzeichen (Linie) und Zusatz-Infos zum Besucher
    fwrite($datei, "<hr width=50% align=center>");
    fwrite($datei, $_SERVER[REMOTE_ADDR]);
    fwrite($datei, " schrieb ");
    fwrite($datei, date("r", $_SERVER[REQUEST_TIME]));
    fwrite($datei, ":<br>\n");
    // Beitrag
    fwrite($datei, $_POST[beitrag]);
    fwrite($datei, "\n");
    fclose($datei);
   }

 ?>

 <H2>Dies haben andere geschrieben:</H2>

 <!-- Wir wollen eigentlich nur Text-Postings erlauben, daher PRE.../PRE -->
 <PRE> <?php readfile(".gaestebuch.txt"); ?> </PRE>

</BODY>
</HTML>
```

Wo könnte das Problem liegen? Würde ein Ändern der Datei-Lokation von **`.gaestebuch.txt`** im Verzeichnis, in der auch die PHP-Datei liet, nach **`/tmp/gaestebuch.txt`** Abhilfe schaffen?

Empfehlungen des BSI

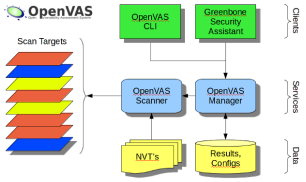
(Bundesamt für Sicherheit in der Informationstechnik)

- Empfehlungen zur „Cyber-Sicherheit“
- Netzwerk-Tool für Sicherheits-Beauftragte: ☞ Nessus Security Scanner (ehemals Open Source, Lizenzänderung durch die Urheber erlaubt nicht mehr die freie Verbreitung ☞ Open Source Fork OpenVAS (nächste Seite)

Folie 53

OpenVAS

Vom ☞ BSI empfohlen: Open Vulnerability Assessment System (OpenVAS)



Folie 54

Weitere Tools

- ☞ BSI: mapWOC
- ☞ OWasp XSS Testing

Folie 55

4.1.1 Eindringen von Schadsoftware in Netzwerke über Messaging / Mail

Problem: Fast alle Nachrichtendienste, angefangen mit der nach wie populären E-Mail, erlauben grundsätzlich das Senden „ohne Erlaubnis des Empfängers", lediglich das Senderbezogene Filtern („Blacklist") wird von den meisten Clients unterstützt, ist aber aufwändig und unzuverlässig. „Opt-In"-Mailinglisten sind hingegen eher freiwillig, auch wenn Landesrecht inzwischen unverlangte „Massenmails" untersagt (für Versender innerhalb Deutschlands).

Speziell bei E-Mail sieht weder das Container-Format, noch das SMTP-Protokoll eine Authentifizierung

und Verifikation des Absenders vor. Absenderadressen und die meisten Mailheader sind trivial zu „fälschen“, wenn man die Details über das Mailprotokoll kennt.

```
telnet localhost 25

HELO localhost
MAIL FROM: weihnachtsmann@nordpol.de
RCPT TO: knopper@knopper.net
DATA
From: Der Weihnachtsmann <weihnachtsmann@nordpol.de>
To: Klaus Knopper <knopper@knopper.net>
Subject: Ho Ho Ho

Test

.
```

SPAM / Mailfilter (auch Malware)

Ziel: Entfernen bzw. Quarantäne unerwünschter bzw. gefährlicher Inhalte

Probleme (1):

1. Erkennung von Schadsoftware und unerwünschten Inhalten anhand von Signaturen, die ständig in einer Datenbank aktualisiert werden müssen. „Neue“ Schadsoftware kann nur anhand eines angenommenen „Verhaltens“ identifizert werden, was eine schwierige Aufgabe ist.
2. Verschlüsselte Inhalte oder solche, die in Archiven mit Passwort (das dem Empfänger über andere Kanäle mitgeteilt wird) untergebracht ist, kann nicht untersucht werden.
3. Überlastung des Filters durch Massenmails und große Anhänge.

Folie 56

SPAM / Mailfilter (auch Malware)

Probleme (1):

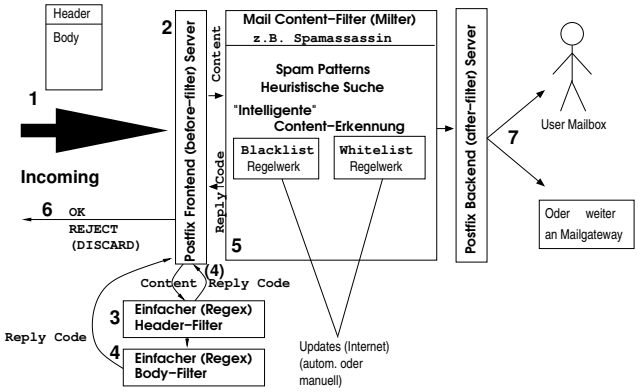
1. „False Positives“: Auch ungefährliche / wichtige Inhalte werden fälschlicherweise gefiltert, wenn eine zu einfach gestaltete Sigatur passt und erreichen den Empfänger nicht.
2. Arbeitsaufwand: In Quarantäne geschickte Inhalte müssen manuell untersucht werden, wenn sie nicht generell verworfen werden sollen.
3. Sollen die Absender-Adressen (die sich leicht fälschen lassen) darüber informiert werden, dass ihre E-Mail nicht ankam? Gefahr durch ☞ Backscatter

Lösungsansatz aus der Praxis (1)

„Sandwich-Konfiguration“ des Nachrichtenfilters:

1. Ein „Frontend“-Server nimmt die Nachricht zunächst an, sendet aber noch keinen Bestätigungs-Code für den „erfolgreichen Empfang“ sondern hält die TCP-Verbindung aufrecht.
2. Ein leistungsfähiger „Backend“-Server (Rechenleistung, Parallelverarbeitung, Speicher, temporärer Speicherplatz, ...) untersucht die Nachricht auf unerwünschte Inhalte oder Malware und meldet den Status der Untersuchung („OK“ oder gefundene Bedrohungen mit Fehlercode) an den Frontend-Server.
   (a) Bei „guten“ Nachrichten stellt der Frontend- (oder Backend-)Server die E-Mail zu, der Frontend-Server meldet einen Erfolgs-Code über die noch bestehende Verbindung zum Server und kann diese trennen (oder auf weitere Nachrichten über die gleiche Verbindung warten).
   (b) Bei „gefährlichen“ oder „unerwünschten“ Nachrichten meldet der Frontend-Server einen Fehlercode über die noch bestehende Verbindung zum Sender und trennt die Verbindung. Die Nachricht wird verworfen oder ein Quarantäne-Verzeichnis verschoben.

Lösungsansatz aus der Praxis (2)



Folie 59

Lösungsansatz aus der Praxis (3)

Beispiel: ☞ Postfix-Mailserver in Sandwich-Konfiguration
Hier: Der Spam- und Malware-Filter läuft zwischen erstem und zweitem Mailserver als sog. „milter“ (Mailfilter), nimmt Mails nur vom Frontend-Server an, der allerdings zuvor schon SMTP-Protokollbasiert und/oder mit Hilfe einfacher „Muster-Filter“ eine erste Zurückweisung unerwünschter Mails vornehmen kann. Der Backend-Server stellt schließlich die als „OK“ durchgegangenen Mails zu oder leitet sie an ein Mailgateway weiter. ☞ Für Technik-Fans: Konfigurationsbeispiel im Ordner **`postfix-config`**

Folie 60

Lösungsansatz aus der Praxis (4)

„Sandwich-Konfiguration“ des Nachrichtenfilters:

Vorteile:

- Der Sender erhält einen Fehlercode, der auch (bei SMTP) eine Klartext-Meldung erhalten kann, und erfährt daher, dass die Nachricht nicht zugestellt wurde.
- Backscatter wird vermieden, da keine Fehler-Nachricht neu generiert und an den ggf. gefälschten Absender aktiv zugesandt wird.

Nachteile:

- Der Backend-Server muss leistungsfähig genug sein, dass auch in Zeiten hohen Nachrichtenaufkommens innerhalb einer dienstspezifischen Timeout-Spanne kein Verbindungsabbruch während der Untersuchung der Nachrichteninhalte auftritt.
- Zwei Server notwendig, wodurch die Möglichkeit eines Software-Fehlers erhöht wird, durch den der Nachrichtenversand möglicherweise komplett zum Erliegen kommt.