3. Kryptographie

3.1. Rechnen mit großen Zahlen

Wenn die Zahlen so groß werden, dass sie nicht mehr in die Register des Rechners passen, so muss man die Berechnungen softwaremäßig implementieren.

Dies läßt sich auf das Rechnen mit Polynomen zurückführen. Sei B eine so gewählte Basis, dass die Zahl B^2 noch in das Register des Rechners passt. Dann schreiben wir eine große Zahl X>B als Polynom in B:

$$X = x_0 + x_1 B + x_2 B^2 + \ldots + x_{n-1} B^{n-1} = \sum_{k=0}^{n-1} x_k B^k$$

wobei $0 \le x_i < B$. Analog

$$Y = y_0 + y_1 B + y_2 B^2 + \ldots + y_{n-1} B^{n-1} = \sum_{k=0}^{n-1} y_k B^k$$

Addition: $X + Y = \sum_{k=0}^{n-1} (x_k + y_k) B^k$

Dabei sind Werte $B<(x_k+y_k)<2B$ möglich. In diesem Fall ist ein Übertrag auf k+1 zu realisieren.

Aufwand des Verfahrens: O(n). [n ist die Zahl der Terme in den obigen Polynomen für X und Y. Durch Einsetzen von $x_i=B-1$ zeigt man, dass $X=B^n-1$ die größte darstellbare Zahl ist.]

Multiplikation: Entsprechend der Multiplikation von Polynomen gilt:

$$X \cdot Y = \sum_{k=0}^{2(n-1)} \left(\sum_{i=0}^{k} x_i y_{k-i} \right) B^k$$

Jeder einzelne Produktterm kann Werte $B<(x_i y_{k-i})<B^2$ annehmen. Sobald dies passiert, ist ein Übertrag auf k+1 zu realisieren.

Aufwand des Verfahrens: $O(n^2)$.

Beispiel für B=100:

$$12208 \cdot 14332 = (1 \cdot 100^2 + 22 \cdot 100^1 + 8) \cdot (1 \cdot 100^2 + 43 \cdot 100^1 + 32)$$

$$= 8 \cdot 32 + (8 \cdot 43 + 22 \cdot 32) \cdot 100^1 + (8 \cdot 1 + 22 \cdot 43 + 1 \cdot 32) \cdot 100^2 +$$
$$(22 \cdot 1 + 1 \cdot 43) \cdot 100^3 + (1 \cdot 1) \cdot 100^4 \qquad | \; 9 = 3 \cdot 3 = n^2 \text{ Multiplikationen}$$

$= 256 + 1048 \cdot 100^1 + 986 \cdot 100^2 + 65 \cdot 100^3 + 1 \cdot 100^4$

$= 56 + (2 + 48 + 10 \cdot 100^1) \cdot 100^1 + (86+9 \cdot 100^1) \cdot 100^2 + 65 \cdot 100^3 + 1 \cdot 100^4$

$= 56 + 50 \cdot 100^1 + (10+86) \cdot 100^2 + (9+65) \cdot 100^3 + 1 \cdot 100^4$

$= 56 + 50 \cdot 100^1 + 96 \cdot 100^2 + 74 \cdot 100^3 + 1 \cdot 100^4 = 1\ 74\ 96\ 50\ 56$

Hinzu kommt noch der Aufwand, für das Produkt (mod m) zu rechnen. Das ist zwar bei m=B=100 einfach, denn 1 74 96 50 56 (mod 100) ist einfach 56, aber bei anderen Zahlen m ist das ein Aufwand in sich.

Man braucht aber nicht unbedingt n^2 Multiplikationen, sondern kann den Aufwand mit dem CRT (Satz S1-5) auf n Multiplikationen reduzieren.

Beispiel: Berechne (a·b) = (220·599), wenn der Rechner nur zweistellige Zahlen multiplizieren kann.

Lösung mit CRT: Man sucht zunächst teilerfremde Zahlen $m_1,\ldots,m_n$, die im Produkt eine Zahl größer als das größte erwartete Ergebnis liefern. Hier im Beispiel reichen 3 hohe zweistellige Zahlen, z.B. $97 \cdot 98 \cdot 99 \approx 1.000.000$, da das Produkt 220·599 im niedrigen 6-stelligen Bereich zu erwarten ist. Dann stellt man jede Zahl bezüglich ihrer Reste dar

	mod 97	mod 98	mod 99
220	26	24	22
599	17	11	5
220·599	26·17 = 54	24·11 = 68	22·5 = 11

Hierzu sind nur n Multiplikationen im zweistelligen Zahlbereich nötig UND NICHT n^2. Gemäß dem Chinesischen Restsatz (CRT) sind alle Zahlen zwischen 0 und 97·98·99-1 eindeutig durch ihre drei Reste bzgl. 97, 98 und 99 charakterisiert, also charakterisiert das Tripel

(54, 68, 11)

eindeutig die Produktzahl 220·599.

Es soll nicht verschwiegen werden, dass zum Rücktransformieren auch noch einige Multiplikationen nötig sind, nach Satz S1-7 (Explizite Lösung für CRT) hat man ja zu bilden

$$x = (a_1 \cdot M_1 \cdot N_1 + \ldots + a_n \cdot M_n \cdot N_n) \ \text{mod} \ m$$

was in diesem Fall bedeutet:

$$x = (54 \cdot 98 \cdot 99 \cdot N_{97} + 68 \cdot 97 \cdot 99 \cdot N_{98} + 11 \cdot 97 \cdot 98 \cdot N_{99}) \ \text{mod} \ m \qquad (*)$$

mit bestimmten multiplikativen Inversen N_{97}, N_{98} und N_{99}, die jedoch vorweg gerechnet werden können. Aber manchmal braucht man nicht rücktransformieren, sondern man kann mit dem Tripel (54, 68, 11) weiterrechnen.

Oder man muß zwar rücktransformieren, kann aber gleich die Faktorisierung in Gl. (*) nutzen, um x mod m oder x mod q, worin q eine weitere Zahl q<m ist, relativ bequem auszurechnen.

Praktisch zeigen mit CrypTool – Einzelverfahren – Anwendungen des CRT – Modulare Hin- und Rücktransformation.

Modulo-Berechnungen: X mod Y läßt sich analog zu einer Polynomdivision durchrechnen (nach dem Muster der schriftlichen Division)

Beispiel für B=10:

$27800 \bmod 117 = (2B^4 + 7B^3 + 8B^2) \bmod (1B^2+1B^1+7) = 2B^2 + 5B - B - 3 = \mathbf{237}$

$-(2B^4 + 2B^3+14B^2)$

$5B^3 - 6B^2$

$-(5B^3 + 5B^2 +35B)$

$-1B^3 - 4B^2 - 5B$ ← Überträge beachten!!

$-(-1B^3 - B^2 - 7B)$

$-3B^2 +2B$

$-(-3B^2 - 3B - 21)$ ← Übertrag beachten!!

$7B + 1 = \mathbf{71}$

Die rot eingekreisten Zahlen sind die jeweiligen Zielwerte, die man durch Multiplikation von $1B^2$ mit geeignetem Faktor erreichen muss, also z.B. $2B^4 = 1B^2 \cdot 2B^2$.

"Überträge beachten!" heißt für B=10: $35B = 3B^2+5B$ sowie in gleicher Zeile $-6B^2-5B^2-3B^2 = -14B^2 = -1B^3 - 4B^2$.

Es gilt also $27800 \bmod 117 = \mathbf{71}$, weil $27800 : 117 = \mathbf{237} \cdot 117 + 71$
(wobei **237** aus $2B^2+5B-B-3 = 200+50-10-3$ folgt).

Potenzen modulo m: Bei $X^k \bmod m$ verbietet sich eine direkte Berechnung von X^k auf jeden Fall, denn wenn X und k jeweils 1000stellige Zahlen sind, was in der Kryptographie nichts Ungewöhnliches ist, dann hätte X^k etwa 10^{3000} Bits – und dies ist eine Zahl, die die Anzahl

der Atome im Weltall bei weitem überschreitet. Glücklicherweise liefert uns Folgerung S1-4 einen Trick, mit dem wir die Berechnung durchführen können:

1. Zerlege k in seine Binärdarstellung:
$$k = \sum_{i=0}^{n-1} k_i 2^i = (\ldots((k_{n-1} \cdot 2 + k_{n-2}) \cdot 2 + \ldots + k_1) \cdot 2 + k_0$$
2. Wende die Binärdarstellung des Exponenten in der Form fortgesetzen Quadrierens an (aus "mal 2" wird "hoch 2", aus "+b" wird "$\cdot X^b$")
3. Halte dabei die Zahlen durch fortgesetztes "mod m" klein (s. Folgerung S1-4)

Beispiel: Berechne $2208^5 \mod 7$.

Wir zerlegen 5 = (**1**·2 + **0**)2 + **1** [5 = **101**$_{\text{binär}}$] und rechnen aus, dass 2208 mod 7 = 3 gilt

$2208^5 \mod 7$

$= ((2208^1)^{2+0})^{2+1} \mod 7$ | jedes "+1" im Exp. durch "· 3" ersetzen

$= ((\ 3\ )^{2+0})^{2} \cdot 3 \mod 7$

$= (\ 3^2\ )^{2} \cdot 3 \mod 7$ | $3^2 = 9 = 2 \pmod 7$

$= (\ 2\ )^{2} \cdot 3 \mod 7 \quad = \quad 4 \cdot 3 \mod 7 \quad \mathbf{= 5}$

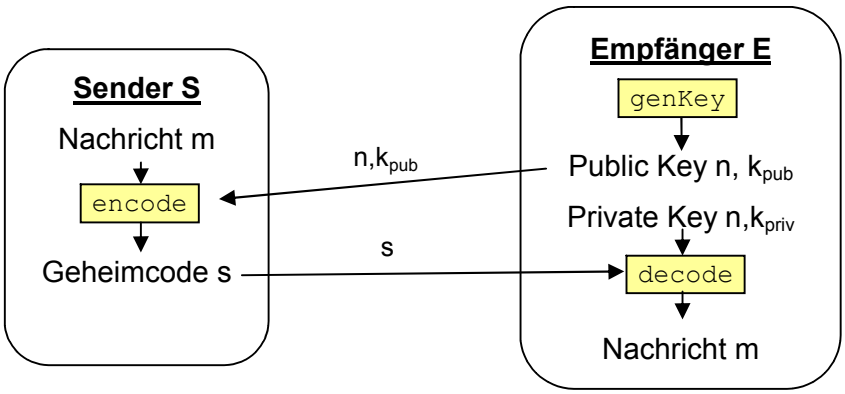
Eigentlich müßten in der 2. Zeile die Klammern im Exponenten stehen (wäre aber etwas unübersichtlicher). Folgt man jedoch, wenn 2208 den Rest 3 (mod 7) hat, der Regel: <<Ersetze jedes „+1" im Exponenten durch einen Term „·3">>, so ist alles o.k. Dann die Klammern von innen nach außen auflösen und jede Basiszahl bei Überschreiten des Moduls m durch einen kleineren Vertreter aus $\mathbf{Z}_m$ ersetzen.

Übung: Berechne analog $324^{37} \mod 11$.

3.2. Kryptographische Protokolle: RSA

Mit dem **RSA**-Verfahren – benannt nach seinen Erfindern **R**ivest, **S**hamir und **A**dleman aus dem Jahr 1977 – kann ein Sender S an Empfänger E eine verschlüsselte Nachricht übermitteln, die niemand ausser E lesen kann. Das Besondere: S und E müssen sich nie treffen, um einen geheimen Schlüssel zu vereinbaren.

Das Prinzip:



Der Empfänger E generiert zunächst den Public Key (n, k_{pub}), den er als öffentlichen Schlüssel jedem zur Verfügung stellt, der ihm eine Nachricht schicken will. Gleichzeitig generiert er den Private Key k_{priv} (den geheimen Schlüssel). Der Sender S holt sich den Public Key, kodiert damit seine Nachricht zum Geheimcode s, den er an den Empfänger E schickt. E kann mit seinem Private Key k_{pirv} den Geheimcode entschlüsseln, d.h. er erhält damit aus s wieder die Nachricht m. (O.B.d.A nehmen wir hier die Nachricht m als *ganze Zahl* an, etwaige Texte können ja über den ASCII-Code in (möglicherweise lange) ganze Zahlen kodiert werden.)

Das RSA-Verfahren nennt man **asymmetrisch**, weil die Schlüssel zum Kodieren (n,k_{pub}) und zum Dekodieren (n,k_{priv}) verschieden sind. Weder der Sender S noch irgend jemand sonst ausser E kann aus den öffentlich sichtbaren Daten (s,n,k_{pub}) wieder auf die Nachricht m zurückrechnen!

Wie funktioniert nun der Algorithmus von RSA im Innern? – Wir haben (fast) alle Bausteine zusammen, um das RSA-Verfahren zu verstehen. Die Idee ist, dass der Empfänger zu einer Zahl n eine Zahl $X = k_{pub} \cdot k_{priv}$ findet, die als Potenz auf die Nachricht m angewendet wieder die Nachricht m (mod n) ergibt. Dann ist `encode` und `decode` nahezu trivial:

Verschlüsseln (`encode`):	$s = m^{k_{pub}} \bmod n$
Gesendete Nachricht	s
Entschlüsseln (`decode`):	$m = s^{k_{priv}} \bmod n = m^{X} \bmod n$

Damit das funktioniert, muss lediglich $m \in \mathbf{Z}_n = \{0,1,...,n-1\}$ gelten. Ist m zu groß, so zerlegt man die Nachricht in kleinere "Pakete".

Wie findet der Empfänger E nun die Zahlen n und $X = k_{pub} \cdot k_{priv}$? Er geht nach folgendem Rezept vor (`genKey`):

1. Bestimme zwei Primzahlen p,q und ihr Produkt n=p·q.
2. Bestimme g = (p-1)(q-1).
3. Suche eine nicht zu kleine Zahl $k_{pub} \leq g$, die teilerfremd zu g ist. Dies gelingt mit Satz S1-8 (Algorithmus Erweiterter Euklid), den man mit g und mit verschiedenen Zahlen k_{pub} testet, bis man ein $ggT(k_{pub},g)=1$ findet. Als wichtiges Nebenprodukt ergibt sich nach Folgerung S1-9 die Zahl $x=k_{priv}$, die die Gleichung $k_{pub} \cdot x = 1 \pmod g$ erfüllt.

Zu 3.: Wir schreiben die Folgerung S1-9 nochmal so auf, wie wir sie hier brauchen

Folgerung S1-9: Wenn $k_{pub}, g \in \mathbb{N}$ mit $k_{pub} \leq g$ zwei **teilerfremde** Zahlen sind, dann läßt sich für die Gleichung

$$k_{pub} \cdot k_{priv} = 1 \pmod g$$

die Lösung k_{priv} aus dem Algorithmus ErweiterterEuklid(g, k_{pub}) gewinnen, der ein k_{priv} für die Gleichung

$$k_{pub} \cdot k_{priv} + r' \cdot g = 1$$

bestimmt (mit irgendeinem $r' \in \mathbf{Z}$).

Man kann nun zeigen, dass mit den so bestimmten Zahlen n, k_{pub}, k_{priv}, für alle $m \in \mathbb{Z}_n$ die Beziehung $m = m^{k_{pub}k_{priv}} \bmod n$ gilt (Korrektheit von RSA, siehe Vorlesung).

Beispiel-RSA für sehr kleine Zahlen durchrechnen!

Wie könnte ein Angreifer RSA knacken? – Gefährlich wäre es, wenn ein Angreifer die Zahl g = (p-1)(q-1) in die Hand bekäme, denn dann könnte er mit k_{pub}, g und Algo Erweiterter Euklid den geheimen Schlüssel k_{priv} leicht errechnen. Aber er kennt g nicht, er kennt nur n. Um g zu errechnen, braucht er aber p und q. **Die kryptographische Stärke von RSA liegt darin, dass es einem Angreifer für große n nicht gelingt, die Primzahlzerlegung n = p·q zu finden.** Das ist zwar für kleine n leicht möglich (man verwende z.B. `ifactor(n);` in Maple), aber die Berechnungszeit wächst für größere n exponentiell an. Vermutlich ist die Primzahlzerlegung ein NP-schwieriges Problem (das ist aber bisher nicht bewiesen). Damit RSA nicht

"geknackt" werden kann, müssen n, p und q große Zahlen sein (RSA Inc. empfiehlt 1024 Bits für p und q, also etwa 300 Dezimalstellen).

Beispiele:

- Wenn n die Größenordnung 10^{130} hat (d.h. etwa 450 Bits lang ist), dann braucht ein 100MHz-Pentium PC etwa 50 Jahre, um n zu faktorisieren. Hundert Millionen PCs parallel würden die Aufgabe in etwa 15 Sekunden schaffen.
- Wenn n die Größenordnung 10^{308} hat (d.h. etwa 1024 Bits lang ist), dann brauchen hundert Millionen PCs parallel 1000 Jahre zum Faktorisieren.

Die andere Möglichkeit RSA zu "knacken" liegt darin, bei bekanntem s, k_{pub} und n eine Zahl m zu "erraten", die $m^{k_{pub}} = s \pmod n$ erfüllt. Dieses Problem nennt man auch den *diskreten Logarithmus*, und dessen Berechnung ist für großes k_{pub} ebenfalls schwierig. Folgerung: Auch der öffentliche Schlüssel k_{pub} sollte eine große Zahl sein (ebenfalls einige 100 Dezimalstellen).

Darin liegt auch gleichzeitig ein Problem von RSA: Weil man mit sehr großen Zahlen arbeiten muss, sind Verschlüsselung und Entschlüsselung aufwendige Verfahren (s. Kap. 2), besonders wenn lange Nachrichten m in vielen "Portionen" übermittelt werden müssen.

Abhilfe: Es gibt performantere Verfahren (z.B. **DES** = **D**ata **E**ncryption **S**tandard), die aber zuvor das Einigen auf einen gemeinsamen Schlüssel notwendig machen (**symmetrisches** Verfahren, d.h. Dekodier-Schlüssel = Kodier-Schlüssel). Dieser DES-Schlüssel wird zu Beginn sicher mit RSA vom Sender zum Empfänger geschickt, danach folgt die eigentliche Nachricht DES-verschlüsselt. Dies ist u.a. Grundlage der Software **PGP** (**P**retty **G**ood **P**rivacy).

3.3. Hashfunktionen und Digitale Signatur

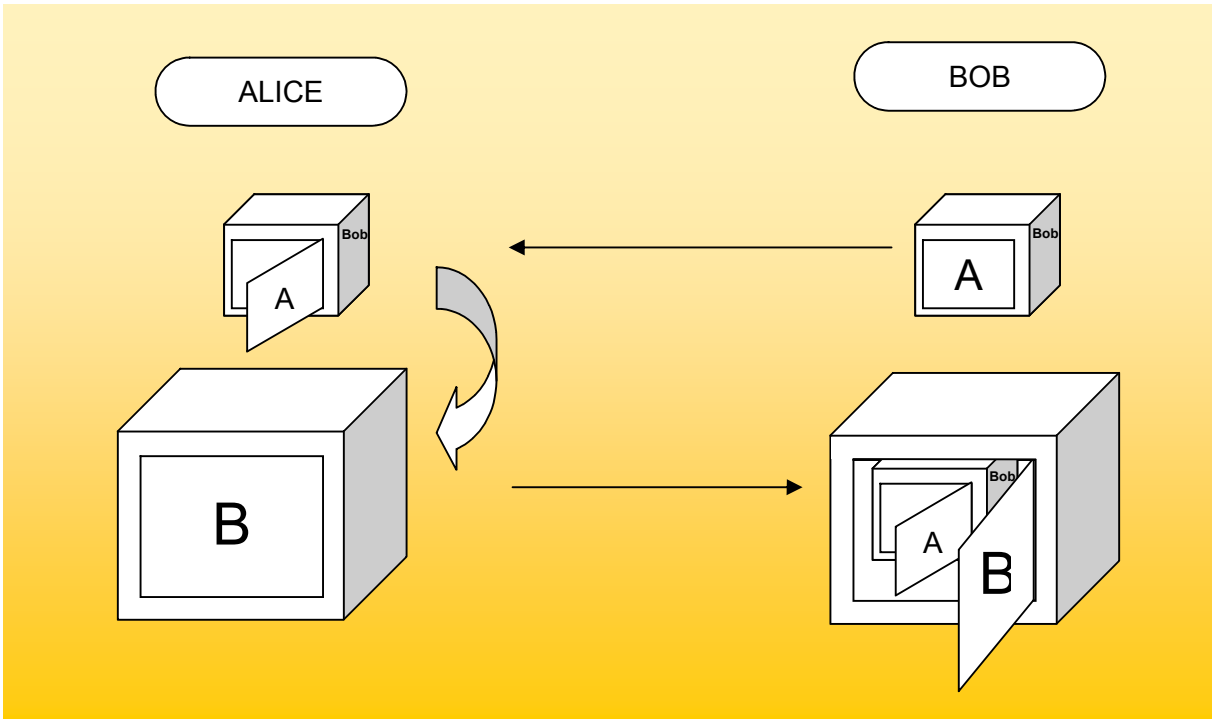
[Teschl05, Bd. 1, S. 74ff]

[mehr im crypTool-Skript]

Aktivierung: Wie kann ich bei einer Nachricht überprüfen, ob sie wirklich vom Sender kommt? (Früher gab es das Siegel, aber das war auch nicht 100% sicher und es paßt so schlecht durch die Datenleitungen im Internet)

Aktivierung 2: ALICE will BOB eine geheime Nachricht senden und BOB will wirklich sicher sein, dass die Nachricht von ALICE kommt und nicht von einem Dritten. Wie gehen sie vor?

Idee: Ein Tresor in einem Tresor (Erläuterung in Vorlesung)



Lösung mit RSA:

Um nicht Tresore durchs Internet schicken zu müssen, gehen ALICE und BOB wie folgt vor:

1. ALICE verschlüsselt ihre Nachricht m mit S_A, ihrem privaten Schlüssel (stellt Autorschaft sicher). Dann verschlüsselt ALICE diese Kryptonachricht mit K_B, dem öffentlichen Schlüssel von Bob (stellt Geheimhaltung sicher):

$$s = \left(m^{S_A} \bmod n_A\right)^{K_B} \bmod n_B$$

2. ALICE sendet s an BOB

3. BOB dechiffriert nun zunächst die Kryptonachricht s mit seinem privaten Schlüssel S_B – nur er kann das machen – und erhält eine dechiffrierte Nachricht d

$$d = s^{S_B} \bmod n_B = \left(m^{S_A} \bmod n_A\right)^{K_B \cdot S_B} \bmod n_B = m^{S_A} \bmod n_A$$

4. Diese Nachricht d könnte zwar jeder entschlüsseln (mittels ALICEs öffentlichem Schlüssel), aber d war niemals "ungeschützt" draußen im Netzwerk zu sehen. BOB entschlüsselt d nun mit ALICEs öffentlichem Schlüssel:

$$m^{S_A \cdot K_A} \bmod n_A = m$$

und erhält die Nachricht m. Was hier als Nachricht m herauskommt, kann nur von ALICE geschrieben worden sein.

Problem: die ganze Nachricht **s** verschlüsseln mit RSA ist sehr aufwendig. Gesucht: Funktion H, die aus Nachricht s einen kürzeren „Fingerabdruck“ H(s) herstellt.

Def D 3-1 (Hashfunktion):

Eine Hashfunktion H bildet eine Nachricht s beliebiger Länge auf eine Zahl oder Zeichenfolge aus einem fest definierten Wertebereich W = {0,1,…,N-1}, den **Hashwert** H(s), der eine feste Länge nicht überschreitet, ab.

(„hash“ (engl.) = Hackfleisch, ‚Haschee’, die Nachricht wird ‚zerhackt’, ein kleines Stück vom ‚Gehackten’, der Fingerabdruck, steht stellvertretend für die Nachricht.)

Einfache Beispiele für Hashfunktionen sind:

a. Nehme den Zahlwert des 1. Buchstabens der Nachricht als Hashwert: A=0, B=1, …, Z=25=N-1. (Analogie: Karteikästen in der Bibliothek)

b. Verwandle die Nachricht in eine Zahl Z_S und bilde $H(s) = Z_S \bmod N$.

Natürlich kann und wird es bei Hashfunktionen zu sog. **Kollisionen** kommen, das sind Nachrichten s_1 und s_2, die den gleichen Hashwert haben:

$$H(s_1) = H(s_2)$$

Bei der Datenspeicherung ist eine Kollision nicht sehr tragisch, man speichert einfach nicht in der Datenzelle k sondern in der Datenzelle k+N ab. Das bringt immer noch den Vorteil, beim Lookup nicht alle Daten, sondern nur 1/N der Daten anschauen zu müssen.

Übung: Bilden Sie die Menge der Städte {Aachen, Bonn, Bochum, Dortmund, Berlin} mit der Hashfunktion a. in eine Hashtabelle ab und vollziehen Sie die Schritte nach, die notwendig sind, um die Stadt Bochum wiederzufinden.

Aktivierung: Welche Anforderungen muss man an die Hashfunktion H in der Informatik allgemein stellen?

Aktivierung: Welche Anforderungen muss man an die Hashfunktion H *in der Kryptographie* stellen?

Aktivierung 2: Wieso sind die obigen Beispiele für Hashfunktionen keine guten kryptographische Hashfunktionen?

Wenn Zeit: Wdh. Prüfziffern:

ISBN-Beispiel bringen