

Dokumentation
„Protokolle höherer Schichten“
oder
„Kryptographie für Anfänger“

Martin Amelsberg - Amel@AmelFin.de

Daniel Finger - Fin@AmelFin.de

Thorsten Maruhn - Thorsten@TMaruhn.de

Torsten Bloth - Torsten@Bloth.net

Sommersemester 2003, FH OOW Emden, <http://www.technik-emden.de>

Version dieser Dokumentation: 1.00

Inhaltsverzeichnis

1. Einleitung	
1.1. Auszug aus dem Grundgesetz.....	4
1.2. Warum Verschlüsselung?.....	4
1.3. Unterschiedliche Verschlüsselungsmethoden.....	4
2. Grundlagen	
2.1. Sichere Passworte.....	5
2.2. Modulo-Rechnung und Kongruenz.....	7
2.3. Große Exponenten.....	8
2.4. Primzahlen.....	9
2.5. Zufallzahlen.....	9
2.6. Symmetrische und asymmetrische Verschlüsselung.....	10
3. Primzahlen	
3.1. Auffinden von Primzahlen.....	11
3.2. Unterschiedliche Verfahren.....	11
3.3. Ein einfacher Ansatz.....	11
3.4. Die Probedivision.....	11
3.5. Satz des Fermat.....	12
3.6. Der Rabin-Miller-Test.....	14
3.7. Anmerkung.....	16
4. Symmetrische Verschlüsselung	
4.1. Einleitung.....	17
4.2. SSL.....	17
4.2. DES und AES.....	17
5. Asymmetrische Verschlüsselung	
5.1. Einleitung.....	18
5.2. Der RSA-Algorithmus	
5.2.1. Die Sicherheit des RSA.....	18
5.2.2. Ein ausführliches Beispiel.....	19
5.2.3. Ein (nicht ganz so) ausführliches Beispiel.....	24
5.2.4. Abschließende Anmerkungen.....	26
6. Hybride Verschlüsselung	
6.1. Warum „hybrid“?.....	26
6.2. Ablauf einer hybriden Verschlüsselung.....	26
7. Schlüsselerzeugungs-Verfahren	
7.1. Einleitung.....	27
7.2. Diffie-Hellman: Der Ablauf.....	27
7.3. Ein Beispiel.....	28
7.4. Warum ist „ $k=k$ “?.....	28
7.5. Brute Force.....	29

8.	Signaturen	
8.1.	Einleitung.....	30
8.2.	Eine einfache Methode.....	30
8.3.	Verbesserung: die HASH-Funktion.....	31
8.4.	Bekannte HASH-Funktionen.....	32
8.5.	Blinde Signaturen	
8.5.1.	Einleitung.....	33
8.5.2.	Ein anschauliches Beispiel.....	33
8.5.3.	Blinde Signatur - mathematisch.....	33
9.	PGP	
9.1.	Einleitung.....	34
9.2.	Warum zwei Schlüssel?.....	34
9.3.	ElGamal	
9.3.1.	Einleitung.....	35
9.3.2.	Der grundsätzliche Aufbau.....	35
9.3.3.	Ein einfaches Beispiel - Schritt für Schritt.....	36
10.	Base64	
10.1.	Einleitung.....	39
10.2.	Das Base64-Alphabet.....	39
10.3.	Das ASCII-Alphabet.....	40
10.4.	Beispiel 1.....	41
10.5.	Beispiel 2.....	42
10.6.	Beispiel 3.....	43
11.	Blinde Signatur (am Beispiel von E-Cash)	
11.1.	Anmerkung.....	45
11.2.	E-Cash.....	45
12.	Kurz und knapp	
12.1.	Anmerkung zu diesem Kapitel.....	46
12.2.	Was ist ein Gateway?.....	46
12.3.	Was ist ein Proxy?.....	47
12.4.	Was ist eine Firewall?.....	47
12.5.	Was ist TCP?.....	47
12.6.	Was ist SNMP?.....	48
12.7.	Was ist ein VPN?.....	48
12.8.	Was ist HTTP/FTP?.....	49
12.9.	Was ist SMTP?.....	49
13.	Über dieses Dokument	
13.1.	Sinn dieser Dokumentation.....	50
13.2.	Nutzungsrecht.....	50
13.3.	Aktuelle Version.....	50

1. Einleitung

1.1. Auszug aus dem Grundgesetz

Artikel 10, Absatz 1:

Das Briefgeheimnis sowie das Post- und Fernmeldegeheimnis sind unverletzlich.

1.2. Warum Verschlüsselung?

Schon seit vielen hundert Jahren werden geheime Botschaften oder andere sensible Daten verschlüsselt. Hierbei ging es meist darum, den politischen Gegner oder den Konkurrenten nicht über die eigenen Schritte zu informieren.

Im Laufe der Zeit wurde die Verschlüsselung immer wichtiger, da es viele Angreifer (auch Hacker, Cracker, Cyber-Piraten usw. genannt) gibt, die (fast) jede Art von Information sammeln und auswerten wollen. Und dies nicht nur im politischen Bereich und bei Firmen, sondern auch im privaten Umfeld.

Was kann ein Angreifer verwerten? Nun, dies sind nicht nur offensichtliche Daten wie Geheimnummern, Passworte und Zugangscodes, sondern es fängt schon bei persönlichen Gewohnheiten (Welche Webseiten werden besucht? Welche Firmenmarken für Nahrung, Kleidung usw. werden bevorzugt?) an, die einen Aufschluss darüber geben können, mit welcher Werbung der Ausspionierte ggf. konfrontiert werden kann.

Für einen Hersteller für Hundenahrung ist es durchaus interessant zu wissen, ob Person X überhaupt einen Hund hat. Frau Y ist nicht verheiratet, also wäre eine Werbung für Herrenmoden für sie voraussichtlich uninteressant.

Des Weiteren können einzelne Personen-Daten (Geburtstag, Name des Ehepartners, Name des Hundes,...) dazu genutzt werden, um Geheimcodes und auch Passworte zu erraten. Leider nehmen viele Computernutzer die Wahl des Passwortes nicht sonderlich ernst, so dass dieses ggf. leicht herauszubekommen ist. Aber dazu später mehr.

1.3. Unterschiedliche Verschlüsselungsmethoden

Je nach Verwendungszweck stehen dem Anwender eine Vielzahl von Verschlüsselungsmethoden zur Verfügung. Dies beginnt schon bei einer sehr einfachen Verschlüsselung, wie der Verschiebung in der ASCII-Tabelle, geht über das XOR-Verfahren bis hin zu Systemen mit mehreren Schlüsseln.

Einige dieser Systeme sollen in diesem Dokument (größtenteils mit einfachen und ausführlichen Beispielen) erklärt und auf ihre Sicherheit geprüft werden.

2. Grundlagen

2.1. Sichere Passworte

Um Daten schützen zu können, muss sich der Anwender in den meisten Fällen ein Passwort erdenken (und merken), das dem Verschlüsselungsalgorithmus übergeben wird.

Leider geben sich hierbei sehr viele keine Mühe und nehmen den Namen des Ehepartners, das Geburtsdatum oder eine Telefonnummer. Nicht nur einem Bekannten des Angegriffenen, sondern durch das Ausspionieren von eMails usw. ist es auch einem Fremden möglich, solche Passworte zu erraten. Im schlimmsten Fall muss der Eindringling einen sogenannten „Brute Force“-Angriff¹ starten, doch auch dieser ist ggf. schnell erfolgreich.

Beispiele:

- a) Person X verschlüsselt einen Text mit einer 6-stelligen Zahlenkombination und ist sich sicher, dass niemand diesen Geheimcode „knacken“ kann, da es weder eine Telefonnummer, noch (s)ein Geburtsdatum ist.

Nehmen wir an, dass wir es mit den Ziffern 0-9 zu tun haben, so dass sich im Höchstfall 10^6 Möglichkeiten ergeben, also die Zahlenkombinationem 000000, 000001, ..., 999998, 999999.

Ein durchschnittlicher Computer schafft heutzutage durch optimierte Algorithmen 1 Millionen Versuche² pro Sekunde, so dass schnell die „geheime“ Zahlenkombination von Person X ermittelt ist.

- b) Person Y verschlüsselt einen Text mit einer 7-stelligen Kombination aus wahllos gewählten Kleinbuchstaben.

Hieraus ergeben sich im Maximum $26^7 = 8031810176$ Möglichkeiten, was auf den ersten Blick nach einer sehr großen Zahl aussieht.

Im Schnitt muss ein Angreifer die Hälfte aller möglichen Kombinationen ausprobieren, somit hier 4015905088. Bei 1 Millionen Versuche pro Sekunde ergeben sich rund 4015 Sekunden, also etwas mehr als 1 Stunde.

- c) Person Z verschlüsselt einen Text mit einem langen Wort, da sie es sich gut merken kann und sie sich sicher ist, dass 10 Buchstaben ausreichen.

¹ Dieses Verfahren bezeichnet einen Angriff, bei dem alle möglichen Kombinationen ausprobiert werden.

² Durch die Optimierung der Algorithmen und Vernetzung von Computern können durchaus 10^{15} Schlüssel pro Sekunde getestet werden; der Aufwand und die Kosten hierfür sind natürlich sehr hoch, also sollte sich der Angriff auch „lohnen“

Nun, für einen „Brute Force“-Angriff sind 26^{10} durchaus schwer zu knacken, doch ein sogenannter Wörterbuch-Angriff ist hier sehr effektiv. Es müssen nur alle Worte aus einem umfangreichen Wörterbuch ausprobiert werden.

Man sieht, dass ein sicheres Passwort gar nicht so leicht zu finden ist. Die Problematik besteht darin, dass der Anwender sich das Passwort merken muss und dies fällt bei einer kryptischen Zeichenfolge, wie bspw. „3aJl7LX9\$K4z“ sehr schwer, obwohl gerade diese Kombination aus Kleinbuchstaben, Großbuchstaben, Zahlen und ggf. auch Sonderzeichen „sicher“ ist.

Eine Möglichkeit, sich einen (relativ) sicheren Geheimcode zu erstellen ist es, einen einfachen Satz zu überlegen (und zu merken) und die Anfangsbuchstaben als Code zu nutzen. Idealerweise besteht dieser Satz

- aus 10 oder mehr Worten
- aus (klein geschriebenen) Verben und (groß geschriebenen) Nomen
- nicht nur aus Buchstaben, sondern beinhaltet auch eine oder mehrere Zahlen.

Satz : „Das große A hat als ASCII den Ziffern-Wert 65“.
Code : „DgAhaAdZW65“

Versucht nun ein Angreifer den Code zu knacken, bleibt ihm nichts anderes übrig, als einen „Brute Force“-Angriff zu starten, der jedoch bei dem o.g. Code mit insgesamt $62^{11}=52036560683837093888$ Möglichkeiten¹ wenig erfolgreich sein wird. (Hinweis: Sprichworte und Redewendungen sollten hierbei wiederum nicht genutzt werden, da die entstehenden Codes im Prinzip mit einem Wörterbuchangriff² geknackt werden können.)

Somit ist klar, dass die Sicherheit einer Verschlüsselung nicht nur vom verwendeten Algorithmus, sondern auch sehr stark von der Schlüssellänge abhängt.

Im Gegensatz zu früheren Verschlüsselungen reichen heutzutage 40-Bit (12 Stellen) nicht mehr aus, da sie je nach Aufwand und Kosten innerhalb von wenigen Sekunden „geknackt“ werden. Erst ab ca. 70 bis 80 Bit muss ein dermaßen hoher Aufwand betrieben werden, dass sich die unerlaubte Entschlüsselung meist gar nicht lohnt. Bei den heute üblichen 128 Bit kann man von „echter Sicherheit“³ sprechen, sofern der Algorithmus keine Schwachstellen besitzt.

¹ bei 1 Mio. Versuchen pro Sekunde ergeben sich im Schnitt ca. 82 500 Jahre für diesen Angriff!

² es existieren auch Wörterbücher mit Redewendungen, Floskeln usw.

³ Vorsicht mit dem Ausdruck „echter Sicherheit“, denn wer weiß schon, was in 10, 50 oder 100 Jahren für Computer eingesetzt werden? Vor einigen Jahren galten 40 Bit auch also „absolut sicher“.

2.2. Modulo-Rechnung und Kongruenz

Die Modulo-Rechnung ist eine Ganzzahl-Division, bei der der Rest ebenfalls aufgeschrieben wird:

$$\begin{array}{ll} 13 : 6 = 2 \text{ Rest } 1 & \text{da } (6 \cdot 2) + 1 = 13 \\ 50 : 8 = 6 \text{ Rest } 2 & \text{da } (8 \cdot 6) + 2 = 50 \\ 3514 : 13 = 270 \text{ Rest } 4 & \text{da } (270 \cdot 13) + 4 = 3514 \end{array}$$

Die **Restklasse** beschreibt die Menge von Zahlen, die ein bestimmtes n als Reste hat, also $\{1, \dots, n-1\}$

$$\begin{array}{l} n=5 \rightarrow \text{Restklasse} = \{1, 2, 3, 4\} \\ n=11 \rightarrow \text{Restklasse} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} \end{array}$$

Ein weiterer Begriff ist der der **Kongruenz**: Zwei Zahlen m und n heißen „kongruent modulo p “ genau dann, wenn gilt

$$m \bmod p = n \bmod p$$

Man schreibt

$$m \equiv n \pmod{p}$$

Beispiele:

- a) $25 \equiv 4 \pmod{7}$, da gilt: $4 : 7 = 0 \text{ Rest } 4$
 $25 : 7 = 3 \text{ Rest } 4$

- b) $555 \equiv 74 \pmod{13}$, da gilt: $74 : 13 = 5 \text{ Rest } 9$
 $555 : 13 = 42 \text{ Rest } 9$

- c) $835 \equiv 52 \pmod{27}$, da gilt: $52 : 27 = 1 \text{ Rest } 25$
 $835 : 27 = 30 \text{ Rest } 25$

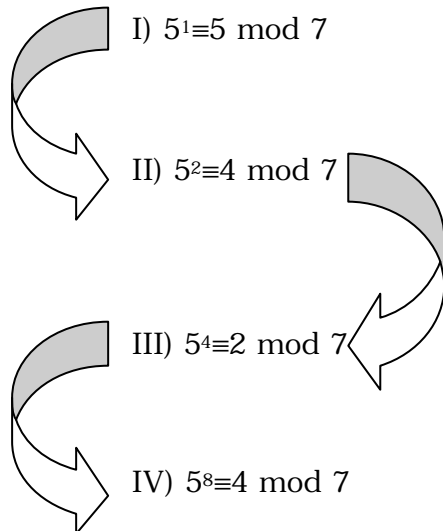
2.3. Große Exponenten

Wie berechnet man das Ergebnis (Rest) bei großen Exponenten?

Lösung: Faktorisierung

Beispiel: $5^{13} \bmod 7$

$1 \cdot 2 = 2$; Potenz für II
 $5^2 = 25$; $25 \bmod 7 = 4$
 4 ist Rest für II



$2 \cdot 2 = 4$; Potenz für III
 $4^2 = 16$; $16 \bmod 7 = 2$
 2 ist Rest für III

$4 \cdot 2 = 8$; Potenz für IV
 $2^2 = 4$; $4 \bmod 7 = 4$
 4 ist Rest für IV

$$\left. \begin{array}{l} 5^8 \equiv 4 \bmod 7 \\ 5^4 \equiv 2 \bmod 7 \\ 5^1 \equiv 5 \bmod 7 \end{array} \right\} 4 \cdot 2 \cdot 5 = 40 \quad 40 \equiv 5 \bmod 7$$

$$\implies 5^8 \cdot 5^4 \cdot 5^1 = 5^{13} \implies 5^{13} \equiv 5 \bmod 7$$

2.4. Primzahlen

Wie später noch zu sehen ist, sind Primzahlen in der Kryptographie sehr bedeutend.

Die Definition einer Primzahl:

Eine natürliche Zahl $p > 1$ heißt Primzahl, wenn sie genau zwei natürliche, positive Teiler hat, nämlich 1 und p . Beispiele: 2, 3, 5, 7, 11, 13,....

Mathematisch sieht es so aus:

$$n \in \text{PRIM} \Leftrightarrow (\forall d < \sqrt{n} : d \in \text{PRIM} \rightarrow d \nmid n)$$

Dies bedeutet: n ist eine Primzahl, wenn alle Primzahlen d , die kleiner sind als \sqrt{n} keine Teiler von n sind.

2.5. Zufallzahlen

Für die Kryptographie benötigen wir in einigen Algorithmen Zufallzahlen.

Was ist eine Zufallszahl?

„Als Zufallszahlen werden Zahlen bezeichnet, die nicht als Ergebnis aus einem festgelegten Zusammenhang (z.B. einer Funktion), sondern nach zufälligen Gesichtspunkten gebildet wurden.“

Eine Münze hat normalerweise 2 Seiten. Niemand kann bei einer (Laplace-) Münze voraussagen, ob der nächste Wurf „Kopf“ oder „Zahl“ bringen wird.

Stellen wir uns vor, dass ein Münzwurf mit „Kopf“ eine Tür öffnet. Dumm dabei ist, dass es nur 2 Möglichkeiten (Kopf oder Zahl) gibt, die die Tür entriegeln. Somit könnte ein Eindringling einfach beide Möglichkeiten ausprobieren (siehe dazu auch Abschnitt 2.1 „Sichere Passworte“) und hätte den „Schutz“ geknackt.

So ziemlich jede Programmiersprache stellt eine Random-Funktion zur Verfügung, die „zufällige“ Zahlen auswirft. Diese Random-Funktionen werden alle mit einer Initialisierung gestartet:

```
zufall = getrandom(init);
```

Die Funktion „getrandom“ wird immer dieselbe Zahl zurückliefern, wenn „init“ denselben Wert hat. Nutzt man bspw. die aktuelle Zeit als „init“, könnte der Angreifer anhand des Erstellungsdatums einer Datei oder der Zeit eines eMail-Versands „init“ wieder konstruieren, da er lediglich

```
zufall = getrandom(init);
```

aufrufen muss. Mit großer Wahrscheinlichkeit kommt er nicht direkt auf den exakten „init“-Wert, aber es wird vielleicht „nur“ 10.000.000 mögliche Werte geben und diese können schnell ausprobiert werden.

Die mit einer Random-Funktion (=Pseudozufallzahlen-Generator) erzeugten Zufallzahlen werden als „Pseudozufallzahlen“ bezeichnet, da es sich nicht wirklich um Zufallzahlen handelt, die hier ermittelt werden.

Um kryptographisch sichere Zufallzahlen zu generieren, verwenden die entsprechenden Programme unterschiedliche Verfahren:

- Zählen von Tastaturanschlägen
- Mauspositionen
- den HASH einer Datei X auf der Festplatte
- Regentropfen auf einer Fläche
- radioaktiver Zerfall
-

2.6. Symmetrische und asymmetrische Verschlüsselung

Man unterscheidet grundsätzlich zwischen 2 Verfahren:

- symmetrische Verschlüsselung: eine Nachricht benötigt **einen** Schlüssel zur Kodierung und Dekodierung; ein Vorteil ist die Ablaufgeschwindigkeit
- asymmetrische Verschlüsselung: eine Nachricht benötigt **zwei** Schlüssel; Schlüssel 2 kann das dekodieren, was Schlüssel 1 kodiert hat und umgekehrt; ein Vorteil ist die hohe Sicherheit

3. Primzahlen

3.1. Auffinden von Primzahlen

Für die folgenden Algorithmen (siehe RSA, Diffie-Hellman,...) spielen große Primzahlen eine entscheidende Rolle.

Hierbei geht es allerdings nicht um Primzahlen, die jeder kennt und aufzählen kann (3, 5, 7, 11, 13, 17,...), sondern um „gigantisch“ große Zahlen mit 600 Bit, also rund 180 Stellen und mehr.

3.2. Unterschiedliche Verfahren

Das Auffinden von Primzahlen unterscheidet zwei generelle Ansätze:

- exakte Verfahren können beweisen, dass eine Zahl prim ist („Las-Vegas-Methode“)
- probabilistische Verfahren können nur mit einer bestimmten Wahrscheinlichkeit aussagen, ob eine Zahl prim ist („Monte-Carlo-Methode“)

3.3. Ein einfacher Ansatz

Man prüft für alle Zahlen von 2 bis $p-1$, ob sie p teilen.

Bei kleineren Testkandidaten wie 13, 17, 51 oder 101 geht das relativ schnell, doch wie eingangs bereits beschrieben, arbeiten Verschlüsselungsverfahren mit großen Zahlen.

Somit gestaltet sich die Probeteilung von bspw. 4398764098762980137695837 schon als wesentlich schwieriger. Ganz zu schweigen von Zahlen mit 600 Bit oder mehr.

3.4. Die Probedivision

Im Abschnitt „Grundlagen“ wurde folgende Definition geliefert:

$$n \in \text{PRIM} \Leftrightarrow (\forall d < \sqrt{n} : d \in \text{PRIM} \rightarrow d \nmid n)$$

Hieraus ergeben sich zwei Regeln:

1. Regel:

Jede natürliche Zahl $a > 1$ hat (mindestens) einen Primteiler, also einen Teiler, der eine Primzahl ist.

2. Regel:

Gehen wir davon aus, dass die zu prüfende Zahl n keine Primzahl ist.

Dann muss gelten: $n = a * b$, mit $a, b > 1$

Entweder ist $a \leq \sqrt{x}$ oder $b \leq \sqrt{x}$, denn sonst würde gelten: $(n=a*b) > (n=\sqrt{x}*\sqrt{x})$ und das kann nicht sein.

Aus diesen beiden Regeln folgt, dass es ausreicht, wenn man für eine zu testende Zahl n alle Primzahlen zwischen 2 und \sqrt{x} prüft, ob sie n teilen.

Beispiel:

$$n=4001 \quad \sqrt{n}=\sqrt{4001}=63 \quad (\text{als Ganzzahl})$$

Alle Primzahlen zwischen 2...63

-> 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59 und 61

Keine dieser Primzahlen teilt 4001 => 4001 ist eine Primzahl

Diese Probedivision ist zwar wesentlich effektiver als der „einfache Ansatz“ (siehe oben), aber nicht wirklich gut, wenn wir die Zahl n wesentlich größer machen. Die Anzahl der zu prüfenden Zahlen $x < \sqrt{n}$ wird mit größer werdendem n zu groß: $\text{AnzahlPrimteiler}(n) = \sqrt{n} / (\log(\sqrt{n}))$

3.5. Satz des Fermat

Der „Satz des Fermat“ ist ein probabilistischer Ansatz, kann also nur mit einer Wahrscheinlichkeit sagen, ob die zu testende Zahl n eine Primzahl ist oder nicht.

Genau genommen beweist der Fermat-Satz auch nur, ob die zu testende Zahl n eine zusammengesetzte Zahl (also keine Primzahl) ist. Der Satz besagt:

Wenn n eine Primzahl ist, dann gilt für alle $a \in \mathbb{N}$, die nicht durch n teilbar sind:

$$a^{n-1} \bmod n = 1$$

Man bestimmt eine Zahl a , die nicht durch n teilbar ist (im einfachsten Fall die 2) und berechnet das Ergebnis.

Ist das Ergebnis ungleich 1, dann ist n mit Sicherheit keine Primzahl. Ist das Ergebnis 1, ist n wahrscheinlich eine Primzahl. Je genauer man mit dieser Wahrscheinlichkeit sein will, desto mehr a müssen gewählt werden.

Beispiel 1:

n=10

$2^9 \bmod 10 = 2$ => Prüfung erledigt, da das Ergebnis $\neq 1$ ist.
=> n=10 ist keine Primzahl

Beispiel 2:

n=541

$2^{540} \bmod 541 = 1$ => Prüfung fortsetzen, wähle das nächste a
 $3^{540} \bmod 541 = 1$ => Prüfung fortsetzen, wähle das nächste a
 $4^{540} \bmod 541 = 1$ => Prüfung fortsetzen, wähle das nächste a
 $5^{540} \bmod 541 = 1$ => Prüfung fortsetzen, wähle das nächste a
 $6^{540} \bmod 541 = 1$ => Prüfung fortsetzen, wähle das nächste a
 $7^{540} \bmod 541 = 1$ => Prüfung fortsetzen, wähle das nächste a
 $8^{540} \bmod 541 = 1$ => Prüfung fortsetzen, wähle das nächste a
...
 $539^{540} \bmod 541 = 1$ => Prüfung fortsetzen, wähle das nächste a
 $540^{540} \bmod 541 = 1$ => ein weiteres a nicht möglich, da $a=(n-1)=540$

==> Laut dem „kleinen Satz des Fermat“ ist n=541 eine Primzahl.

Beispiel 3:

n=561

$2^{560} \bmod 561 = 1$ => Prüfung fortsetzen, wähle das nächste a
 $3^{560} \bmod 561 = 1$ => Prüfung fortsetzen, wähle das nächste a
 $4^{560} \bmod 561 = 1$ => Prüfung fortsetzen, wähle das nächste a
...
 $559^{560} \bmod 561 = 1$ => Prüfung fortsetzen, wähle das nächste a
 $560^{560} \bmod 561 = 1$ => ein weiteres a nicht möglich, da $a=(n-1)=560$

==> Laut dem „kleinen Satz des Fermat“ ist n=561 eine Primzahl.
Problem dabei: 561 ist keine Primzahl, da gilt $561=3*11*17$

Eine Zahl, die den Fermat-Test besteht, aber gar keine Primzahl ist, wird als „Carmichael-Zahl“ bezeichnet. Beispiele: 561, 1105, 1729, 2465, 2821, 6601, 8911, 10585, 15841, 29341, 41041, 46657, 52633, 62745, 63973, 75361,....

Eine Carmichael-Zahl besteht aus mindestens 3 Primfaktoren.

3.6. Der Rabin-Miller-Test

Wie im vorherigen Abschnitt deutlich wurde, ist auch der „Satz des Fermat“ nicht 100%ig zuverlässig.

Eine erneute Verbesserung ist der „Rabin-Miller-Test“. Er besagt:

Sei n eine Primzahl und $n-1=2^t \cdot r$ mit ungeradem r , dann gilt

entweder Gleichung 1 ($a^r \equiv 1 \pmod{n}$) bzw. ($a^r \equiv n-1 \pmod{n}$)

oder Gleichung 2 $\left(a^{2^i \cdot r} \equiv n-1 \pmod{n} \right)$

mit einem i , das eine der beiden Gleichungen erfüllt; mit i aus $[1..t-1]$ und $a < n-1$

Gilt mindestens eine der beiden Bedingungen, sagt der Test aus, dass n eine Primzahl ist. Pro getestetem a liegt man mit einer Wahrscheinlichkeit von $\frac{1}{4}$ daneben. Je mehr Tests, desto „richtiger“ das Ergebnis, also kleiner die Wahrscheinlichkeit für einen Fehler.

Gelten beide Bedingungen nicht, ist n definitiv keine Primzahl.

Beispiel 1:

$$n=541 \qquad 541-1=2^2 \cdot 135 \qquad 540=540 \Rightarrow t=2; r=135$$

Test mit Gleichung 1 und (zufällig gewähltem) $a=3$

$$a^r = 1 \pmod{n}$$

$$3^{135} = 1 \pmod{541}$$

$$3^{135} = 1 \pmod{541}$$

=> Der Test mit Gleichung 2 ist überflüssig, da die Gleichung 1 erfüllt ist. Somit ist $n=541$ zu $\frac{1}{4}$ keine Primzahl.

Test mit Gleichung 1 und (zufällig gewähltem) $a=6$

$$a^r = 1 \pmod{n}$$

$$6^{135} = 1 \pmod{541}$$

$$6^{135} = 52 \pmod{541}$$

=> Der Test mit Gleichung 1 war nicht ausreichend; deshalb wird Gleichung 2 getestet.

$a=6, n=541, i=1, r=135$

$$\left(6^{2^1 \cdot 135} \stackrel{?}{\equiv} 540 \pmod{541} \right) \rightarrow \left(6^{270} \stackrel{?}{\equiv} 540 \pmod{541} \right) \rightarrow \left(6^{270} \equiv 540 \pmod{541} \right)$$

Der Test mit Gleichung 2 war positiv. Somit verringert sich die Wahrscheinlichkeit dafür, dass n keine Primzahl ist, auf $\frac{1}{4} * \frac{1}{4}$

Test mit Gleichung 1 und (zufällig gewähltem) $a=287$

$$a^r \stackrel{?}{\equiv} n-1 \pmod{n} \rightarrow \left(287^{135} \stackrel{?}{\equiv} 540 \pmod{541} \right) \rightarrow \left(287^{135} \equiv 540 \pmod{541} \right)$$

Der Test mit Gleichung 1 war positiv. Somit verringert sich die Wahrscheinlichkeit dafür, dass n keine Primzahl ist, auf $\frac{1}{4} * \frac{1}{4} * \frac{1}{4}$

Dies soll genügen: $n=541$ ist eine Primzahl. (Dies hat ja auch schon der Satz des Fermat bewiesen.)

Beispiel 2:

$$n=561 \qquad 561-1=2^4 \cdot 35 \qquad 560=560 \Rightarrow t=4; r=35$$

Test mit Gleichung 1 und (zufällig gewähltem) $a=3$

$$a^r \stackrel{?}{\equiv} 1 \pmod{n} \qquad 3^{35} \stackrel{?}{\equiv} 1 \pmod{561} \qquad 3^{35} = 78 \pmod{561}$$

Der Test mit Gleichung 1 war nicht ausreichend; deshalb wird Gleichung 2 getestet.

$a=3, n=561, r=35 \quad i = \{1, 2, 3\}$

Test mit $i=1$

$$\left(3^{2^1 \cdot 35} \stackrel{?}{\equiv} 560 \pmod{561} \right) \rightarrow \left(3^{70} \stackrel{?}{\equiv} 560 \pmod{561} \right) \rightarrow \left(3^{70} \equiv 474 \pmod{561} \right)$$

Test mit $i=2$

$$\left(3^{2^2 \cdot 35} \stackrel{?}{\equiv} 560 \pmod{561} \right) \rightarrow \left(3^{140} \stackrel{?}{\equiv} 560 \pmod{561} \right) \rightarrow \left(3^{140} \equiv 276 \pmod{561} \right)$$

Test mit $i=3$

$$\left(3^{2^3 \cdot 35} \stackrel{?}{\equiv} 560 \pmod{561} \right) \rightarrow \left(3^{280} \stackrel{?}{\equiv} 560 \pmod{561} \right) \rightarrow \left(3^{280} \equiv 441 \pmod{561} \right)$$

Sowohl Gleichung 1 als auch Gleichung 2 (mit allen möglichen i) ist negativ ausgefallen. Somit ist $n=561$ definitiv keine Primzahl.

3.7. Anmerkung

Es existieren noch zahlreiche gute und schlechte Ansätze, um Primzahlen zu berechnen. Die Vorstellung aller Berechnungsmöglichkeiten würde aber den Rahmen dieser Ausarbeitung sprengen. Sie wurden auch nicht in der Vorlesung „Protokolle höherer Schichten“ behandelt.

4. Symmetrische Verschlüsselung

4.1. Einleitung

Wie eingangs in den Grundlagen erklärt, zeichnet sich die symmetrische Verschlüsselung dadurch aus, dass es zur Kodierung und zur Dekodierung nur **einen** Schlüssel gibt.

Der Nachteil des Systems ist der direkte Austausch des geheimen Schlüssels. Meist sind Sender und Empfänger nicht persönlich bekannt, so dass die persönliche Absprache sich ggf. als äußerst schwierig gestaltet.

4.2. SSL

Ein in den Zeiten des Internets bekanntes Ein-Schlüssel-Verfahren ist SSL (Secure Socket Layer, also `https://...`), das etwa 1994 von der Firma Netscape entwickelt wurde. Der Ablauf ist ziemlich einfach:

- der User schickt seine Anfrage an den WWW-Server
- der WWW-Server zeigt an, dass es sich um einen sicheren Bereich der Webseite handelt und schickt seinen öffentlichen Schlüssel an den User zurück
- der Browser des Users generiert eine Zufallszahl (den symmetrischen Schlüssel für die SSL-Verbindung), verschlüsselt diesen mit dem öffentlichen Schlüssel des WWW-Servers und schickt die Nachricht zurück
- der WWW-Server dekodiert die Nachricht mit seinem geheimen Schlüssel und schickt die Daten der Webseite von nun an kodiert mit dem symmetrischen Schlüssel an den User

Anfangs wurde mit Schlüssellängen von 40 Bit und 56 Bit gearbeitet¹, doch diese reichen heutzutage nicht mehr aus, so dass nun standardmäßig alle SSL-Verbindungen mit 128Bit-Schlüsseln kodiert werden.

Durch die Mischung von symmetrischer und asymmetrischer Verschlüsselung kann der beschriebene Nachteil (geheimer Schlüsselaustausch) der symmetrischen Verschlüsselung umgangen werden.

Diese Kombination aus beiden Verfahren wird auch als „hybride Verschlüsselung“ bezeichnet. (siehe auch Abschnitt 6 „Hybride Verschlüsselung“)

4.3. DES und AES

Ein bekannter Algorithmus zur symmetrischen Verschlüsselung ist der DES (**D**ata **E**ncryption **S**tandard), der vom US-Verteidigungsministerium entwickelt wurde.

¹ aus Export-Gründen; Auflage der USA, da „starke Kryptographie“ als eine Waffe angesehen wurde

Da der DES nur mit einer Schlüssellänge von 56 Bit arbeitet, gilt er heute nicht mehr als sicher. (siehe dazu auch „2.1 Sichere Passworte“)

Anstelle dessen wird seit 1997/1998 der AES (**A**dvanced **E**ncryption **S**tandard) eingesetzt, der mit 128 Bit, 192 Bit und sogar 256 Bit arbeitet. Zudem gibt es noch eine Abwandlung des DES: den TripleDES oder 3DES, der mit einem 112 Bit bzw. 168 Bit Schlüssel betrieben wird. Hierbei werden drei DES-Stufen hintereinander geschaltet.

5. Asymmetrische Verschlüsselung

5.1. Einleitung

Die asymmetrische Verschlüsselung zeichnet sich nicht nur durch die Sicherheit aus, sondern zudem durch die unproblematische Schlüsselverteilung.

Schwierig dagegen sind die sehr großen Primzahlen, die als Basis der Schlüssel dienen. Außerdem ist die asymmetrische Verschlüsselung wesentlich langsamer als die symmetrische Verschlüsselung¹.

Das wohl bekannteste Verfahren ist der RSA-Algorithmus, der Ende der 70er Jahre von **R**ivest, **S**hamir und **A**dleman entwickelt wurde. Hierbei wird ein Schlüsselpaar erzeugt, das auf der Basis von Primzahlen entsteht. Ein Schlüssel ist frei verfügbar und wird als „Public Key“ bezeichnet. Der andere Schlüssel wird vom Erzeuger geheim gehalten und „Private Key“ genannt.

5.2. Der RSA-Algorithmus

5.2.1. Die Sicherheit des RSA

Im Internet² wurde folgende Beschreibung gefunden, die genau auf den Punkt bringt, was die Sicherheit des RSA ausmacht:

Die Sicherheit des RSA-Algorithmus beruht auf der Schwierigkeit, eine sehr große Zahl n zu faktorisieren - das Wissen um diese Faktorisierung ist die Hintertür-Information.

Man kann vergleichsweise schnell überprüfen, ob eine Zahl n prim ist oder nicht. Dagegen ist es ein sehr schwieriges Problem, von einer Zahl n , von der man weiß, dass sie nicht prim ist, die Primfaktorenzerlegung zu bestimmen. Der einfachste Weg ist, jeden möglichen Teiler durchzuprobieren, was aber dazu führen kann, dass man unter Umständen den gesamten Bereich von 1 bis \sqrt{n} in Betracht ziehen muss.

Die in der Praxis verwandten Zahlen sind aber so groß, dass dies nicht möglich ist.

¹ Beispiel: Der RSA-Algorithmus ist etwa 1000 Mal langsamer als DES

² http://www.wiwi.uni-bielefeld.de/StatCompSci/lehre/material_spezifisch/statalg00/rsa/node9.html

5.2.2. Ein ausführliches Beispiel

- * wähle zwei sehr große Primzahlen p und q (z.B. durch Rabin-Miller-Test)
Beispiel: $p=541$ $q=613$ ¹
- * errechne n mit $n = p * q$
Beispiel: $n = 541 * 613 = 331633$
- * errechne $\phi(n) = (p-1) * (q-1)$
hier: $\phi(n) = (541-1) * (613-1) = 540 * 620 = 330480$
Bedeutung von $\phi(n)$: Für $n=331633$ existieren genau 330480 Zahlen zwischen 1 und n, die relativ prim zu n sind. Das heißt: Zahlen sind, dessen $\text{ggT}=1$ ist. (Sie haben also keine gemeinsamen Faktoren, außer die 1)

Die übrigen 1153 Zahlen sind nicht prim zu $n=331633$, der ggT ist also $\neq 1$:

$\text{ggT}(331633, 541) = 541$ ($331633:541=613$ und $541:541=1$)
 $\text{ggT}(331633, 613) = 613$ ($331633:613=541$ und $613:613=1$)
 $\text{ggT}(331633, 1082) = 541$ ($331633:541=613$ und $1082:541=2$)
 $\text{ggT}(331633, 1226) = 613$
 $\text{ggT}(331633, 1623) = 541$
 $\text{ggT}(331633, 1839) = 613$
 $\text{ggT}(331633, 2164) = 541$
 $\text{ggT}(331633, 2452) = 613$
 $\text{ggT}(331633, 2705) = 541$
 $\text{ggT}(331633, 3065) = 613$
 $\text{ggT}(331633, 3246) = 541$
 $\text{ggT}(331633, 3678) = 613$
 $\text{ggT}(331633, 3787) = 541$
 $\text{ggT}(331633, 4291) = 613$
 $\text{ggT}(331633, 4328) = 541$
 $\text{ggT}(331633, 4869) = 541$
 $\text{ggT}(331633, 4904) = 613$
...

Es ergeben sich somit 2 Mengen:

$M_1 = \{1, 2, 3, 4, 5, 6, 7, \dots, 331632\} \rightarrow 330480$ Zahlen teilerfremd zu $n=331633$
 $M_2 = \{541, 613, 1082, 1226, \dots, 331633\} \rightarrow 1153$ Zahlen nicht-teilerfremd zu n

¹ diese Primzahlen sind natürlich für den „echten“ Einsatz viel zu klein, jedoch für eine Beispielrechnung angemessen

Der Satz von Euler besagt, dass $a^{\Phi(n)} \bmod n = 1$

oder umgeschrieben $a^{\Phi(n)} \equiv 1 \pmod{n}$

Aus diesem Beispiel heraus gilt: $\phi(n)=120$, $n=143$ und a gilt für die oben berechneten teilerfremden Zahlen M_1

Probe:

$$2^{330480} \equiv 1 \pmod{331633} \quad \text{da gilt } 2^{330480} : 331633 = X_1 \text{ Rest } 1$$

$$3^{330480} \equiv 1 \pmod{331633} \quad \text{da gilt } 3^{330480} : 331633 = X_2 \text{ Rest } 1$$

...

X_1 , X_2 usw. sind sehr groß und an sich auch unwichtig. Wichtig ist nur, dass bei den Berechnungen mit M_1 immer ein Rest von 1 herauskommt.

Bei den anderen Zahlen $M_2=\{\dots\}$ als Basis a wird sich ein anderer Rest ergeben.

Jetzt wird das e (also der öffentliche encrypting-Schlüssel) ausgewählt. Einzige Bedingung ist, dass e teilerfremd zu $\phi(n)$ sein muss. Demnach ergibt sich aus diesem Beispiel:

Mögliche Werte für e : 1, 7, 11, 13, 19, 23, 29, 31, 37, 41, 43, 47, 49, 53, 59, 61, 67, 71, 73, 77, 79, 83, 89, 91, 97, 101, 103, 107, 109, 113, 121, 127, 131, 133, 137,...

Für dieses Beispiel wird $e = 137$ gewählt.

Nun erfolgt die Berechnung von d , das den geheimen decryption-Schlüssel repräsentiert. Es muss gelten: die Zahl d ist die multiplikative Inverse zu e :

$$ed \bmod \Phi(n) = 1$$

Die Berechnung mittels euklidischem Algorithmus.

$$\vec{W}_0 = \begin{pmatrix} a \\ 1 \\ 0 \end{pmatrix} \quad \vec{W}_1 = \begin{pmatrix} b \\ 0 \\ 1 \end{pmatrix} \quad q_k = \left[\frac{W_{k-1,1}}{W_{k,1}} \right] \quad \vec{W}_{k+1} = \vec{W}_{k-1} - q_k \cdot \vec{W}_k$$

Aus diesem Beispielen heraus gilt $a = \phi(n) = 330480$ und $b = \underline{e} = 137$
(Voraussetzung: $a > b$)

$$\vec{W}_0 = \begin{pmatrix} 330480 \\ 1 \\ 0 \end{pmatrix} \quad \vec{W}_1 = \begin{pmatrix} 137 \\ 0 \\ 1 \end{pmatrix}$$

$$q_1 = \left[\frac{W_{0,1}}{W_{1,1}} \right] = \left[\frac{330480}{137} \right] \quad \text{Anmerkung: } \left[\frac{330480}{137} \right] \text{ ist als Ganzzahl-} \\ \text{Division zu betrachten. Hier ist die Lösung} \\ \text{demnach } \mathbf{2412}, \text{ da } 330480 : 137 = \mathbf{2412} \text{ Rest } 36$$

$$\vec{W}_2 = \begin{pmatrix} 330480 \\ 1 \\ 0 \end{pmatrix} - \left(\frac{330480}{137} \right) \cdot \begin{pmatrix} 137 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 330480 \\ 1 \\ 0 \end{pmatrix} - 2412 \cdot \begin{pmatrix} 137 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 330480 \\ 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 330444 \\ 0 \\ 2412 \end{pmatrix} = \begin{pmatrix} 36 \\ 1 \\ -2412 \end{pmatrix}$$

$$q_2 = \left[\frac{W_{1,1}}{W_{2,1}} \right] = \left[\frac{137}{36} \right] = 3$$

$$\vec{W}_3 = \begin{pmatrix} 137 \\ 0 \\ 1 \end{pmatrix} - \left(\frac{137}{36} \right) \cdot \begin{pmatrix} 36 \\ 1 \\ -2412 \end{pmatrix} = \begin{pmatrix} 137 \\ 0 \\ 1 \end{pmatrix} - 3 \cdot \begin{pmatrix} 36 \\ 1 \\ -2412 \end{pmatrix} = \begin{pmatrix} 137 \\ 0 \\ 1 \end{pmatrix} - \begin{pmatrix} 108 \\ 3 \\ -7236 \end{pmatrix} = \begin{pmatrix} 29 \\ -3 \\ 7237 \end{pmatrix}$$

$$q_3 = \left[\frac{W_{2,1}}{W_{3,1}} \right] = \left[\frac{36}{29} \right] = 1$$

$$\vec{W}_4 = \begin{pmatrix} 36 \\ 1 \\ -2412 \end{pmatrix} - \left(\frac{36}{29} \right) \cdot \begin{pmatrix} 29 \\ -3 \\ 7237 \end{pmatrix} = \begin{pmatrix} 36 \\ 1 \\ -2412 \end{pmatrix} - 1 \cdot \begin{pmatrix} 29 \\ -3 \\ 7237 \end{pmatrix} = \begin{pmatrix} 36 \\ 1 \\ -2412 \end{pmatrix} - \begin{pmatrix} 29 \\ -3 \\ 7237 \end{pmatrix} = \begin{pmatrix} 7 \\ 4 \\ -9649 \end{pmatrix}$$

$$q_4 = \left[\frac{W_{3,1}}{W_{4,1}} \right] = \left[\frac{29}{7} \right] = 4$$

$$\vec{W}_5 = \begin{pmatrix} 29 \\ -3 \\ 7237 \end{pmatrix} - \left(\frac{29}{7} \right) \cdot \begin{pmatrix} 7 \\ 4 \\ -9649 \end{pmatrix} = \begin{pmatrix} 29 \\ -3 \\ 7237 \end{pmatrix} - 4 \cdot \begin{pmatrix} 7 \\ 4 \\ -9649 \end{pmatrix} = \begin{pmatrix} 29 \\ -3 \\ 7237 \end{pmatrix} - \begin{pmatrix} 28 \\ 16 \\ -38596 \end{pmatrix} = \begin{pmatrix} 1 \\ -19 \\ 45833 \end{pmatrix}$$

$$q_5 = \left[\frac{W_{4,1}}{W_{5,1}} \right] = \left[\frac{7}{1} \right] = 7 \quad \text{ganzzahliges Ergebnis ohne Rest } \Rightarrow \text{ fertig}$$

$\text{ggT}(330480, 137) = W_{5,1} = 1$ (war vorher bereits bekannt)

Des weiteren ergeben sich $W_{5,2} = u_1 = -19$ und $W_{5,3} = v_1 = 45833$.

Um den Schlüssel **d** zu bekommen, wird folgende Formel angewandt:

$$e^{-1} = d = x \text{ mod } \Phi(n)$$

Dies bedeutet, dass das x (hier: u_1 und v_1) „moduliert“ mit $\Phi(n)$ werden muss und sich daraus ein (vermeintliches) **d** ergibt:

$$\begin{aligned} u_2 &= u_1 \text{ mod } \Phi(n) = -19 \text{ mod } 330480 = 330461 \\ v_2 &= v_1 \text{ mod } \Phi(n) = 45833 \text{ mod } 330480 = 45833 \end{aligned}$$

Welche der beiden Zahlen u_2 und v_2 der geheime Schlüssel **d** ist, wird über eine Vorschrift ermittelt. Es muss gelten:

$$(\mathbf{d} * \mathbf{e}) \text{ mod } \Phi(n) = 1$$

Prüfung:

- * **e** wurde vorher von uns mit $\mathbf{e} = 137$ gewählt
- * $\Phi(n)$ ist ebenfalls bekannt: 330480
- * **d** ist entweder u_2 oder v_2

für u_2 gilt:

$$\begin{aligned} (\mathbf{d} * \mathbf{e}) \text{ mod } \Phi(n) &= 1 \\ \text{mit } \mathbf{d} &= u_2 \\ (u_2 * \mathbf{e}) \text{ mod } \Phi(n) &= 1 \\ \implies (330461 * 137) \text{ mod } 330480 &= 327877 \neq 1 \end{aligned}$$

für v_2 gilt:

$$\begin{aligned} (\mathbf{d} * \mathbf{e}) \text{ mod } \Phi(n) &= 1 \\ \text{mit } \mathbf{d} &= v_2 \\ (v_2 * \mathbf{e}) \text{ mod } \Phi(n) &= 1 \\ \implies (45833 * 137) \text{ mod } 330480 &= 1 \end{aligned}$$

Somit ist der geheime Schlüssel **d**=45833 und alle erforderlichen Daten zur Verschlüsselung und Entschlüsselung sind bekannt:

$$\begin{aligned} n &= 331633 \\ \mathbf{d} &= 45833 \\ \mathbf{e} &= 137 \end{aligned}$$

Die Werte von p, q und $\Phi(n)$ werden vernichtet, da ein Angreifer mit diesen Werten den RSA-verschlüsselten Text „knacken“ könnte. (Er kann ja mittels dem oben gezeigten Algo. selber den geheimen Schlüssel erzeugen, wenn er $\Phi(n)$ oder p oder q kennt!

Nun kann ein Text mittels RSA und den o.g. Schlüssel kodiert werden.

Das Wort „geheim“ soll verschlüsselt werden. Als Grundlage braucht der RSA-Algorithmus Zahlen, so dass „geheim“ zuerst in Zahlen umwandelt werden muss. Hierfür wird die ASCII-Tabelle genutzt, so dass sich ergibt:

DEC(g) = 103
DEC(e) = 101
DEC(h) = 104
DEC(e) = 101
DEC(i) = 105
DEC(m) = 109

Allgemein gilt: $B^e \bmod n = C$

- * B ist ein ASCII-Zahlenwert, also 103, 101, 104,...
- * e ist der öffentliche Schlüssel **e**
- * n ist in unserem Beispiel 331633
- * C ist der kodierte ASCII-Zahlenwert

g-> $103^{137} \bmod 331633 = 309241$
e-> $101^{137} \bmod 331633 = 325604$
h-> $104^{137} \bmod 331633 = 261840$
e-> $101^{137} \bmod 331633 = 325604$
i-> $105^{137} \bmod 331633 = 158849$
m-> $109^{137} \bmod 331633 = 321375$

Die Nachricht „geheim“ ist also kodiert:

309241 325604 261840 325604 158849 321375

Lediglich mit dem geheimen Schlüssel **d** kann aus den obigen Zahlen die Nachricht zurückkodiert werden.

Allgemein gilt: $C^d \bmod n = B$

- * C ist der jeweilige kodierte ASCII-Zahlenwert (309241, 325604,...)
- * d ist der geheime Schlüssel **d**
- * n ist in unserem Beispiel 331633
- * B ist der entschlüsselte ASCII-Zahlenwert

$309241^{45833} \bmod 331633 = 103$	Probe: ASC(103) = g
$325604^{45833} \bmod 331633 = 101$	Probe: ASC(101) = e
$261840^{45833} \bmod 331633 = 104$	Probe: ASC(104) = h
$325604^{45833} \bmod 331633 = 101$	Probe: ASC(101) = e
$158849^{45833} \bmod 331633 = 105$	Probe: ASC(105) = i
$321375^{45833} \bmod 331633 = 109$	Probe: ASC(109) = m

Anmerkung:

In einigen „privaten“ Dokumentation (und auch in der Vorlesung wurde es gesagt) steht, dass der zu kodierende ASCII-Werte unbedingt in der Menge M_2 (siehe oben), der Wert also teilerfremd zu n sein muss. Dies bestätigen viele offizielle Dokumente zum RSA-Algorithmus nicht und auch diverse Kodier- und Dekodier-Tests können dies nicht bestätigen.

Es muss lediglich gelten: $x < n$

5.2.3. Ein (nicht ganz so) ausführliches Beispiel

Dieses zweite Beispiel beschreibt lediglich die „Eckdaten“, da die Herleitungen bzw. Berechnungen für p , q , n , $\phi(n)$, \mathbf{e} , u_1 , v_1 , u_2 , v_2 und \mathbf{d} bereits ausführlich erklärt wurden.

Zur schnelleren Berechnung kam MAPLE in der Version 8 zum Einsatz, da der erweiterte Euklid bei so großen Zahlen eine Menge Schreiarbeit mit sich bringt.

$p = 2239$ -> Primzahl, frei gewählt
 $q = 3119$ -> Primzahl, frei gewählt

$n = p * q = 2239 * 3119 = 6983441$
 $\phi(n) = 6978084$

\mathbf{e} wählen:

$\mathbf{e} = 245873$ -> teilerfremd zu $\phi(n)$; also $\text{ggT}(6978084, 245873) = 1$

mittels euklidischem Algorithmus erfolgt die Berechnung von u_1 und v_1 :

$u_1 = 2809193$
 $v_1 = -98982$

Berechnung von u_2 und v_2 :

$u_2 = u_1 \bmod \phi(n) = 2809193 \bmod 6978084 = 2809193$
 $v_2 = v_1 \bmod \phi(n) = -98982 \bmod 6978084 = 6879102$

Ermittlung von \mathbf{d} :

für u_2 gilt:

$(\mathbf{d} * \mathbf{e}) \bmod \phi(n) = 1$
 $\implies (2809193 * 245873) \bmod 6978084 = 1$

für v_2 gilt:

$(\mathbf{d} * \mathbf{e}) \bmod \phi(n) = 1$
 $\implies (6879102 * 245873) \bmod 6978084 = 2555706 \neq 1$

Ergebnis:

$n = 6983441$
 $\mathbf{d} = 2809193$
 $\mathbf{e} = 245873$

Als Text wird mit diesen Parametern nun „Protokolle“ kodiert und dekodiert, um die Verschlüsselung erneut zu verdeutlichen.

Kodierung der Nachricht „Protokolle“:

P-> $80^{245873} \bmod 6983441 = 239444$
r-> $114^{245873} \bmod 6983441 = 4676657$
o-> $111^{245873} \bmod 6983441 = 4349430$
t-> $116^{245873} \bmod 6983441 = 1230254$
o-> $111^{245873} \bmod 6983441 = 4349430$
k-> $107^{245873} \bmod 6983441 = 5956437$
o-> $111^{245873} \bmod 6983441 = 4349430$
l-> $108^{245873} \bmod 6983441 = 1248491$
l-> $108^{245873} \bmod 6983441 = 1248491$
e-> $101^{245873} \bmod 6983441 = 571814$

Dekodierung der Nachricht

**239444 4676657 4349430 1230254 4349430
5956437 4349430 1248491 1248491 571814**

$239444^{2809193} \bmod 6983441 = 80$	Probe:ASC(80) = P
$4676657^{2809193} \bmod 6983441 = 114$	Probe:ASC(114) = r
$4349430^{2809193} \bmod 6983441 = 111$	Probe:ASC(111) = o
$1230254^{2809193} \bmod 6983441 = 116$	Probe:ASC(116) = t
$4349430^{2809193} \bmod 6983441 = 111$	Probe:ASC(111) = o
$5956437^{2809193} \bmod 6983441 = 107$	Probe:ASC(107) = k
$4349430^{2809193} \bmod 6983441 = 111$	Probe:ASC(111) = o
$1248491^{2809193} \bmod 6983441 = 108$	Probe:ASC(108) = l
$1248491^{2809193} \bmod 6983441 = 108$	Probe:ASC(108) = l
$571814^{2809193} \bmod 6983441 = 101$	Probe:ASC(101) = e

Protokolle

ist kodiert

**239444 4676657 4349430 1230254 4349430
5956437 4349430 1248491 1248491 571814**

5.2.4. Abschließende Anmerkungen

Durch diese zwei Beispiele sollte nun deutlich geworden sein, wie der RSA funktioniert.

Und was macht ihn so sicher? Das gewählte bzw. berechnete n ist sehr groß (zur Zeit meist 1024 Bit und größer, da p und q jeweils eine Länge von 512 Bit und größer haben) und die Faktorisierung dieser Zahl n ist schwierig bzw. unmöglich.

Herauszufinden, aus welchen Faktoren (hier: p und q) eine Zahl (hier: n) besteht, wird mit „normalen“ Programmen (z.B. MAPLE) ab 50 oder 60 Bit schon zu einer langwierigen Rechenoperation.

Selbst bei speziellen Programmen, die ausschließlich auf die Faktorisierung hin optimiert wurden, sollte bei einem n mit 256 Bit oder sogar 512 Bit die Zeit für die Berechnung nicht mehr im Bereich von Stunden oder Tagen liegen.

Um so sicherer ist die Verschlüsselung mit einem n von 1024 Bit. (PGP bietet mittlerweile ein n von 2048 Bit für den RSA-Schlüssel an.)

6. Hybride Verschlüsselung

6.1. Warum „hybrid“?

Wie in den vorherigen Abschnitten erklärt, haben die symmetrische und asymmetrische Verschlüsselungsverfahren Nachteile:

symmetrisch : Austausch des Schlüssels ist problematisch
asymmetrisch: Verfahren ist langsam

Durch die „hybride Verschlüsselung“, also der Verschmelzung eines symmetrischen Verfahrens mit einem asymmetrischen Verfahren, werden die Vorteile beider Methoden genutzt.

6.2. Ablauf einer hybriden Verschlüsselung

1. Die zu verschlüsselnde Nachricht wird mit einem symmetrischen Schlüssel („Session Key“) verschlüsselt.
2. Mittels einem asymmetrischen Verfahren wird dieser Session Key mit dem Public Key verschlüsselt und an die kodierte Nachricht gehängt.
3. Der Besitzer des Private Key kann den Session Key entschlüsseln und damit wiederum die Nachricht.

7. Schlüsselerzeugungs-Verfahren

7.1. Einleitung

Eine Möglichkeit, den Schlüssel einer symmetrischen Verschlüsselung auszutauschen, wurde bereits beschrieben.

Eine andere Methode ist die gemeinsame Schlüsselerzeugung, bei der es darum geht, dass sich zwei Personen **sicher** auf einen gemeinsamen Schlüssel einigen, ohne sich jemals persönlich begegnen zu müssen.

7.2. Diffie-Hellman: Der Ablauf

Zwei Personen A und B möchten einen gemeinsamen Schlüssel vereinbaren. Mit Hilfe des **Schlüsselerzeugungsverfahrens** („Diffie-Hellman“-Algorithmus) ist dies möglich.

Die beiden Teilnehmer einigen sich auf eine (normalerweise sehr große) Primzahl p , sowie eine Basis g , für die gilt $1 < g < p$ und g ist keine Primzahl². Diese beiden Zahlen werden offen ausgetauscht; es wäre also nicht riskant, wenn eine Person C diese Zahlen bekommt.

Nun folgt ein zweistufiges Verfahren:

1. A und B denken sich jeweils eine geheime Zahl

$$a < p-1 \quad \text{bzw.} \quad b < p-1$$

aus und berechnen die diskrete Exponentialfunktion, wobei „diskret“ bedeutet, dass es sich um ganzzahlige Werte handelt:

$$\begin{aligned} \text{Person A:} \quad & \alpha = g^a \bmod p \\ \text{Person B:} \quad & \beta = g^b \bmod p \end{aligned}$$

Die errechneten Werte für α und β werden jeweils an den anderen Teilnehmer verschickt. (Auch hier gilt: kein Risiko, wenn Person C mithört)

2. Beide berechnen wieder die diskrete Exponentialfunktion und erhalten einen geheimen Schlüssel k :

$$\begin{aligned} \text{Person A:} \quad & k = \beta^a \bmod p \\ \text{Person B:} \quad & k = \alpha^b \bmod p \end{aligned}$$

Das von beiden errechnete k ist identisch und kann als Schlüssel verwendet werden. Zu keinem Zeitpunkt konnte ein Angreifer C den Austausch gefährden, da alle ihm vorliegenden Informationen (für ihn) nutzlos sind.

¹ benannt nach den Erfindern Whitfield Diffie und Martin Hellman

² offiziell heißt es: g muss aus einer „multiplikativen Gruppe“ kommen, bedeutet: darf keine Primzahl sein

7.3. Ein Beispiel

Durch ein kleines Beispiel soll das Verfahren noch einmal verdeutlicht werden.

Es wurden relativ kleine Zahlen genutzt, um nicht den Überblick zu verlieren. Im Normalfall wird mit Primzahlen von ca. 180-200 Dezimalstellen gearbeitet.

- öffentlich sind $p=2239$ und die Basis $g=500$
- Person A wählt $a=2111$:
 $\alpha = g^a \bmod p$
 $797 = 500^{2111} \bmod 2239$
- Person B wählt $b=1327$:
 $\beta = g^b \bmod p$
 $2025 = 500^{1327} \bmod 2239$
- Person A gibt sein α an Person B.
- Person B gibt sein β an Person A.
- Person A berechnet:
 $k = \beta^a \bmod p$
 $2069 = 2025^{2238} \bmod 2239$
- Person B berechnet:
 $k = \alpha^b \bmod p$
 $2069 = 2025^{1327} \bmod 2239$

Für beide ist nun $k=2069$ der geheime Schlüssel.

7.4. Warum ist „ $k=k$ “?

Wieso haben A und B nun denselben Schlüssel k , obwohl doch deren Geheimzahlen a bzw. b der andere nicht kennt?

Laut den Potenzgesetzen gilt:

$$g^{a*b} = (g^a)^b = (g^b)^a = g^{b*a}$$

Demnach gilt auch:

$$g^{a*b} \bmod p = (g^a)^b \bmod p = (g^b)^a \bmod p = g^{b*a} \bmod p$$

Person A hat ja berechnet: $k = \mathbf{b}^a \bmod p$

das β wurde durch Person B aus $\beta = g^b \bmod p$ errechnet

also ist $\mathbf{b}^a = (g^b)^a \bmod p$

Person B wiederum hat berechnet: $k = a^b \bmod p$

das α wurde durch Person A aus $\alpha = g^a \bmod p$ errechnet

also ist $a^b = (g^a)^b \bmod p$

Und da (wie oben beschrieben) gilt $(g^a)^b = (g^b)^a$, muss k identisch sein.

Was kann nun Person C (Angreifer) machen, um a oder b zu ermitteln? Ihm stehen im besten Fall die Werte für α , β , g und p zur Verfügung und er weiß, dass sich α so berechnet:

$$\alpha = g^a \bmod p$$

Also muss er den diskreten Logarithmus von α zur Basis g berechnen:

$$a = \log_g(\alpha) \bmod p$$

Diese Berechnung ist aber nicht möglich, wenn man bedenkt, dass die Werte von p , g , a und b in der Größenordnung von jeweils 600 Bit liegen. Es kommt also nur das Ausprobieren („BruteForce“) in Frage und dies macht bei ca. 10^{100} Ausprobier-/Suchoperationen keinen Sinn.

Die Sicherheit des Verfahrens beruht entscheidend darauf, dass die diskrete Exponentialfunktion eine Einwegfunktion ist.

7.5. Brute Force

Ein BruteForce-Versuch für die obigen Zahlen war erwartungsgemäß innerhalb weniger Sekunden „geknackt“.

Bei $p=8647739$, $g=86477$ und $a=8000000$ dauerte der Angriff bereits mehr als 20 Minuten. (Getestet mit MAPLE)

Dies sind aber Zahlen mit „nur“ 7 Stellen, im echten Einsatz werden Zahlen mit 200 Stellen genutzt. Die Sicherheit dieses Systems sollte deutlich sein.

8. Signaturen

8.1. Einleitung

Die Eigenschaften

- Herkunft
- Unverfälschtheit/Echtheit
- Nicht-Widerufbarkeit

eines Dokuments werden im Alltag mit einer handschriftlichen Unterschrift zugesichert.

Im Bereich von Computern, Internet und PayServices ist dieser Vorgang schwierig, da die beteiligten Personen sich meist nicht kennen und relevante Daten-Bits ggf. unbemerkt verändert werden können.

Um diesem Manko entgegenzuwirken, werden digitale Signaturen eingesetzt, die obige Eigenschaften sogar noch wesentlich eindeutiger und sicherer erfüllen, als es die handschriftliche Unterschrift kann.

8.2. Eine einfache Methode

Person 1 möchte Person 2 einen Text zusenden, nimmt die Nachricht

Dies ist mein Text



und verschlüsselt den Text mit seinem eigenen Private-Key.



3B3§/23n4i;:.34,32

Nun verschickt Person 1 den Klartext und den kodierten Text an Person 2, die wiederum prüfen kann, ob der Text verändert wurde (es liegen beide Text-Versionen vor) und ob der verschickte Text wirklich von Person 1 (da der Public-Key passen muss) stammt.

Leider hat diese Methode einen gravierenden Nachteil, denn wie schon mehrmals in dieser Ausarbeitung geschrieben, sind asymmetrische Verschlüsselungsverfahren (Verfahren mit unterschiedlichen Schlüsseln zum Kodieren und Dekodieren einer Nachricht) sehr langsam.

Handelt es sich bei der Nachricht „Dies ist mein Text“ um eine Nachricht mit einer Größe von mehreren MB, würde die Kodierung sehr lange dauern. Außerdem muss die doppelte Datenmenge (verschlüsselter und nicht-verschlüsselter Text) übermittelt werden.

8.3. Verbesserung: die HASH-Funktion

Um die Nachteile aus Abschnitt 8.2 „Eine einfache Methode“ zu kompensieren, nutzt man sogenannte „HASH“-Funktionen, die öffentlich bekannt sind.

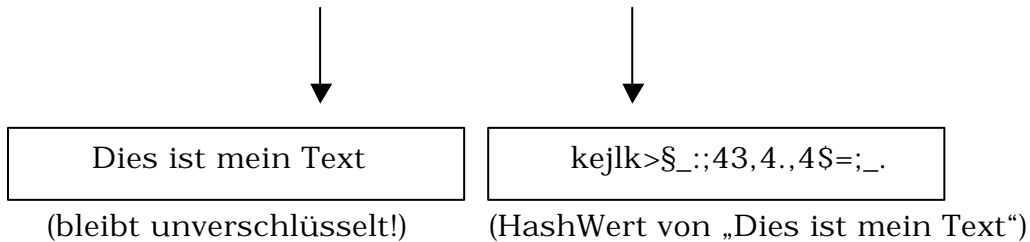
Person 1 will eine Nachricht an Person 2 schicken:

Dies ist mein Text

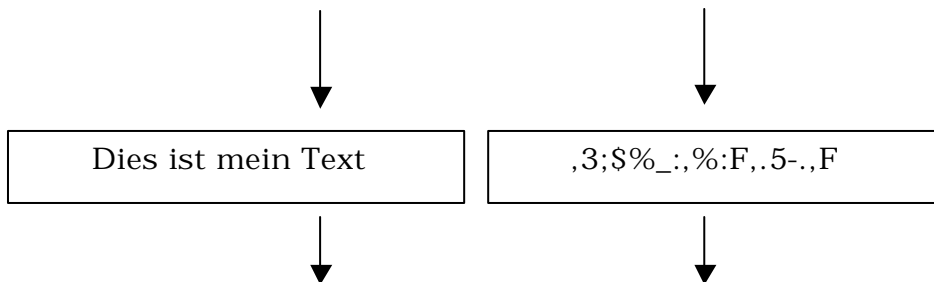


Eine HASH-Funktion erzeugt von diesem Text einen HASH (auch „Fingerprint“ genannt), der eine feste Länge (z. B. 128 Bit) besitzt. Die Bezeichnung „feste Länge“ bedeutet hierbei, dass die Länge des erzeugten Fingerprints stets 128 Bit beträgt, egal, ob der Text 10KB oder 10 MB groß ist.

Man besitzt nun den unkodierten Text und den HASH-Wert:



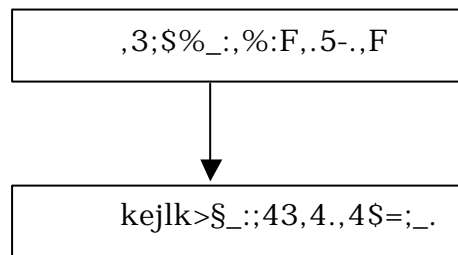
Der relativ kleine Fingerprint (z. B. 128 Bit) wird nun mit dem Private-Key von Person 1 verschlüsselt. Die unkodierte Nachricht bleibt unverändert.



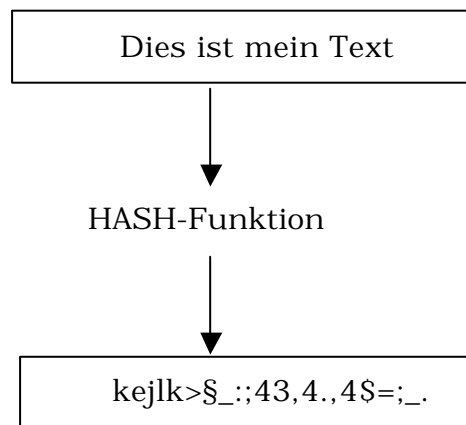
Der kodierte HASH-Wert wird an die unkodierte Nachricht gehängt und beides an Person 2 geschickt:

Dies ist mein Text
,3;\$%_:,%:F,.5-.,F

Beim Erhalt des „Pakets“ möchte Person 2 die Authentizität überprüfen und entschlüsselt dafür mit dem Public-Key von Person 1 den HASH-Wert:



Außerdem setzt Person 2 die (öffentlich bekannte) HASH-Funktion auf den zugeschickten Klartext an:



Nun kann Person 2 beide HASH-Werte vergleichen und somit prüfen, ob der Text von Person 1 stammt und ob dieser Text verändert wurde.

Anmerkung: Der Text kann immer gelesen werden, auch wenn Person 2 nicht über die HASH-Funktion verfügt.

8.4. Bekannte HASH-Funktionen

Es gibt zahlreiche gute und schlechte HASH-Funktionen. Zwei davon gelten mittlerweile wegen der verwendeten Algorithmen als sicher und zuverlässig:

- MD5: erzeugt HASH-Werte mit einer Länge von 128 Bit/16 Byte
- SHA-1: erzeugt HASH-Werte mit einer Länge von 160 Bit/20 Byte (Weiterentwicklung auf 192 Bit und 256 Bit)

8.5. Blinde Signaturen

8.5.1. Einleitung

Bei der blinden Signatur geht es darum, dass ein Signierer eine (digitale) Unterschrift leistet, ohne dass er die Nachricht selber kennt. Diese Art von Signaturen wird im elektronischen Zahlungsverkehr „eCash“ genutzt, um eine anonyme Geldübergabe zu ermöglichen.

8.5.2. Ein anschauliches Beispiel

Person A hat ein Blatt Papier, das er von Person B blind unterschreiben lassen möchte. Person A steckt das Blatt zusammen mit Kohlepapier in einen Umschlag, legt B diesen Umschlag vor und B unterzeichnet auf dem Umschlag. Durch das Kohlepapier bekommt A automatisch die Unterschrift auf seinem Blatt Papier und Person B weiß nicht, was auf dem Blatt steht, da sie nur den Umschlag gesehen hat.

8.5.3. Blinde Signatur - mathematisch

Das Dokument m einer Person A soll von B blind signiert werden. Seien n der Modul, e der öffentliche und d der geheime Schlüssel des Unterzeichners B.

- A wählt zunächst eine Zufallszahl r (den sogenannten „Blendparameter“), die teilerfremd zu n ist, potenziert diese mit e und schickt den Wert $x=(m*r)^e \bmod n$ an B.
- B unterschreibt den erhaltenen Wert, indem sie x mit d potenziert und das Ergebnis $y=x^d \bmod n$ an A zurückschickt.
- A multipliziert den erhaltenen Wert y mit dem Inversen von r : r^{-1} . Dadurch erhält A wegen

$$y \cdot r^{-1} = x^d \cdot r^{-1} = (m \cdot r^e)^d \cdot r^{-1} = m^d \cdot r^{ed} \cdot r^{-1} = m^d \cdot r \cdot r^{-1} = m^d \bmod n$$

die unterschriebene Nachricht. Das „Reinziehen“ der Potenz d in das Produkt $(m*r)^e$, also die Tatsache, dass

$$(m \cdot r^e)^d = m^d \cdot r^{ed}$$

gilt, entspricht der Verwendung des Kohlepapiers in dem oben genannten Beispiel.

Person B weiß nun nicht, was sie unterschrieben hat, denn dadurch, dass m mit der Zufallszahl r^e verschlüsselt wurde, kann sie aus $(m*r)^e$ nicht auf die Nachricht m schließen.

Anmerkung: die „Blinde Signatur“ im Bankwesen ist etwas komplizierter und gehört nicht gerade zum „Grundwissen“. Deshalb wurde e in eigenes Kapitel dafür erstellt: „11. Blinde Signatur (am Beispiel von E-Cash)“

9. PGP

9.1. Einleitung

Eine Möglichkeit, seine Daten zu schützen, ist das „Public-Key / Private-Key“-Verfahren, wie es in PGP (<http://www.pgpi.org>) u.a.¹ mit eMails gemacht wird.

Im Vergleich zur herkömmlichen Post kann man eine einfache, unverschlüsselte eMail wie eine Postkarte ansehen. Da sie nicht nur für den eigentlichen Empfänger lesbar ist, stellt sie keinen sicheren Weg einer Nachrichtenübermittlung dar.

Beim herkömmlichen Versenden von Dokumenten wird die Sicherheit durch Briefumschläge, Siegel und/oder Einschreiben erhöht.

Die PGP-Verschlüsselung entspricht im weitesten Sinne diesen Sicherheitsmaßnahmen bei der „Elektronischen Post“.

PGP steht für „Pretty Good Privacy“, was man frei mit „ganz gute Vertraulichkeit“ übersetzen kann.

An eine Datenkommunikation werden Ansprüche wie Erkennbarkeit des Absenders, Unversehrtheit der Nachricht und Vertraulichkeit gestellt. Diese Ansprüche werden durch ein System wie PGP erreicht.

Die PGP-Verschlüsselung hat zwei integrierte Verschlüsselungsverfahren:

Die eigentlichen Daten werden mit einem symmetrischen Verschlüsselungsverfahren (z.B. AES, CAST, IDEA, TripleDES, ...) und der dafür erzeugte Schlüssel mit einem asymmetrischen Verfahren (z. B. RSA, DH/DSS) geschützt.

9.2. Warum zwei Schlüssel?

Wie bereits in den vorangegangenen Abschnitten geschrieben, ist es für den Empfänger einer Nachricht nicht nur wichtig zu wissen, ob die Nachricht verändert oder von anderen gelesen wurde. Den gleichen Stellenwert bekommt die Authentizität, also die Gewissheit, dass sowohl Empfänger, als auch Absender der Nachricht diejenige sind, für die sie sich ausgeben.

Genau dies stellt PGP sicher.

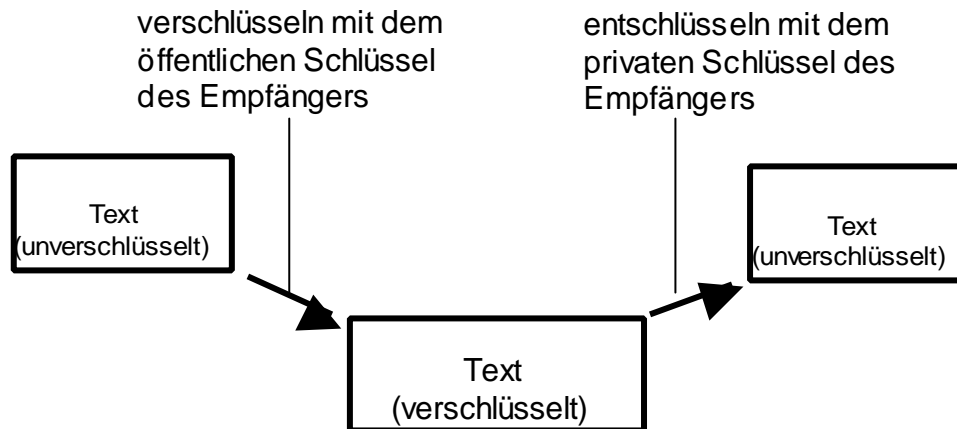
Auf der einen Seite benötigt eine Person einen Public-Key, der auf der eigenen Homepage hinterlegt oder als eMail-Anhang vertrieben werden kann. Des Weiteren ist ein Private-Key erforderlich, den diese Person unter Verschluss hält und auf keinen Fall an Dritte weitergibt.

¹ der Einsatz von PGP ermöglicht auch die Verschlüsselung von Dateien und sogenannten „Disc-Volumes“

Sind Daten mit dem Public-Key eines Empfängers verschlüsselt worden, so besteht die Möglichkeit der Entschlüsselung nur mit dem zugehörigen Private-Key, den ausschließlich der Empfänger der Nachricht besitzt.

Andersherum kann mit dem Private-Key etwas verschlüsselt werden, so dass die Empfänger mittels des Public-Keys nicht nur die Nachricht dekodieren, sondern auch gleich die Authentizität prüfen können.

Die eindeutige Erkennbarkeit des Absenders kann demnach durch PGP übermittelt werden, da hier die „elektronische Unterschrift“ integriert wurde.



9.3. ElGamal

9.3.1. Einleitung

Der bereits im Abschnitt 7 erklärte Diffie-Hellman-Algorithmus wird bei PGP in einer abgewandelten Form eingesetzt, die man als „ElGamal“¹ bezeichnet.

Korrekterweise sollten die Schlüssel also „ElGamal-Schlüssel“ und nicht „Diffie-Hellman-Schlüssel“ genannt werden.

9.3.2. Der grundsätzliche Ablauf

1. Schlüsselerzeugung
 - wähle eine große Primzahl p für die gilt: $(p-1)/2$ muss ebenfalls prim sein (N ist die Anzahl der Bits von p)
 - wähle eine Basis $\alpha < p$
 - wähle einen geheimen Schlüssel $a < p$
 - berechne $\beta = \alpha^a \bmod p$
 - veröffentliche p , α und β

¹ nach dem Erfinder T. ElGamal

2. Nachrichten-Verschlüsselung
 - wähle eine geheime Zufallszahl k , die zu $(p-1)$ teilerfremd ist
-> zur Erinnerung: teilerfremd = $\text{ggT}(k, p-1) = 1$
 - teile die zu kodierende Nachricht in Blöcke zu je $N-1$ Bit
 - berechne für jeden Block x die kodierte Nachricht:
 $e(x, k) = (y_1, y_2)$, wobei gelten muss

$$y_1 = \alpha^k \pmod{p}$$

$$y_2 = \beta^k x \pmod{p}$$

Dabei sind y_1 und y_2 jeweils Geheimtextblöcke der Länge N .

3. Nachrichten-Entschlüsselung
 - zerlege die kodierte Nachricht in N -Bit Blöcke
 - für jeweils zwei aufeinanderfolgende Blöcke y_1 und y_2 löse

$$y_1^a x = y_2 \pmod{p}$$

nach x auf. Demnach ist $d(y_1, y_2) = x = y_2 (y_1^a)^{-1} \pmod{p}$ die gesuchte dekodierte Nachricht.

9.3.3. Ein einfaches Beispiel - Schritt für Schritt

1. Schlüsselerzeugung
 - wähle eine Primzahl p für die gilt, dass $(p-1)/2$ ebenfalls prim ist
 $p = 2063$ $N=12$
 - wähle eine Basis $\alpha < p$
 $\alpha = 300$
 - wähle einen geheimen Schlüssel $a < p$
 $a = 400$
 - berechne $\beta = \alpha^a \pmod{p}$
 $\beta = 300^{400} \pmod{2063} = 1624$
 - veröffentliche p , α und β
...
2. Nachrichten-Verschlüsselung
 - wähle eine geheime Zufallszahl k , die zu $(p-1)$ teilerfremd ist
 $k = 747$
 - teile die zu kodierende Nachricht in Blöcke zu je $N-1$ Bit
 $(N-1) = 11$ Nachricht = „Text“
„T“ = 84_d „e“ = 101_d „x“ = 120_d „t“ = 116_d

Aus Gründen der Übersichtlichkeit sollen hier lediglich einzelne Dezimalzahlen genutzt und jeweils Nullen vorangestellt werden, um $N-1=11$ Bit zu erhalten. Dies ist weniger effizient (mehr Rechenarbeit), aber dafür sicherlich deutlicher und somit nachvollziehbar.

Es muss nur darauf geachtet werden, dass in der folgenden Formel das „x“ (also der Code des Klartextes) kleiner als p ist, also in diesem Fall kleiner oder gleich 2062.

- berechne für jeden Block x die kodierte Nachricht:

„T“ = 84_d

$$\rightarrow e(x, k) = (y_1, y_2)$$

$$\begin{aligned}\rightarrow y_1 &= \text{alpha}^k \pmod{p} \\ &= 300^{747} \pmod{2063} \\ &= 33\end{aligned}$$

$$\begin{aligned}\rightarrow y_2 &= \text{beta}^k \cdot x \pmod{p} \\ &= 1624^{747} \cdot 84 \pmod{2063} \\ &= 1495\end{aligned}$$

$$\rightarrow e(84, 747) = (33, 1495)$$

„e“ = 101_d

$$\rightarrow e(x, k) = (y_1, y_2)$$

$$\begin{aligned}\rightarrow y_1 &= (\text{bereits erledigt; ändert sich nicht}) \\ &= 33\end{aligned}$$

$$\begin{aligned}\rightarrow y_2 &= \text{beta}^k \cdot x \pmod{p} \\ &= 1624^{747} \cdot 101 \pmod{2063} \\ &= 1773\end{aligned}$$

$$\rightarrow e(101, 747) = (33, 1773)$$

„x“ = 120_d

$$\rightarrow e(x, k) = (y_1, y_2)$$

$$\rightarrow y_1 = 33$$

$$\begin{aligned}\rightarrow y_2 &= \text{beta}^k \cdot x \pmod{p} \\ &= 1624^{747} \cdot 120 \pmod{2063} \\ &= 1841\end{aligned}$$

$$\rightarrow e(120, 747) = (33, 1841)$$

„t“ = 116_d

$$\rightarrow e(x, k) = (y_1, y_2)$$

$$\rightarrow y_1 = 33$$

$$\begin{aligned}\rightarrow y_2 &= \text{beta}^k \cdot x \pmod{p} \\ &= 1624^{747} \cdot 116 \pmod{2063} \\ &= 198\end{aligned}$$

$$\rightarrow e(116, 747) = (33, 198)$$

3. Nachrichten-Entschlüsselung

- zerlege die kodierte Nachricht in N-Bit Blöcke (erledigt)

- für jeweils zwei aufeinanderfolgende Blöcke löse nach x auf:
(33,1495)

$$\begin{aligned}\rightarrow x &= y_2 \cdot (y_1^a)^{-1} \bmod p \\ &= 1495 \cdot (33^{400})^{-1} \bmod 2063 \\ &= 84 \rightarrow 84_d = „T“\end{aligned}$$

$$(33,1773)$$

$$\begin{aligned}\rightarrow x &= y_2 \cdot (y_1^a)^{-1} \bmod p \\ &= 1773 \cdot (33^{400})^{-1} \bmod 2063 \\ &= 101 \rightarrow 101_d = „e“\end{aligned}$$

$$(33,1841)$$

$$\begin{aligned}\rightarrow x &= y_2 \cdot (y_1^a)^{-1} \bmod p \\ &= 1841 \cdot (33^{400})^{-1} \bmod 2063 \\ &= 120 \rightarrow 120_d = „x“\end{aligned}$$

$$(33,198)$$

$$\begin{aligned}\rightarrow x &= y_2 \cdot (y_1^a)^{-1} \bmod p \\ &= 198 \cdot (33^{400})^{-1} \bmod 2063 \\ &= 116 \rightarrow 116_d = „t“\end{aligned}$$

Als verschlüsselte Nachricht ergibt sich für

Text

demnach

1495 1773 1841 198

mit den oben genannten Parametern.

10. Base64

10.1. Einleitung

Der Text einer eMail muss als reiner ASCII-Text übertragen werden. Mit der Zeit wurden die Ansprüche der Benutzer aber immer größer, so dass heutzutage jedes Standard-eMail-Programm auch Anhänge zulässt. Dies können Binärdateien, Grafiken, HTML-Dokumente usw. sein.

Um nun kompatibel zu den ursprünglichen Protokollen und Leitungen zu bleiben, bedient man sich der Umkodierung von 8Bit auf 6Bit.

10.2. Das Base64-Alphabet

Table 1: The Base64 Alphabet

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

10.3. Das ASCII-Alphabet

ASCII Hex Symbol			ASCII Hex Symbol			ASCII Hex Symbol			ASCII Hex Symbol		
0	0	NUL	16	10	DLE	32	20	(space)	48	30	0
1	1	SOH	17	11	DC1	33	21	!	49	31	1
2	2	STX	18	12	DC2	34	22	"	50	32	2
3	3	ETX	19	13	DC3	35	23	#	51	33	3
4	4	EOT	20	14	DC4	36	24	\$	52	34	4
5	5	ENQ	21	15	NAK	37	25	%	53	35	5
6	6	ACK	22	16	SYN	38	26	&	54	36	6
7	7	BEL	23	17	ETB	39	27	'	55	37	7
8	8	BS	24	18	CAN	40	28	(56	38	8
9	9	TAB	25	19	EM	41	29)	57	39	9
10	A	LF	26	1A	SUB	42	2A	*	58	3A	:
11	B	VT	27	1B	ESC	43	2B	+	59	3B	;
12	C	FF	28	1C	FS	44	2C	,	60	3C	<
13	D	CR	29	1D	GS	45	2D	-	61	3D	=
14	E	SO	30	1E	RS	46	2E	.	62	3E	>
15	F	SI	31	1F	US	47	2F	/	63	3F	?

ASCII Hex Symbol			ASCII Hex Symbol			ASCII Hex Symbol			ASCII Hex Symbol		
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	
77	4D	M	93	5D]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	□

10.4. Beispiel 1

Der 3-stellige String „rlTM“¹ soll kodiert werden:

r	?	104d	?	0111	00102	}	siehe ASCII-Tabelle
l	?	108d	?	0110	11002		
TM	?	153d	?	1001	10012		

0111 0010 0110 1100 1001 1001
 011100100110110010011001
011100100110110010011001

00011100 ₂	?	28d	?	c	}	siehe Base64-Alphabet (Beispiel: 0->A, 1->B,...)
00100110 ₂	?	38d	?	m		
00110010 ₂	?	50d	?	y		
00011001 ₂	?	25d	?	Z		

Demnach ergibt sich für den String „rlTM“ eine Umkodierung nach „cmyZ“ als Base64-Nachricht.

Bei der Dekodierung wird genau umgekehrt verfahren:

c	?	28d	?	00011100 ₂
m	?	38d	?	00100110 ₂
y	?	50d	?	00110010 ₂
Z	?	25d	?	00011001 ₂

00 011100 00 100110 00 110010 00 011001
 011100 100110 110010 011001
 011100100110110010011001
011100100110110010011001

01110010 ₂	?	104d	?	r	}	siehe ASCII-Tabelle
01101100 ₂	?	108d	?	l		
10011001 ₂	?	153d	?	TM		

¹ „l“ ist das kleine „L“ (nicht die Eins) und „TM“ ist als ein Zeichen zu betrachten

10.5. Beispiel 2

Bei Beispiel 1 wurde ein 3-stelliger Text kodiert, was bei der Aufteilung in 4 Blöcke à 6 Bits sehr vorteilhaft ist. Was passiert aber nun bei einem Text, mit 4 oder 5 Zeichen?

Dann wird ein sogenanntes „PAD-Feld“ eingesetzt, das zum „Auffüllen“ benutzt wird:

W?	87 _d	?	0101 0111 ₂
o?	111 _d	?	0110 1111 ₂
r?	114 _d	?	0111 0010 ₂
t?	116 _d	?	0111 0100 ₂

```

01010111      01101111      01110010      01110100
  01010111011011110111001001110100
    01010111011011110111001001110100
010101 110110 111101 110010      011101 00
  
```

Die erste 4er-Gruppe stellt einen erlaubten Base64-Block dar. Allerdings ist der nächste Block nicht vollständig. Deshalb:

- Der 1. Teil „011101“ bleibt bestehen.
- Der 2. Teil „00“ wird mit vier Nullen aufgefüllt, so dass „000000“ entsteht.

Die Teile 3 und 4 werden zum PAD-Feld und bei der folgenden Umwandlung nicht betrachtet:

010101 ₂	→	21 _d	→	V	}	erster Base64-Vierer-Block	
110110 ₂	→	54 _d	→	2			
111101 ₂	→	61 _d	→	9			
110010 ₂	→	50 _d	→	y			
						}	zweiter Base64-Vierer-Block (Teil 3+4 vorerst unberücksichtigt!)
011101 ₂	→	29 _d	→	d			
000000 ₂	→	0 _d	→	A			

Übrig bleiben jetzt noch die genannten Teile 3 und 4, die jeweils mit „=“, also 61_d bzw. 111101₂ besetzt werden. Dementsprechend ergibt sich aus der Konvertierung des Strings „**Wort**“ folgender Base64-String:

V29ydA==

Die Rückkodierung erfolgt entsprechend:

```
V   → 21d → 000101012
2   → 54d → 001101102
9   → 61d → 001111012
y   → 50d → 001100102
d   → 29d → 000111012
A   → 0d  → 000000002
=   → PAD-Feld, bleibt unbeachtet
=   → PAD-Feld, bleibt unbeachtet
```

Hieraus ergibt sich folgender Binär-String, der wiederum in 8Bit-Strings umgewandelt wird:

```
00010101 00110110 00111101 00110010 00011101 00000000
010101    110110    111101    110010    011101    000000
010101110110111101110010011101000000
01010111011011110111001001110100
01010111011011110111001001110100
01010111 01101111 01110010 01110100
0101 01112 ? 87d ? W
0110 11112 ? 111d ? o
0111 00102 ? 114d ? r
0111 01002 ? 116d ? t
```

Daraus folgt, dass der Base64-String „**V29ydA==**“ dem 8Bit-String „**Wort**“ entspricht.

10.6. Beispiel 3

Die Kodierung eines 5-stelligen Strings erfolgt entsprechend den Ausführungen von Beispiel 1 und Beispiel 2 und soll nur noch in Kurzform dargestellt werden. Das zu kodierende Wort lautet „Hallo“.

```
H   ? 72d   ? 0100 10002
a   ? 97d   ? 0110 00012
l   ? 108d  ? 0110 11002
l   ? 108d  ? 0110 11002
o   ? 111d  ? 0110 11112
```

01001000	01100001	01101100	01101100	01101111		
010010	000110	000101	101100	011011	000110	111100
18d	6d	5d	44d	27d	6d	60d
S	G	F	s	b	G	8

Erster Vierer-Base64-Block : „**SGFs**”

Zweiter Vierer-Base64-Block: „bG8” ? hier fehlt 1 Byte, da es sich lediglich um ein 3-Byte-String handelt ⇒ „=“ anhängen
 ? ⇒ „**bG8=**”

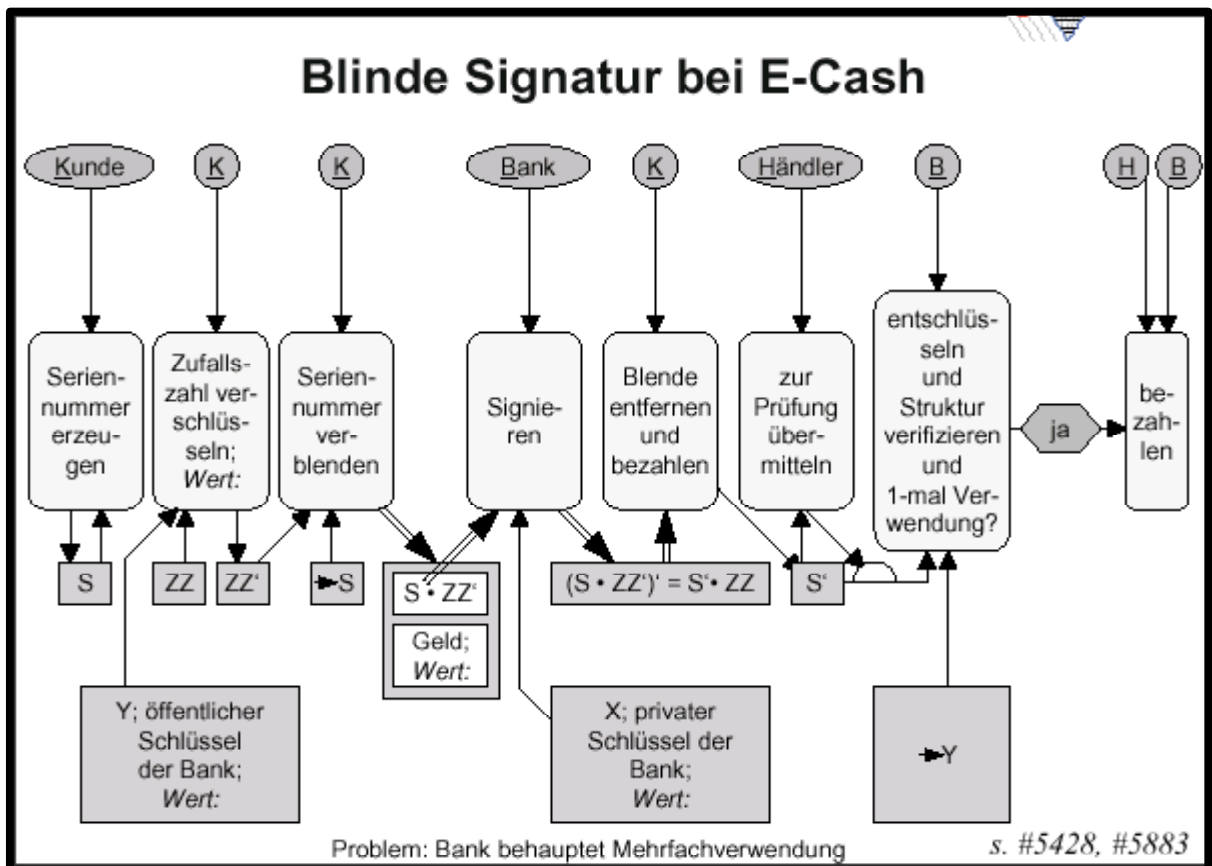
Somit ist „**Hallo**“ base64-kodiert „**SGFsbG8=**”.

11. Blinde Signatur (am Beispiel von E-Cash)

11.1. Anmerkung

Auf die „Blinde Signatur“ wurde bereits schon vorher eingegangen, doch soll hier mittels einer Grafik „E-Cash“ erläutert werden.

11.2. E-Cash



(Quelle: unbekannt)

- Kunde erzeugt Seriennummer S , verbunden mit dem privaten Schlüssel, der der Bank bekannt ist
 - Kunde nimmt ZZ und verbindet sie mit dem öffentlichen Schlüssel der Bank (Y)
 - > es gibt mehrere öffentliche Bank-Schlüssel, die jeweils an einem bestimmten Geldbetrag gebunden sind
 - > Ergebnis ist ZZ'
 - Kunde verblendet seine Seriennummer S mit ZZ' -> $S \cdot ZZ'$
 - Bank signiert mit ihrem privaten Schlüssel: $(S \cdot ZZ')'$
 - Kunde holt die ZZ wieder heraus -> $(S \cdot ZZ')' = S \cdot ZZ$ und erhält somit S
- Er kann $S \cdot ZZ$ trennen, da ihm ZZ bekannt ist. Die Bank allerdings kann diese Trennung nicht vornehmen! (Stichwort: Faktorisierung, siehe RSA)

- Kunde bezahlt mit S' den Händler
- Händler übergibt S' an Bank
- Bank prüft S' , indem sie ihren eigenen Schlüssel Y herauszieht
- wenn S' okay: Bank zahlt Betrag an Händler
- Bank speichert S' , um Doppelnutzung zu vermeiden

Das Herausrechnen von $(S * ZZ')' = S' * ZZ$ ist anscheinend ein mathematisches Problem. Es dürfte ein mathematisches Gesetz gelten, so dass im Prinzip folgendes möglich ist:

$$S = 3; ZZ = 5; ZZ' = 5^{1/4}$$

$$(S * ZZ') = S' * ZZ$$

$$(3 * 5^{1/4})^4 = 3^4 * 5$$

Wie das aber wirklich gelöst wird, ist unbekannt. (Und konnte auch auf Rückfrage nicht in der Vorlesung geklärt werden.)

ZZ wird als „Blendparameter“ oder auch „blinding factor“ bezeichnet.

12. Kurz und knapp

12.1. Anmerkung zu diesem Kapitel

Während des Semesters sind immer mal wieder Abkürzungen oder Fachbegriffe aufgetaucht, die hier ganz kurz erklärt werden sollen.

Diese Formulierungen sind größtenteils aus dem Siemens Lexikon und haben nicht den Anspruch vollständig zu sein bzw. sollten nicht als Eigenleistung gewertet werden.

12.2. Was ist ein Gateway?

Unter einem Gateway versteht man die Hard- und Software, um verschiedene Netze miteinander zu verbinden oder an andere Netze durch Protokollumsetzung anzuschließen.

Ein Gateway hat die Aufgabe, Nachrichten von einem Rechnernetz in ein anderes zu übermitteln, wofür vor allem die Übersetzung der Kommunikationsprotokolle notwendig ist; es kann also auch als eine Art Protokollkonverter betrachtet werden.

12.3. Was ist ein Proxy?

Proxy heißt „Bevollmächtigter“ oder „Stellvertreter“. Ein Proxy-Server ermöglicht Systemen, die keinen direkten Zugang zum Internet haben, den indirekten Zugang zum Netz. Das können solche Systeme sein, die durch einen Firewall aus Sicherheitsgründen vom unmittelbaren Zugang ausgeschlossen sind.

Ein Proxy kann einzelne Datenpakete aus dem Datenstrom zwischen dem Internet und einem lokalen Netz herausfiltern und so zur Erhöhung der Sicherheit beitragen. Proxies werden auch dazu benutzt, Zugriffe auf bestimmte Server zu begrenzen. Allerdings können beim Zugang über Proxy-Server Internet-Dienste wie zum Beispiel Telnet ausgeschaltet sein und dem Nutzer nicht alle Internet-Ressourcen zur Verfügung stehen.

12.4. Was ist eine Firewall?

Unter Firewalls versteht man Netzwerkkomponenten, über die ein internes, privates Unternehmensnetzwerk an ein öffentliches Netzwerk angekoppelt wird: also ein gesichertes Netzwerk an ein ungesichertes.

Die Aufgabe von Firewalls ist es, durch verschiedene Mechanismen die Sicherheit im Unternehmensnetz zu erhöhen. Dazu gehören ein möglichst ungestörter Zugriff auf das öffentliche Netzwerk, die Verhinderung eines unberechtigten Zugriff auf das eigene Netzwerk, die Beschränkung von extern nutzbaren Diensten, die Beschränkung auf eine begrenzte Zahl von Kommunikationsrechnern, die Authentifizierung und Identifikation sowie die Datenverschlüsselung.

12.5. Was ist TCP?

TCP ist ein verbindungsorientiertes Transportprotokoll für den Einsatz in paketvermittelten Netzen. Das Protokoll baut auf dem IP-Protokoll auf, unterstützt die Funktionen der Transportschicht und stellt vor der Datenübertragung eine gesicherte Verbindung zwischen den Instanzen her. Die Daten der höheren Schichten werden durch TCP nicht verändert, sondern lediglich segmentiert und in einzelnen Datenpaketen versendet, die einen Umfang von bis zu 65 kByte haben können.

Das darunter liegende IP-Protokoll fragmentiert die TCP-Datensegmente in kleinere Datenpakete. Jedes Oktett eines Segments wird von TCP mit einer sogenannten Sequenznummer versehen, was empfangsseitig die richtige Reihenfolge garantiert.

Die wesentlichen Dienstleistungen, die das TCP-Protokoll in Verbindung mit dem IP-Protokoll für die Anwendungsprozesse bereitstellt, sind die Ende-zu-Ende-Kontrolle, das Verbindungs-Management, die Flusskontrolle, die Zeitkontrolle und das Multiplexen von Verbindungen sowie die Fehlerbehandlung.

12.6. Was ist SNMP?

Das SNMP-Protokoll erlaubt ein zentrales Netzwerkmanagement für viele Netzwerkkomponenten. Die primären Ziele von SNMP sind die Verringerung der Komplexität der Management-Funktionen, die Erweiterbarkeit des Protokolls und die Unabhängigkeit von irgendwelchen Netzwerkkomponenten.

Es gibt drei Managementbereiche, in denen dieses Protokoll eingesetzt werden kann:

- * Beim Monitoring können Ereignisse aufgezeichnet und Netzwerkstatistiken gesammelt werden.
- * Das Controlling unterstützt das Ändern von Geräteparametern und Gerätevariablen.
- * Bei der Administrierung werden Informationen aufgezeichnet, die die Entwicklung des Netzwerk-Aufbaus beschreiben.

Die Netzwerkkomponenten Router, Host, Bridge oder Terminal-Server haben Management-Agenten, die die Management-Funktionen ausführen. Mittels SNMP findet zwischen der Managementstation und den Netzwerkkomponenten eine Kommunikation statt. Diese Kommunikation umfasst die Übertragung relevanter Management-Daten, die von den Netzwerkkomponenten gesammelt wurden und in der MIB abgelegt sind und zur Managementstation übertragen werden, ebenso wie die Steuerungsdaten für die Netzwerkkomponenten.

12.7. Was ist ein VPN?

Der Begriff VPN wird in mehreren Bedeutungen verwendet.

Generell handelt es sich bei einem Virtual Private Network (VPN) um ein geschlossenes, logisches Netz, das auf unterschiedlichen Schichten des OSI-Referenzmodells aufsetzt und für eine bestimmte Benutzergruppe etabliert wird.

VPNs setzen in der Regel auf den Schichten 2 oder 3 des OSI-Schichtenmodells auf und verwenden Tunneling-Mechanismen für den IP-Verkehr.

Ein VPN nutzt immer die öffentlich zugänglichen Übertragungsnetzwerke, bei denen die Verbindungen durch einen öffentlichen Carrier bereitgestellt werden. Der Anwender bildet sich über diese Übertragungswege praktisch sein privates Netz. Er verfügt über Sicherheitsmechanismen, wie die Identifikation und die Authentifikation der Netzteilnehmer, so, dass sich Unbefugte keinen Zugang zum VPN verschaffen können. Die Dienstleistungen werden über ein öffentliches Netz erbracht, wodurch der Anwender auch die Ressourcen des Carriers nutzen kann.

12.8. Was ist HTTP/FTP?

HTTP ist ein allgemeines, statusloses, objektorientiertes Protokoll zur Datenübertragung im Rahmen des World Wide Web (WWW). Es beschreibt einen definierten Satz von Nachrichten und Antworten, mit denen ein Client und ein Server während einer HTML-Sitzung kommunizieren. Jede Anfrage eines Web-Browsers an einen Web-Server nach einem neuen Dokument stellt eine neue Verbindung dar.

Das HTTP-Protokoll dient der Adressierung der Objekte über URL, es wickelt die Interaktion zwischen Client und Server ab und sorgt für die Anpassung der Formate zwischen Client und Server.

Das Serverprogramm zu HTTP ist `httpd`. Es beantwortet Anfragen von WWW-Clients.

Das File-Transfer-Protokoll (FTP) dient dem Dateitransfer zwischen verschiedenen Systemen und der einfachen Dateihandhabung. FTP basiert auf dem Transportprotokoll TCP und kennt sowohl die Übertragung zeichencodierter Information als auch von Binärdaten. Die Dateiübertragung wird vom lokalen System aus gesteuert, die Zugangsberechtigung für das Zielsystem wird für den Verbindungsaufbau mittels User-Identifikation und Passwort überprüft.

Will ein Client mit dem Server kommunizieren, baut der Benutzer über den Interpreter im Client eine Verbindung zum Interpreter im Server auf. Über diese Steuerverbindung kommunizieren Client und Server über einen Satz festgelegter Kommandos. Zum Austausch der Nutzdaten baut der Server eine zweite Verbindung zum Client auf, und zwar wird diese Verbindung von dem Datentransferprozess (DTP) gesteuert.

Über diese Verbindung werden die Nutzdaten übertragen. Beim Verbindungsabbau bestätigt der Server-Interpreter das Ende des Datentransfers über die Steuerverbindung an den Interpreter des Clients.

12.9. Was ist SMTP?

Das SMTP ist der Internet-Standard zur Verteilung von elektronischer Post. Das Protokoll ist textorientiert und setzt auf dem TCP-Protokoll auf. Eine Nachricht besteht aus Header und Nutzdaten.

Der Header enthält u. a. Datum, Bezug, Empfänger, Absender, Kopienempfänger usw. Der Nutzdatenteil besteht typischerweise aus freiem ASCII-Text. Nachrichten mit mehreren Empfängern auf einem Ziel-Host werden nur einmal zum Ziel übertragen und dort verteilt.

SMTP definiert nicht, wie eine Nachricht von einem oder zu einem Benutzer von SMTP vermittelt wird. Des Weiteren ist nicht festgelegt, wie empfangene Nachrichten vom Benutzer präsentiert oder zwischengespeichert werden. Diese Aufgaben werden für das SMTP-Protokoll von anderen Applikationsprogrammen durchgeführt.

13. Über dieses Dokument

13.1. Sinn dieser Dokumentation

Diese Dokumentation hat sicherlich nicht den Anspruch, ein Skript für die Vorlesung „Protokolle höherer Schichten“ zu sein. Um ein Skript (oder Buch) über Kryptographie schreiben zu können, reicht unser Wissen nicht aus, was sicher auch der ein oder andere Professor bestätigen wird.

Unser Wissen reicht im Maximum zum Pommes-Verkäufer, also machen wir jeden Leser darauf aufmerksam, dass durchaus noch Fehler in diesem Dokument enthalten sein können.

13.2. Nutzungsrecht

Jeder Schüler/Student darf dieses Dokument für private Zwecke bzw. als Lernmittel nutzen.

Professoren benötigen eine schriftliche Genehmigung, wenn sie dieses Dokument (komplett oder Teile davon) für ihre Vorlesung nutzen möchten. Schreiben Sie in diesem Fall eine eMail an Genehmigung@AmelFin.de

Die Weitergabe der Dokumentation ist erlaubt, sofern keine Änderungen vorgenommen wurden und die Weitergabe kostenlos ist.

13.3. Aktuelle Version

Eigentlich ist für uns das Thema „Kryptographie“ in Bezug auf diese Dokumentation und die genannte Vorlesung erledigt, doch könnte es sein, dass noch Updates kommen; insbesondere, wenn sich trotz großer Sorgfalt noch Fehler eingeschlichen haben.

Die aktuelle (und somit einzig offizielle Version) liegt auf

<http://www.AmelFin.de>

zum Download bereit.