Geheime Nachrichten am Computer und Internet:

Wie funktioniert Verschlüsselung in der digitalen Welt?

Klaus Kusche

Frühjahr 2026

Inhalt

- Händische Verschlüsselung
- Was verschlüsselt der Computer
  ... und wie?
- Das Hauptproblem am Internet
  ... und seine Lösung
- Sicherheit und Knackbarkeit
- Prüfsummen
- Die digitale Unterschrift
- Passwort-Speicherung

Händisches Verschlüsseln (1)

Ersetze jeden einzelnen Buchstaben

Beispiele:

- *"Ersetze jeden Buchstaben durch den Buchstaben, der im Alphabet 4 Plätze weiter hinten steht"*

  (Cäsar-Methode, sehr unsicher!)

- *"Schreib immer wieder das Schlüsselwort unter den Text und addiere die Buchstaben-Positionen im Alphabet"*

- *"Schreib das bereinigte Schlüsselwort unter das Alphabet und dann den Rest des Alphabets (verkehrtherum), nutze das als Ersetzungs-Tabelle"*

Händisches Verschlüsseln (2)

Ersetzen mit technischer Hilfe

Beispiel: *Enigma*

Mehrere ***verstellbare Walzen***

... mit wild *überkreuzten elektrischen Kontakten*

... verbinden *26 Tasten* mit *genausovielen Lämpchen*

Die Einstellung der Walzen ist der Schlüssel

==> Je nach Walzen-Stellung
leuchtet bei jedem Tastendruck
ein anderes Lämpchen für diesen Buchstaben auf

Händisches Verschlüsseln (3)

Ersetzung (von Buchstaben, Silben, Worten) mit Tabellen

Beispiele:

- Tabelle, die *jedes Buchstabenpaar* durch ein *anderes Buchstabenpaar* ersetzt
- Tabelle, die für *jeden Buchstaben mehrere mögliche Worte* angibt

  (mit denen man einen *lesbaren Text* schreibt ==> **Steganographie**)
- Tabelle, die jeder *Silbe* eine *Zahl* zuordnet

==> In Computer-Verschlüsselungen ***sehr selten***!

Händisches Verschlüsseln (4)

Durcheinandermischen der Positionen der Buchstaben

Beispiel:

Zuerst das 7., 14., 21., 28., ... Zeichen,
dann das 6., 13., 20., 27., ... Zeichen,
dann das 5., 12., 19., 26., ... Zeichen,
...
am Ende das 1., 8., 15., 22., ... Zeichen.

Händisches Verschlüsseln (5)

Verstecken der geheimen Nachricht in unverdächtigen Texten, Bildern usw.:

Beispiele:

- Geheimnachricht = 3. Buchstabe jedes Wortes
- Geheimtext mit unsichtbarer Tinte schreiben, normalen Text sichtbar drüberschreiben
- Für Zahlen: Anzahl der Buchstaben jedes Wortes
- Bestimmte Gegenstände, Formen oder Texte in einem Bild oder einer Landkarte

Händisches Verschlüsseln (6)

"Verschlüsseln durch Verstecken"

ist *nicht* Kryptographie,
sondern eine andere Wissenschaft:

Steganographie

==> *Nicht* in diesem Vortrag

==> Geht aber *am Computer sehr gut*,
wird *oft* verwendet

Beispiel: Beliebige Daten in *Bildern*:
Kaum zu erkennen!

Gute Computer-Verschlüsselungen

...sind eine ***eigene, sehr komplizierte Wissenschaft*:**

"Kryptographie"

ist ein Teilgebiet der ***Mathematik***
(*nicht* der Programmierung / Informatik)

Wenn es um die **ernsthafte Verschlüsselung** geht:

Niemals selber programmieren!

- *Allgemein anerkannte Verschlüsselungs-Verfahren* wählen
- *Weit verbreitete Programme* benutzen (bevorzugt *Open Source*)

Was wird verschlüsselt? (1)

- ***Alle Daten*** am Computer werden intern als (sehr) ***viele Bits*** dargestellt
- ***Ein Bit*** hat den Wert **0** oder **1** (z.B. *"Strom ein oder aus"*)

Beispiele (immer *Vielfache von 8*):

- Ein *Zeichen* Text ... 8 Bits
- Ein *Farbpunkt* eines Bildes ... 24 Bits
- Eine *Kommazahl* ... 64 Bits
- Eine Sekunde *CD-Musik* ... 1.411.200 Bits

Was wird verschlüsselt? (2)

==> Der Computer verschlüsselt immer

viele aufeinanderfolgende Bits

ohne zu wissen,
welche Art von Daten
diese Bits darstellen

==> Es ist der Verschlüsselung

völlig egal, ob sie
Texte, Programme, Bilder und Videos,
Zip-Files oder Webseiten verschlüsselt!

Was wird verschlüsselt? (3)

Verschlüsselungsverfahren arbeiten immer

mit ***gleichlangen Blöcken*** von Bits

(je nach Verfahren: *128 Bits ... 4096 Bits*)

==> Die Daten werden in solche Blöcke *aufgeteilt*,
die Blöcke werden *der Reihe nach* verschlüsselt

Intern ist auch

das ***"Codewort"*** = der ***"Schlüssel"***
ein ***Bitmuster fixer Länge*** *(meist derselben Länge)*

Für Menschen wird der Schlüssel als *Text*, als *Zahl*, als *QR-Code*, ... angezeigt

Wie wird verschlüsselt?

"Einfache" Verfahren:

- Arbeiten großteils auf **einzelnen Bits**:

  Bits ***verschieben / umordnen***

  und mit "***XOR***" zusammenrechnen

- … oder betrachten jeweils **1-8 Bytes (8-64 Bits) als Zahl**

  1 Byte = 8 Bits = als Zahl: 0…255 = als Text: 1 Zeichen

  Bytes sind für den Computer *besonders praktisch*!

- … und machen mit diesen Zahlen **einfache Rechnungen**

  z.B zwei solche Zahlen ***zusammenzählen*** oder ***vergleichen***

  (*keine* Multiplikation, Division oder Restrechnung)

Das XOR (1)

Das XOR hat für einzelne Bits die Rolle des *"Ersetzens von einzelnen Buchstaben"*:

***XOR* = *"exklusives Oder"* =**

"das eine oder das andere,
aber nicht beide"

0⊕0=0 1⊕0=1 0⊕1=1 1⊕1=0

"Das Ergebnis ist dann **1***,*
wenn genau eines der beiden Bits **1** *ist"*

Das XOR (2)

Mit XOR werden Bit für Bit zusammengerechnet

- Bestimmte (oder alle) Bits der *zu verschlüsselnden Daten*
- mit *genausovielen Bits* des *Schlüssels*

Eingabe:	"**Hallo!**"	`01001000 01100001 01101100 01101100 01101111 00100001`
⊕ Schlüssel:	"**CodeCo**"	`01000011 01101111 01100100 01100101 01000011 01101111`
= Geheimtext:		`00001011 00001110 00001000 00001001 00101100 01001110`
⊕ Schlüssel:	"**CodeCo**"	`01000011 01101111 01100100 01100101 01000011 01101111`
= Eingabe:	"**Hallo!**"	`01001000 01100001 01101100 01101100 01101111 00100001`

Zweimal XOR mit ***demselbem Schlüssel***
liefert ***wieder die Originaldaten!***

"Einfache" Verfahren? (1)

Diese Schritte sind **Grundlage aller "einfachen" Verfahren**,
z.B. ***AES = "Advanced Encryption Standard"*** *oder* ***"Twofish"***

"Einfache" Verfahren machen

viele Wiederholungen dieser Schritte!

Dafür ist die Hardware des Computers ***ideal***:

==> Geht mit Programmen ***einfach und sehr schnell***,
(**einige 100 Mio. Bytes** pro Sekunde)

==> Lässt sich als ***Spezial-Verschlüsselungs-Chip*** *bauen*
(noch schneller!)

==> Ist ideal für ***Verschlüsselung großer Datenmengen!***

"Einfache" Verfahren? (2)

"Einfache" Verfahren heißen wissenschaftlich

"Symmetrische" Verfahren

weil

Beide Seiten verwenden
denselben Schlüssel
zum Verschlüsseln und Entschlüsseln!

Gut für Verschlüsselung *lokaler Daten* *(z.B auf Laufwerken)*

Verschlüsseln & Entschlüsseln durch
dieselbe Person und/oder denselbem Computer

Verschlüsseln am Internet?

Am Internet: Sender und Empfänger

- sitzen ***weit auseinander*** auf ***verschiedenen Rechnern***
- müssten beide ***dasselbe Passwort*** kennen
- haben ***keine sichere Verbindung***, um sich ***geheim das Passwort*** zum Datenaustausch auszumachen
- wollen nicht für jeden Web-Zugriff ein Passwort eintippen

Henne-und-Ei-Problem:

Um Daten verschlüsselt übertragen zu können, brauche ich zuerst einmal eine verschlüsselte Verbindung zum Schlüssel-Austausch???

“Asymmetrische” Verschlüsselung

Geniale *Mathematiker* haben sich Verfahren ausgedacht, die dieses Problem lösen:

“Asymmetrische” Verschlüsselungen

Heißen so, weil sie

zum Verschlüsseln und Entschlüsseln zwei verschiedene Schlüssel verwenden

Die beiden Schlüssel

funktionieren nur gemeinsam, d.h. sie bilden ein “Schlüsselpaar”

Public & Private Key (1)

Der ***Empfänger*** der Daten (= der Webserver, Mailserver, ...)

*erzeugt **einmal** für sich ein **Schlüsselpaar***

- Einen Schlüssel ***behält er streng geheim für sich***:

  *Den **"Private Key"***

- Den zweiten Schlüssel ***veröffentlicht*** er (bzw. ***schickt ihn unverschlüsselt*** an jeden, der ihm Daten schicken will):

  *Den **"Public Key" ("öffentlichen Schlüssel")***

Deshalb auch der Name ***"Public-Key-Verfahren"***

Public & Private Key (2)

Grundprinzip:

- Die Daten werden mit dem ***öffentlichen* Schlüssel** des ***Empfängers* verschlüsselt**

  ==> ***Jeder*** *kann sich den öffentlichen Schlüssel holen und Daten für den Empfänger* ***verschlüsseln***

- Die Daten können nur mit dem ***privaten* Schlüssel *desselben* Schlüssel-Paares entschlüsselt** werden

  ==> ***Nur der Besitzer*** *des privaten Schlüssels (=Empfänger) kann die Daten wieder* ***entschlüsseln***

Programmierung (1)

Public-Key-Verfahren ...

- ... betrachten den **Datenblock** und die **Schlüssel**

nicht *als* ***einzelne Bits oder Bytes****,*

sondern als ***eine einzige riesige ganze Zahl***

Typische **Blockgröße** und **Schlüssellänge**:

4096 Bits

==> **Länge als normale Zahl** angeschrieben:

1233 Ziffern!!!

Programmierung (2)

- ... machen mit diesen langen Zahlen

aufwändige Rechenoperationen:

Multiplikation, Hoch-Rechnung,
Division** und vor allem **Divisions-Rest

==> Asymmetrische Verfahren sind daher

- ... viel ***langsamer*** *(viele tausend Mal!)*
  als symmetrische Verfahren
- ... zu kompliziert für eigene Spezial-Chips
- ... ***nicht für Massendaten geeignet!***

Anwendung am Internet

Schlüsseltausch-Verfahren

(z.B "**`https://`**..." = ***"Diffie-Hellman-Verfahren"***):

- Zuerst wird mit einer ***asymmetrischen Verschlüsselung*** der ***Schlüssel*** *für die* ***symmetrische Verschlüsselung*** vereinbart

  Der symmetr. Schlüssel ist *kurz* ==› das darf langsam sein

- Dabei geht jedesmal eine ***Zufallszahl*** mit ein

  ==› Jedesmal ein ***anderer symmetrischer Schlüssel***

- Die ***eigentlichen Daten*** werden danach ***symmetrisch*** verschlüsselt **==› schnell!**

Mathematik (1)

Asymmetrische Verfahren beruhen
auf **komplexen mathematischen Problemen**
aus der **Zahlentheorie** (vor allem: Lehre von Primzahlen),
sogenannten

"Einweg-Funktion" *bzw.*
"Falltür-Problemen"

... die ***in eine Richtung "leicht"*** zu berechnen sind

==> diese Richtung dient zur
Berechnung eines neuen Schlüsselpaares

Mathematik (2)

... aber in die ***umgekehrte Richtung sehr schwierig***
oder ***nur durch Durchprobieren*** zu lösen sind

==> die ***umgekehrte Richtung*** bräuchte man
für das ***Brechen der Verschüsselung*** = für die

Berechnung des Private Keys
aus dem Public Key

Die ***"Nicht-Knackbarkeit"*** beruht auf der
praktischen Nicht-Berechenbarkeit
einer ***mathematischen*** Aufgabe!

Mathematik (3)

Verfahren 1: *Faktorisierung großer Zahlen*

Wenn *p* und *q* *zwei sehr große* ***Primzahlen*** sind (einige hundert Stellen),

- dann ist $n = p * q$ "*leicht*" zu berechnen,
- aber die Berechnung von *p* und *q* aus *n*
  ist zwar theoretisch bekannt und im Prinzip eindeutig lösbar,
  ist aber *praktisch unmöglich*:

  Bestmögliche Laufzeit der Faktorisierung

  von Zahlen der Länge *L*: $\mathbf{c * e^{\sqrt{(L * \log(L))}}}$

Mathematik (4)

Beispiel:

98639 * 85027 = ?

8386978253

kann man mit etwas Mühe händisch rechnen.

Aber

*“sag mir alle Teiler von **8386978253**”*

ist händisch ***aussichtslos***!

Und jetzt dasselbe mit 1200-stelligen Zahlen…

Mathematik (5)

Verfahren 2: *“Diskreter Logarithmus”*

a^x kongruent m mod p

für a, x, m, p **ganzzahlig**

- *Leicht* für a, x, p gegeben und m gesucht
- *Sehr schwer*

  für a, m, p gegeben
  (sorgfältig gewählt, u.a. p prim, und groß!)

  und kleinstes lösendes x gesucht

Mathematik (6)

Beispiel:

"Welcher Rest bleibt,
*wenn ich vierzehn Mal die Zahl **7** zusammenmultipliziere*
*und dieses Ergebnis durch **17** teile?"*

geht noch *händisch* (*eine* Restrechnung) und ergibt Rest **8**

Aber

*"Wie oft muss ich **7** mit sich selbst multiplizieren,*
*damit das Ergebnis geteilt durch **17** den Rest **8** ergibt?"*

geht nur mit *durchprobieren*: **7/17**, **(7*7)/17**, **(7*7*7)/17**, ...
==> 14 Restrechnungen!

Praktische Verfahren

Faktorisierung:

RSA-Verschlüsselung

(die Erfinder der Public-Key-Verfahren,
sind damit berühmt und reich geworden)

Diskreter Logarithmus:

Diffie-Hellman-Verfahren

(Grundlage von *SSL/TLS = verschlüsselten Internet-Verbindungen*)

Elgamal

(immer schon *lizenzfrei*, Grundlage von *PGP Mail-Verschlüsselung*)

Sichere Verschlüsselungen

Verschlüsselte Daten schauen aus wie

"zufälliger Bitsalat"

- **Keine regelmäßigen Muster**
- **Statistisch "unauffällig"**: `0` *und* `1` *in etwa gleich oft, ...*

  (Problem bei Hand-Verschlüsselung:
  z.B. häufigstes Zeichen = meistens verschlüsseltes 'e'):
- **Kein erkennbarer Zusammenhang**
  zwischen einzelnen Bits im **Input** und im **Output**
- "**Lawineneffekt**": 1 Bit in den Originaldaten ändert sich
  ==> *Viele Bits in den verschlüsselten Daten ändern sich!*

Angriffe (1)

Entweder die

Originaldaten herausfinden

... oder (noch schlimmer) den

Schlüssel herausfinden!

Bei händischen Verfahren: Das *Verfahren herausfinden?*

*Das **Verfahren am Computer** ist meist*
öffentlich bekannt**, nur der **Schlüssel** ist **geheim!

Weil: Verfahren = Programme lassen sich
am Computer nicht wirklich geheim halten!

Angriffe (2)

Besonders gemein:

*Der Angreifer kennt die **verschlüsselten Daten***
*und die **dazugehörigen Originaldaten**!*

Oder noch schlimmer:

*Der Angreifer hat **die Krypto-Box geklaut***

==> Er kann selbst *beliebig viele Paare*
Originaldaten & verschlüsselte Daten erzeugen

Trotzdem:

Der Schlüssel darf daraus nicht berechenbar sein!

Unknackbar? (1)

Es gibt ***kein*** "**theoretisch unknackbares**" **Verfahren:**

"Durchprobieren aller Möglichkeiten"
funktioniert theoretisch immer!

Aber selbst bei *nur 128 Bit Schlüssel-Länge*:

2^{128} = **340282366920938463463374607431768211456**
Möglichkeiten für den Schlüssel

"Alles Durchprobieren" dauert viel zu lange!

==> Praktisch unknackbar!

Unknackbar? (2)

Für die Kryptologen gilt ein Verfahren als ***geknackt***,
wenn das Herausfinden der Originaldaten oder des Schlüssels

- durch irgendeine ***direkte Berechnung (ohne Probieren)*** (selten)
- oder ***deutlich weniger Probier-Versuche als 2^n (bei n Bit Schlüssel)*** = *"intelligentes Probieren"*

möglich ist.

Beispiel AES mit 128 Bit Schlüssel:

Alle bekannten Attacken liegen immer noch **über 2^{100}**
==> AES gilt (gerade) noch als sicher.

Unknackbar? (3)

Weitere Gefahr:

"Seitenkanal-Attacken"

- Messung feinster **Zeit-Unterschiede**
- Messung von **Stromverbrauch, Funkabstrahlung,** *"Hot Spots"* usw. des Prozessors
- Analyse der **Cache- und Speicherzugriffe**

==> Das Programm oder der Krypto-Chip muss von ***außen*** beobachtet ***immer "gleich" rechnen,*** unabhängig von Originaldaten und Schlüssel!

Alles bestens für's Internet?

Nein!!!

- "Rede" ich mit dem *richtigen / echten Server*?
- Stammt das heruntergeladene Update wirklich *vom Hersteller*?
- Hat niemand in das Update nachträglich *Schadcode eingebaut*?
- Ist die Mail *unverfälscht*?
- Ist der *Absender* der Mail echt?
- ...

Prüfsummen = Hashes (1)

Berechnen aus beliebig langen Bit-Folgen

eine Prüfsumme fixer Länge (z.B. 512 Bits)

Aber: ***Es ist kein Schlüssel beteiligt!***

Haben ***sehr viele Anwendungen***,
mehr als die echten Verschlüsselungs-Verfahren!

Rechnen ***ähnlich wie symmetrische Verfahren***

==> sind auch *auf großen Datenmengen sehr schnell*

==> sind relativ *einfach programmierbar*
oder *in Hardware zu bauen*

Prüfsummen = Hashes (2)

Sind nicht rückrechenbar (dürfen sie nicht sein!),

d.h. die Originaldaten
sind daraus nicht mehr herstellbar

==> Eine Art ***"Einweg-Verschlüsselung"***
mit gleichen Anforderungen wie bei der Verschlüsselung:

- Liefern ***"zufälligen Bitsalat"***
- Sind immun gegen statistische Analysen
- Haben denselben ***"Lawineneffekt"***
- Erfüllen dieselbe theoretische ***"Knackbarkeits"-Bedingung***:
  -- Knacken im Idealfall nur durch ***Durchprobieren aller Inputs***
  -- Geknackt, wenn mit deutlich unter 2^n Versuchen knackbar

Knacken von Hashes

Zwei Dinge dürfen ***nicht "schnell" lösbar*** sein:

- ***Erzeuge gezielt eine Datei mit einem vorgegebenen Hashwert***
  (Beispiel: Baue gezielt eine *manipulierte / gefälschte Datei* mit *gleicher Prüfsumme* wie die Original-Datei)
- ***Erzeuge gezielt zwei Dateien mit gleichem Hashwert***

Sonst:

- ***Fälschen*** *von Software, Dokumenten, …*
- ***Vortäuschen*** *falscher Identitäten*
- ***Knacken*** *von Passwörtern*

Die digitale Unterschrift (1)

= *die "elektronische Signatur"*

- stellt sicher,
  dass ***eine bestimmte Person am Computer*** sitzt
- stellt sicher, dass ***eine Mail / eine Datei***
  - von einem ***bestimmten Absender*** kommt
    (und von niemand anderem!),
  - dass sie ***unverfälscht*** ist,
  - und dass der Absender ***nicht abstreiten kann***,
    diese Mail / Datei unterschrieben zu haben

Die digitale Unterschrift (2)

Die elektronische Signatur ist

auch ein "Public-Key-Verfahren",
aber "verkehrtherum":

- Verwendet wird das ***Schlüsselpaar des Absenders***
  (= Erstellers der Datei), nicht des Empfängers
- ***Signiert*** wird mit dem ***"Private Key"***
  ==> ***Nur der Schlüsselpaar-Besitzer*** *kann signieren*
- Die Signatur wird mit dem ***"Public Key" geprüft***
  ==> ***Jeder*** *kann die Signatur überprüfen*

Die digitale Unterschrift (3)

	Verschlüsselung	**Signatur**
Senden (Verschlüsseln / Signieren)	***Öffentlicher*** *Schlüssel des* ***Empfängers***	***Privater*** *Schlüssel des* ***Senders***
Empfangen (Entschlüsseln / Signatur prüfen)	***Privater*** *Schlüssel des* ***Empfängers***	***Öffentlicher*** *Schlüssel des* ***Senders***

Die digitale Unterschrift (4)

"Public Key Verfahren" sind **langsam**

==> Signieren der **Originaldaten** wäre **zu aufwändig**

==> Signiert wird fast immer nur

der Hashwert der Originaldaten

Bei **unsicherer Hashfunktion** könnte man ein ***zweites Dokument mit selbem Hashwert*** basteln!

==> Die Signatur wäre auch
für das zweite Dokument gültig!!!

Zertifikate (1)

Die Signatur beweist nur,

dass die Daten vom ***Besitzer (oder vom Dieb!) des private Key*** signiert wurden,

aber ***nicht, welche reale Person / Firma*** das ist!

Für den Zusammenhang Schlüsselpaar ‹==› Person:

Komplizierte Struktur von ***Zertifikaten***
PKI = "Public Key Infrastructure"
CA = "Certificate Authority"

Mehr organisatorisches als technisches Problem...

Zertifikate (2)

Weitere ***wichtige Anwendung von Zertifikaten***
(Klick auf das ***Schloss der URL-Zeile*** im Browser):

Zertifikate sollen sicherstellen,
dass der Webserver,
zu dem ich mich verbinde,
wirklich zu der Firma (Bank, ...) gehört,
zu der ich will!

Das (signierte!) Zertifikat sagt fälschungssicher,

zu welcher Internet-Domain das Schlüsselpaar dieses Webservers gehört!

Passwort-Speicherung (1)

Unterscheide **zwei Fälle:**

- Das ***ursprüngliche Passwort***
  muss ***wiederherstellbar*** sein
  (z.B. Passwort-Tresor im Web-Browser)

  ==> Normale ***symmetrische Verschlüsselung***,
  verschlüsselt mit einem Master-Passwort
  (oder dem Login-Passwort)

- Das ***eingegebene Passwort***
  muss ***nur geprüft*** werden

  ==> Passwort wird ***gehasht, nicht verschlüsselt!***

Passwort-Speicherung (2)

Idee der Passwort-Speicherung mit Hashes:

- Beim ***Festlegen des Passwortes*** merkt sich der Webserver / das Windows, Linux, ...

  nur den Hashwert *des neuen Passwortes*

- Beim ***Anmelden mit dem Passwort*** prüft der Webserver / das Windows, Linux, ...

  ob der ***Hashwert des eingetippten Passworts*** *und der* ***gespeicherte Hashwert gleich*** *sind*

==> *Dann war auch das Password gleich!*

Passwort-Speicherung (3)

Vorteil dieser Methode:

Auch wenn jemand die Datei
mit den ***gehashten Passwörtern klaut***

... kann sich der Dieb mit den Daten in der Datei
nicht anmelden

... kann der Dieb aus der gestohlenen Datei
nicht die Original-Passwörter berechnen

(aber es gibt leider immer noch viele Server am Internet,
die Passwörter völlig unverschlüsselt im Original oder mit
einer untauglichen symmetrischen Verschlüsselung speichern!)

“The End”

Fragen?