29 Kryptografie: asymmetrische Verfahren

(Zum Teil nach Ideen aus den Beiträgen von Witten/Schulz: „RSA & Co in der Schule“ in der Zeitschrift LOGIN 2008).

Nachdem im letzten Kapitel die Verschlüsselung behandelt worden ist, wenden wir uns jetzt dem anderen wesentlichen kryptologischen Problem zu: der Authentifizierung.

„Habe ich die Nachricht wirklich von dem angeblichen Sender bekommen?“

Die Idee bei der Authentifizierung mit asymmetrischen Verfahren ist, die Kommunikationsrichtung umzukehren und das „Geheimnis“ zu teilen:

- *Nicht ich* versende verschlüsselte Nachrichten, sondern ich lasse mir verschlüsselte Nachrichten zuschicken.
- Dazu gebe ich dem Absender die zur Verschlüsselung benötigte Information.
- Wenn ich die Nachricht entschlüsseln kann, weiß ich, dass der Absender sie mit meinem Schlüssel verschlüsselt hat und dass sie zwischenzeitlich nicht geändert wurde, denn dazu müsste sie entschlüsselt und neu verschlüsselt worden sein..
- Das funktioniert nur, wenn man aus dem Verschlüsselungsverfahren **nicht** auf die Entschlüsselung kommen kann. Das ist die schon oft erwähnt „Asymmetrie“.

Wir machen zunächst einen Ausflug in die höhere Mathematik, um dann das RSA – Verfahren erklären zu können. Es geht also letztlich um mehr als „nur“ Verschlüsselung.

29.1 Addition in Restklassen

Das Caesar-Verfahren, zahlentheoretisch betrachtet:
Wir haben es im einfachen Fall mit 26 Buchstaben zu tun

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	w	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
									+5															+5	

Wenn wir anstelle der Buchstaben die entsprechenden Zahlen verwenden, ist es eine Addition (hier mit 5) in einer endlichen Menge von Zahlen Z_{26} = { 1, 2, 3, … 26 }. In der Tabelle ist dargestellt: 8 + 5 = 13, aber auch 23 + 5 = ***2***, was nicht unseren Vorstellungen von der Addition entspricht.
(Für Mathematiker: Wir rechnen in der Restklasse modulo 26)

Weitere Eigenschaften dieser seltsamen Addition: (nachzuvollziehen in der Tabelle)
Für jede Zahl a aus Z_{26} gilt: a + 26 = a und 26 + a = a; die Zahl 26 hat also hier die Eigenschaft, die wir von der Zahl 0 kennen (neutrales Element der Addition).
Bei der Entschlüsselung können wir, statt 5 zu subtrahieren, auch 21 addieren (13 + 21 = ***8*** und 5 + 21 = 26 = ***0***).
Es gibt noch 25 weitere Zahlenpaare mit dieser Eigenschaft (Summe 26; man nennt sie zueinander invers).
Weil sich auch das Kommutativgesetz und das Assoziativgesetz von der gewöhnlichen Addition vererben, spricht man in der Mathematik von einer additiven Gruppe.
Um einfacher rechnen zu können (ohne die Tabelle), stellen wir uns vor, dass es sich bei den 26 Zahlen um die Reste bei der (ganzzahligen) Division durch 26 handelt.
Man teilt alle ganzen Zahlen durch 26 und sortiert sie nach den auftretenden Resten:

0 = { … -26, 0, 26, 52, 78, … } k * 26
1 = { …-25, 1, 27, 53, … } k * 26 + 1
…
5 = { … -21, 5, 31, 57, … } k * 26 + 5
…
8 = { …-18, 8, 34, 60, … } k * 26 + 8
…
25 = { …-1, 25, 51, 77, …} k * 26 + 25

Wenn man nun eine Zahl aus ***5*** und eine Zahl aus ***8*** addiert, hat die Summe wieder einen Rest beim Teilen durch 26.
Die Rechnung $k_1 * 26 + 5 + k_2 * 26 + 8 = k_1 * 26 + k_2 * 26 + 5 + 8 = (k_1 + k_2) * 26 + 13$ birgt keine Überraschung und liefert eine Zahl aus ***13*** . Also gilt ***5 + 8 = 13***
Was ist nun ***23 + 5*** ?
Rechnung $k_1 * 26 + 23 + k_2 * 26 + 5 = k_1 * 26 + k_2 * 26 + 23 + 5 = (k_1 + k_2) * 26 + 28$
Die Zahl 28 muss noch zerlegt werden als 26 + 2, also … = $(k_1 + k_2 + 1) * 26 + 2$
Damit ***23 + 5 = 2***

Formel für die Verschlüsselung: $c = m + z \pmod p$
(Dabei ist m die Nachricht (Zahl zu dem Buchstaben), z die Schlüsselzahl und c die verschlüsselte Nachricht (Chiffre))

Zur Entschlüsselung brauchen wir die Gegenzahl (additiv Inverse), also eine Zahl z' mit

$$c + z' = m + z + z' = m * (z + z') = m + 0 = m \pmod p$$

Eine solche Zahl kann man sofort angeben: Um bei der oben betrachteten Verschiebung um 5 vom Buchstaben M zum Buchstaben H zurück zu kommen, müssen wir 21 addieren, also die Klasse zu 26 – 5.
Allgemein: die Restklasse zu: Modul – Verschlüsselungszahl.

Das Verfahren von Caesar ist **nicht geeignet**, weil die additiv Inverse der Verschlüsselungszahl leicht berechnet werden kann.

Wenn es mit der Addition nicht geht, dann vielleicht mit der Multiplikation.

Aufgaben
Addition in Restklassen üben, Gegenzahlen bestimmen.

29.2 modulares Multiplizieren

Als nächst-„schwierigere“ Rechenart steht die Multiplikation an:

Verschlüsselt wird eine Nachricht m mit dem Schlüssel z durch die Rechnung: $c = m * z \pmod p$

Für die Entschlüsselungszahl z’ muss dann gelten:

$$c * z' = m * z * z' = m * (z * z') = m * 1 = m \pmod p$$

Es wird also die Inverse bezüglich der Multiplikation gesucht: $z * z' \equiv 1 \pmod p$

Multiplikation modulo 26

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
2	2	4	6	8	10	12	14	16	18	20	22	24	0	2	4	6	8	10	12	14	16	18	20	22	24
3	3	6	9	12	15	18	21	24	1	4	7	10	13	16	19	22	25	2	5	8	11	14	17	20	23
4	4	8	12	16	20	24	2	6	10	14	18	22	0	4	8	12	16	20	24	2	6	10	14	18	22
5	5	10	15	20	25	4	9	14	19	24	3	8	13	18	23	2	7	12	17	22	1	6	11	16	21
6	6	12	18	24	4	10	16	22	2	8	14	20	0	6	12	18	24	4	10	16	22	2	8	14	20
7	7	14	21	2	9	16	23	4	11	18	25	6	13	20	1	8	15	22	3	10	17	24	5	12	19
8	8	16	24	6	14	22	4	12	20	2	10	18	0	8	16	24	6	14	22	4	12	20	2	10	18
9	9	18	1	10	19	2	11	20	3	12	21	4	13	22	5	14	23	6	15	24	7	16	25	8	17
10	10	20	4	14	24	8	18	2	12	22	6	16	0	10	20	4	14	24	8	18	2	12	22	6	16
11	11	22	7	18	3	14	25	10	21	6	17	2	13	24	9	20	5	16	1	12	23	8	19	4	15
12	12	24	10	22	8	20	6	18	4	16	2	14	0	12	24	10	22	8	20	6	18	4	16	2	14
13	13	0	13	0	13	0	13	0	13	0	13	0	13	0	13	0	13	0	13	0	13	0	13	0	13
14	14	2	16	4	18	6	20	8	22	10	24	12	0	14	2	16	4	18	6	20	8	22	10	24	12
15	15	4	19	8	23	12	1	16	5	20	9	24	13	2	17	6	21	10	25	14	3	18	7	22	11
16	16	6	22	12	2	18	8	24	14	4	20	10	0	16	6	22	12	2	18	8	24	14	4	20	10
17	17	8	25	16	7	24	15	6	23	14	5	22	13	4	21	12	3	20	11	2	19	10	1	18	9
18	18	10	2	20	12	4	22	14	6	24	16	8	0	18	10	2	20	12	4	22	14	6	24	16	8
19	19	12	5	24	17	10	3	22	15	8	1	20	13	6	25	18	11	4	23	16	9	2	21	14	7
20	20	14	8	2	22	16	10	4	24	18	12	6	0	20	14	8	2	22	16	10	4	24	18	12	6
21	21	16	11	6	1	22	17	12	7	2	23	18	13	8	3	24	19	14	9	4	25	20	15	10	5
22	22	18	14	10	6	2	24	20	16	12	8	4	0	22	18	14	10	6	2	24	20	16	12	8	4
23	23	20	17	14	11	8	5	2	25	22	19	16	13	10	7	4	1	24	21	18	15	12	9	6	3
24	24	22	20	18	16	14	12	10	8	6	4	2	0	24	22	20	18	16	14	12	10	8	6	4	2
25	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Die Tabelle zeigt, dass es zu allen **geraden** Zahlen keine multiplikativ **Inverse** gibt; in den jeweiligen Zeilen kommt die 1 als Ergebnis nicht vor.
Eine Entschlüsselung ist damit *nicht* möglich.
Anders sieht das zum Beispiel bei 23 aus (Primzahl).

Multiplikation modulo 23

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**Inverse**
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	**1**
2	2	4	6	8	10	12	14	16	18	20	22	1	3	5	7	9	11	13	15	17	19	21	**12**
3	3	6	9	12	15	18	21	1	4	7	10	13	16	19	22	2	5	8	11	14	17	20	**8**
4	4	8	12	16	20	1	5	9	13	17	21	2	6	10	14	18	22	3	7	11	15	19	**6**
5	5	10	15	20	2	7	12	17	22	4	9	14	19	1	6	11	16	21	3	8	13	18	**14**
6	6	12	18	1	7	13	19	2	8	14	20	3	9	15	21	4	10	16	22	5	11	17	**4**
7	7	14	21	5	12	19	3	10	17	1	8	15	22	6	13	20	4	11	18	2	9	16	**10**
8	8	16	1	9	17	2	10	18	3	11	19	4	12	20	5	13	21	6	14	22	7	15	**3**
9	9	18	4	13	22	8	17	3	12	21	7	16	2	11	20	6	15	1	10	19	5	14	**18**
10	10	20	7	17	4	14	1	11	21	8	18	5	15	2	12	22	9	19	6	16	3	13	**7**
11	11	22	10	21	9	20	8	19	7	18	6	17	5	16	4	15	3	14	2	13	1	12	**21**
12	12	1	13	2	14	3	15	4	16	5	17	6	18	7	19	8	20	9	21	10	22	11	**2**
13	13	3	16	6	19	9	22	12	2	15	5	18	8	21	11	1	14	4	17	7	20	10	**16**
14	14	5	19	10	1	15	6	20	11	2	16	7	21	12	3	17	8	22	13	4	18	9	**5**
15	15	7	22	14	6	21	13	5	20	12	4	19	11	3	18	10	2	17	9	1	16	8	**20**
16	16	9	2	18	11	4	20	13	6	22	15	8	1	17	10	3	19	12	5	21	14	7	**13**
17	17	11	5	22	16	10	4	21	15	9	3	20	14	8	2	19	13	7	1	18	12	6	**19**
18	18	13	8	3	21	16	11	6	1	19	14	9	4	22	17	12	7	2	20	15	10	5	**9**
19	19	15	11	7	3	22	18	14	10	6	2	21	17	13	9	5	1	20	16	12	8	4	**17**
20	20	17	14	11	8	5	2	22	19	16	13	10	7	4	1	21	18	15	12	9	6	3	**15**
21	21	19	17	15	13	11	9	7	5	3	1	22	20	18	16	14	12	10	8	6	4	2	**11**
22	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	**22**

In jeder Zeile kommt die 1 als Ergebnis vor, also gibt es zu jeder der 22 Zahlen eine Inverse bezüglich der Multiplikation modulo 23.

Diese beiden Beispiele lassen sich verallgemeinern: Durchgängig Inverse gibt es im Bereich von 1 bis zum ersten Teiler. Damit die Zahlen nicht unnötig groß werden, kann man gleich Primzahlen als Modul verwenden.

Die Inverse lässt sich immer mit dem „erweiterten Euklidischen Algorithmus“ berechnen.

Zum Beispiel: 15
Dass der größte gemeinsame Teiler 1 ist , wird nicht verwundern – schließlich ist 23 eine Primzahl.
Bei der Berechnung schreiben wir aber jetzt mit, wie oft man 15 und 23 verwendet hat.

m	n	Berechnung	
23	15	r = 23 – **1*** 15 = 8	8 = 23 - 15
15	8	r = 15 – **1***8 = 7	7 =15 – 1 (23 – 15) = 2 * 15 - 23
8	7	r = 8 – 7 = 1	1 = 8 – 7 = (23 – 15) – (2*15 – 23) = 2 * 23 – 3 * 15

Aus der letzten Zeile 1 = 2 * 23 – 3 * 15 entnimmt man, dass (-3) * 15 ≡1 (mod 23) ist.
-3 ≡ 20 (mod 23), denn -3 + 23 = 20, also ist 20 die gesuchte Inverse.

Das Programm pMultiplikation im Verzeichnis „07 Texte verschlüsseln Multiplikation“ zeigt:

- Man kann Texte damit verschlüsseln und entschlüsseln.
- Es ist ein **symmetrisches** Verfahren; aus dem Schlüssel lässt sich der Umkehrschlüssel berechnen.
- Es ist im Prinzip ein weiteres Substitutionsverfahren – mit der Möglichkeit statistischer Auswertungen zur Bestimmung des Schlüssels.
- Man kann es dem Angreifer schwerer machen und größere Zahlen verwenden; bei dem vorliegenden Programm ist dies begrenzt durch den Zahlbereich von Integer (32 Bit; bis 2147483647). Durch die

Verwendung größerer Zahlen, für die man alle Rechen- und Vergleichsoperationen neu entwerfen und implementieren müsste, lässt sich die Sicherheit vergrößern.

- Wenn man schon mit großen Modulen rechnen, kann man als Ausgangsdaten auch große Zahlen verwenden und zum Beispiel mehrere Buchstaben in eine Zahl packen. Im Programm wird zu zwei Buchstaben durch 26 * Nummer des ersten Buchstabens + Nummer des zweiten Buchstabens eine Zahl ermittelt und diese verschlüsselt. Nach der Entschlüsselung erhält man mittels „div 26“ und „mod 26“ wieder die ursprünglichen Nummern und damit die Ausgangsbuchstaben.

Weil es auch wieder ein symmetrisches Verfahren ist und keine Vorteile gegenüber anderen symmetrischen Verfahren besitzt, wird es nicht verwendet.

Schade!

Aufgaben:

1) Multiplikation in Restklassen üben.
2) Berechnung der Inversen mit dem erweiterten Euklidischen Algorithmus.
3) Programm „Texte verschlüsseln Multiplikation“ erforschen und erklären.

29.3 modulares Potenzieren, erster Versuch

Wir verschaffen uns eine (möglichst sehr große) Primzahl p und dann:

Verschlüsselt wird eine Nachricht m mit dem Schlüssel e durch die Rechnung: $c = m^e \pmod p$, wobei m<p, e<p und c < p.

Wie kann man eine solche Botschaft wieder entschlüsseln?
Von r^2 zu r kommt man durch – Wurzelziehen!
Bei festem Exponent e muss man also aus c die e-te Wurzel modulo p ziehen, um m zu erhalten.
Wenn andererseits die Basis fest bleibt und der Exponent variabel ist, brauchte man so eine Art Logarithmus (wird in anderen Verfahren tatsächlich benutzt):

Weil wir bei den bisherigen Verfahren eine weitere Zahl addiert bzw. mit einer solchen Zahl multipliziert haben, um die Entschlüsselung vorzunehmen, bietet es sich an, es hier mit einem **weiteren Potenzieren** zu versuchen.

Also suchen wir d mit $c^d = (m^e)^d = m \pmod p$
Wenn m und p keine gemeinsamen Teiler haben, reduziert sich die Gleichung zu $m^{e\,d\,-\,1} = 1 \pmod p$

Es geht also um Zahlen mit $m^t = 1 \pmod p$

Schauen wir uns die Situation bei kleinen Primzahlen an:

p = 11 **Potenzieren modulo 11**

Exponent >>	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
1	1	1	1	1	1	1	1	1	1	1	1
2	2	4	8	5	10	9	7	3	6	1	2
3	3	9	5	4	1	3	9	5	4	1	3
4	4	5	9	3	1	4	5	9	3	1	4
5	5	3	4	9	1	5	3	4	9	1	5
6	6	3	7	9	10	5	8	4	2	1	6
7	7	5	2	3	10	4	6	9	8	1	7
8	8	9	6	4	10	3	2	5	7	1	8
9	9	4	3	5	1	9	4	3	5	1	9
10	10	1	10	1	10	1	10	1	10	1	10

Rechnung exemplarisch für die Zeile **5**: 5; dann 5*5 = 25 = 2*11 + **3**; 3*5 = 15 = 11 + **4**; 4 * 5 = 20 = 11 + **9**, und weiter mit 5 multiplizieren (modulo 11)

Verschlüsselung ist möglich mit den **Exponenten** (Spalten) 3, 7 und 9; es sind gerade die Zahlen, die **mit p – 1 = 10 teilerfremd** sind. Bei den anderen Exponenten erhält man für unterschiedliche Werte für m dasselbe Ergebnis, so dass eine Entschlüsselung nicht möglich ist.
Für e = 7 und m = 5 liefert die Tabelle den Wert 3 für c. Womit müssen wir potenzieren (mod 11), um wieder 5 zu erhalten? Die Tabelle zeigt uns die Lösungen 3 und 8.
Für e = 7 und m = 8 liefert die Tabelle den Wert 2 für c. Womit müssen wir potenzieren (mod 11), um wieder 8 zu erhalten? Die Tabelle zeigt uns diesmal nur eine Lösung: 3.

p = 13 **Potenzieren modulo 13**

Exponent >>	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	***12***	**13**
1	**1**	1	1	1	1	1	1	1	1	1	1	1	1
2	**2**	4	8	3	6	12	11	9	5	10	7	1	2
3	**3**	9	1	3	9	1	3	9	1	3	9	1	3
4	**4**	3	12	9	10	1	4	3	12	9	10	1	4
5	**5**	12	8	1	5	12	8	1	5	12	8	1	5
6	**6**	10	8	9	2	12	7	3	5	4	11	1	6
7	**7**	10	5	9	11	12	6	3	8	4	2	1	7
8	**8**	12	5	1	8	12	5	1	8	12	5	1	8
9	**9**	3	1	9	3	1	9	3	1	9	3	1	9
10	**10**	9	12	3	4	1	10	9	12	3	4	1	10
11	**11**	4	5	3	7	12	2	9	8	10	6	1	11
12	**12**	1	12	1	12	1	12	1	12	1	12	1	12

Beobachtung in beiden Beispielen: in der vorletzten Spalte (p-1) stehen lauter Einsen.
Dies lässt sich auch für weitere Primzahlen bestätigen und sogar allgemein beweisen:

Der kleine Satz von Fermat
Für eine Primzahl p und alle zu p teilerfremden ganzen Zahlen a gilt : $a^{p-1} \equiv 1 \pmod p$

Den Beweis dazu findet man an verschiedenen Stellen; er soll hier nicht thematisiert werden.

Wir suchen also einen Entschlüsselungsexponenten d mit $m^{e*d} \equiv m \pmod p$

Weitere Beobachtung: Es gibt in den obigen Wertetabellen Zeilen mit mehreren Einsen; die vorkommenden Werte werden dann zyklisch mehrfach durchlaufen. Die Zykluslänge ist aber in jedem Fall ein Teiler von p-1, oft p-1 selbst, wenn es nur eine Eins gibt.

Mit m^e befinden wir uns in der Zeile m; m^e weiter mit m^e multiplizieren (mod p) bedeutet, dass wir uns in dem dortigen Zyklus bewegen.

Beispiel: m=11 und e= 5; $11^5 \equiv 7$, $11^{5*2} = 7*7 \equiv 10$, $11^{5*3} = 7*7*7 \equiv 5$, $11^{5*4} = 7*7*7*7 \equiv 5*7 \equiv 9$, $11^{5*5} \equiv 9*7 \equiv 11$ – Entschlüsselungsexponent gefunden mit d = 5. Rechnungen alle mod 13.
Es ist e * d = 5 * 5 = 25 und damit e * d ≡ 1 mod **12**

Wenn wir dabei einmal in der Spalte mit dem Exponenten **1** landen, sind wir fertig, denn dann haben wir **m** als Ergebnis.
Für die Exponenten gilt dann

$$e * d \equiv 1 \pmod{(p-1)}$$

Anmerkungen:

- Eigentlich mod Zykluslänge, aber dann wäre der Entschlüsselungsexponent von m abhängig. Weil das nicht sinnvoll ist (beim Entschlüsseln kennen wir m ja gar nicht!), verwendet man das Vielfache p-1.
- Die Gleichung ist nur lösbar, wenn e und p-1 keinen gemeinsamen Teiler haben.

Damit haben wir das modulare Wurzelziehen auf die Bestimmung der Inversen bei der Multiplikation zurückgeführt – und dass diese Gleichung lösbar ist, wissen wir aus dem vorigen Kapitel.

Die gute Nachricht: man kann ver- und entschlüsseln. Es ist ein brauchbares Kryptosystem.

Die schlechte Nachricht: Wenn man e kennt, kann man d leicht berechnen. Es ist also wieder kein asymmetrisches Kryptosystem.

29.4 modulares Potenzieren, zweiter Versuch

Statt mit einer Primzahl als Modul – was die Überlegungen im vorigen Abschnitt vereinfacht hatte – starten wir nun mit dem Produkt zweier Primzahlen. Diese werden für die tatsächlichen sicherheitsrelevanten Anwendungen später sehr groß sein müssen, hier nehmen wir der Übersichtlichkeit halber kleine Primzahlen. Vollziehen sie die Überlegungen und Beobachtungen ruhig mit anderen kleinen Primzahlen nach !

Die bisherigen Überlegungen für die Ver- und Entschlüsselung bleiben gültig: Beides geschieht durch Potenzieren modulo n (hier n=p*q).

Auch die Überlegungen zur Bestimmung des Entschlüsselungsexponenten gelten weiterhin: $e * d \equiv 1 \pmod{\text{(Zykluslänge)}}$

Wertetabelle für das Potenzieren modulo 21 (3*7)

Exponent >>	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	4	8	16	11	1	2	4	8	16	11	1	2	4	8	16	11	1	2	4	8
3	3	9	6	18	12	15	3	9	6	18	12	15	3	9	6	18	12	15	3	9	6
4	4	16	1	4	16	1	4	16	1	4	16	1	4	16	1	4	16	1	4	16	1
5	5	4	20	16	17	1	5	4	20	16	17	1	5	4	20	16	17	1	5	4	20
6	6	15	6	15	6	15	6	15	6	15	6	15	6	15	6	15	6	15	6	15	6
7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
8	8	1	8	1	8	1	8	1	8	1	8	1	8	1	8	1	8	1	8	1	8
9	9	18	15	9	18	15	9	18	15	9	18	15	9	18	15	9	18	15	9	18	15
10	10	16	13	4	19	1	10	16	13	4	19	1	10	16	13	4	19	1	10	16	13
11	11	16	8	4	2	1	11	16	8	4	2	1	11	16	8	4	2	1	11	16	8
12	12	18	6	9	3	15	12	18	6	9	3	15	12	18	6	9	3	15	12	18	6
13	13	1	13	1	13	1	13	1	13	1	13	1	13	1	13	1	13	1	13	1	13
14	14	7	14	7	14	7	14	7	14	7	14	7	14	7	14	7	14	7	14	7	14
15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15
16	16	4	1	16	4	1	16	4	1	16	4	1	16	4	1	16	4	1	16	4	1
17	17	16	20	4	5	1	17	16	20	4	5	1	17	16	20	4	5	1	17	16	20
18	18	9	15	18	9	15	18	9	15	18	9	15	18	9	15	18	9	15	18	9	15
19	19	4	13	16	10	1	19	4	13	16	10	1	19	4	13	16	10	1	19	4	13
20	20	1	20	1	20	1	20	1	20	1	20	1	20	1	20	1	20	1	20	1	20

In dieser zugegebenermaßen sehr unübersichtlichen Tabelle gibt es Zeilen, die keine Eins enthalten; diese sind wegen der Zyklen, die zu m zurückführen sollen, nicht interessant.

Beschränken wir uns auf die Zeilen, in denen eine Eins vorkommt:

Wertetabelle für das Potenzieren modulo 21 (nur die Zeilen mit Einsen)

Exponent >>	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	4	8	16	11	1	2	4	8	16	11	1	2	4	8	16	11	1	2	4	8
4	4	16	1	4	16	1	4	16	1	4	16	1	4	16	1	4	16	1	4	16	1
5	5	4	20	16	17	1	5	4	20	16	17	1	5	4	20	16	17	1	5	4	20
8	8	1	8	1	8	1	8	1	8	1	8	1	8	1	8	1	8	1	8	1	8
10	10	16	13	4	19	1	10	16	13	4	19	1	10	16	13	4	19	1	10	16	13
11	11	16	8	4	2	1	11	16	8	4	2	1	11	16	8	4	2	1	11	16	8
13	13	1	13	1	13	1	13	1	13	1	13	1	13	1	13	1	13	1	13	1	13
16	16	4	1	16	4	1	16	4	1	16	4	1	16	4	1	16	4	1	16	4	1
17	17	16	20	4	5	1	17	16	20	4	5	1	17	16	20	4	5	1	17	16	20
19	19	4	13	16	10	1	19	4	13	16	10	1	19	4	13	16	10	1	19	4	13
20	20	1	20	1	20	1	20	1	20	1	20	1	20	1	20	1	20	1	20	1	20

Beobachtungen:

- Die Zeilennummern liefern genau die Zahlen, die zu 21 teilerfremd sind.
- Es sind gleichzeitig genau die Zahlen, die bez. 21 invertierbar sind.
- Die Zykluslänge ist hier 6, die kleinste Zahl s mit $m^s \equiv 1 \pmod n$ für alle m

Wie findet man (ohne Probieren und Wertetabelle) einen Wert s, so dass $m^s \equiv 1 \pmod n$?

Hier hilft der Satz von Fermat-Euler:
Wenn zwei ganze Zahlen m und n (>=2) teilerfremd sind, so gilt $m^{\varphi(n)} \equiv 1 \pmod n$.
Dabei liefert die Eulersche φ-Funktion die Anzahl der natürlichen Zahlen <= n, die zu n teilerfremd sind.

Für Mathematiker: **φ(n) ist die Mächtigkeit der multiplikativen Gruppe in der Restklasse mod n!**

Spezialfall: Wenn n das Produkt von zwei Primzahlen p und q ist, haben (nur) p, 2p, 3p, ..qp und q, 2q, 3q, …pq gemeinsame Teiler mit n. Wenn p und q verschieden sind, kommt in dieser Aufzählung außer pq (=qp) keine Zahl doppelt vor. Also gibt es p + q -1 nicht teilerfremde Zahlen unter den insgesamt n infrage kommenden Zahlen. Teilerfremd sind dann die restlichen Zahlen: n – (p + q – 1) = p*q –q*1 – p*1 + 1*1 = (p – 1)*(q – 1).

Also können wir (p-1)(q-1) als Wert für s benutzen; dies ist evtl. nicht optimal, funktioniert aber.
Im Beispiel oben ist (p-1)(q-1) = 2 * 6 = 12, die (kleinste) Zykluslänge ist 6.
Mögliche Erweiterung – je nach Lerngruppe: Über die Carmichael-Funktion zu einer optimierten RSA – Version. (s. Witten/Schulz: „RSA & Co in der Schule“, LOGIN 2008).

29.5 RSA als Kryptosystem

Man sucht zwei Primzahlen p und q und verwendet n = p*q als Modul.
Der Verschlüsselungsexponent e muss teilerfremd zu p-1 und q-1 sein.
Verschlüsselt wird durch Potenzieren modulo n.
Den Entschlüsselungsexponenten finden wir als Lösung von e * d ≡ 1 (mod ((p-1)*(q-1)) durch Anwenden des erweiterten Euklidischen Algorithmus.
Entschlüsselt wird dann wieder durch modulares Potenzieren mit d.

Aufgrund der bisherigen Beobachtungen und der genannten zahlentheoretischen Sätze wird dies gelingen.
Damit ist RSA ein funktionierendes Kryptosystem – wie auch die vorher betrachteten Systeme.

Wo ist den nun der Unterschied, wo ist die Asymmetrie ?

Bei kleinen Zahlen kann man ja alles noch durchprobieren, bei relevanten (=sicheren) Systemen verwendet man Primzahlen mit mehreren hundert Stellen. Die Multiplikation p*q = n ist eine einfache Rechnung.
Wie findet man aber p und q, wenn man n kennt? Man muss praktisch alle Zahlen bis zur kleineren der beiden (riesigen) Primzahlen durchprobieren, um p und damit auch q zu finden. Weil es keine weiteren Teiler gibt, kann man auch nicht im Laufe der Rechnung mit kleineren Zahlen weiter rechnen. In der Zeit (in Jahren !), die der schnellste Computer braucht, um diese Zerlegung durchzuführen, liegt die Sicherheit diese Verfahrens.

Asymmetrie:
Ich kann einem Kommunikationspartner n und e verraten (öffentlicher Teil des Schlüssels).
Damit kann dieser Nachrichten an mich verschlüsseln.
Er kann aber solche Nachrichten nicht entschlüsseln. Auch andere, die n und e kennen, können nur ver- aber nicht entschlüsseln.
Ich selbst kenne den geheimen Teil des Schlüssels, nämlich p und q, und kann den Entschlüsselungsexponenten berechnen und Nachrichten entschlüsseln.
Damit sind Verschlüsselung und Entschlüsselung getrennt.

Authentifizierung:
Wenn ich eine Nachricht bekomme und diese mit meinem Teil des Schlüssels entschlüsseln kann, weiß ich, dass sie von jemandem stammt, der meinen öffentlichen Schlüssel bekommen hat. Wen ich etwa jedem Bekannten einen anderes e – n – Paar schicke, kann ich sicher unterscheiden, von wem die Nachricht kam.

Beispiele / Aufgaben für die Verschlüsselung:

1) Im Projekt „RAS einfach“ können Sie Ver- und Entschlüsselung mit kleinen Primzahlen nachvollziehen. Der Wertebereich des Datentyps „Longint“ begrenzt die Größe der Zahlen.

2) Beachten Sie auch die Ausführungen unter www.vom-hau.de/Javascript oder im Verzeichnis „09 Javascript RSA“ zu RSA. Dort ist der Zahlbereich größer, der Entschlüsselungsexponent wird aber immer noch durch Probieren gefunden.

3) Zum Nachrechnen:
p = 7 und q = 11, also n = 77
Hilfreich ist hier wieder eine Excel-Tabelle, die aber so groß ist, dass sie hier nicht auf das Blatt passt. (Die zentrale kopierfähige Formel, mit der auch die bisher abgedruckten Tabellen entstanden sind, lautet: =REST((B2*$A2); 77))
Wenn man in dieser Tabelle die Zeilen mit den Vielfachen von 7 und den Vielfachen von 11 löscht (nicht teilerfremd zu 77), erkennt man Spalten mit lauter Einsen bei 30 und 60. 60 = (7-1)*(11-1) ist der nach dem Fermat-Euler-Satz erwartete Wert, 30 der optimale Wert für die Phasenlänge (nehmen wir hier).
Der Verschlüsselungsexponent muss dann zu 30 teilerfremd sein: e = 7 ?

a	b	Berechnung	
30	**7**	r = 30 - 7 = 23	23 = 30 - 7
23	7	r = 23 – **3***7 = 2	2 =(30 – 7) – 3*7 = 30 – 4*7
7	2	r = 7 – **3***2 = 1	1 = 7 – 3*(30 – 4*7) = -3 * 30 + 13 * 7

13 ist also die Inverse von 7 modulo 30.

Eine Zahl (< 77) zum Verschlüsseln: etwa m = 31; die Tabelle liefert für 31^7 mod 77 die Zahl c = 59
Entschlüsseln: 59^{13} mod 77 liefert 31.

4) Unter „08 RSA Python“ finden Sie ein Python-Script – getestet unter ActivePython 2.5 – das die Schritte von zwei Primzahlen bis zur Entschlüsselung nachvollzieht. Der Vorteil von Python liegt hier darin, das es mit beliebig großen ganzen Zahlen umgehen kann.

29.6 Schlüsseltausch nach Diffie-Hellmann

(Gehört logisch zu den symmetrischen Verfahren, ist aber erst mit Kenntnissen über das modulare Potenzieren verständlich)

Wie kann ich – bei einer symmetrischen Verschlüsselung – mit meinem Kommunikationspartner einen Schlüssel vereinbaren? Eine Email mit dem Schlüssel und danach die verschlüsselte Email zu schicken ist sicher keine gute Idee. Einen persönlichen Kontakt werde ich nicht immer vorher herstellen können – dann kann ich auch gleich mit meiner Überweisung zur Bank gehen … Telefonieren ist auch zu umständlich. Man müsste über das Internet (automatisch – auch ohne extra Eingriffe des Benutzers) Informationen zum Schlüssel austauschen können, ohne dass ein Dritter daraus den Schlüssel zusammensetzen bzw. berechnen kann.
Das von Hellmann, Diffie und Merkle entwickelte und 1976 veröffentlichte Verfahren bietet eine Möglichkeit, dieses Problem zu lösen.

Die Grundlage:

- Auch beim modularen Potenzieren gilt : $\left(Basis^{Exponent1}\right)^{Exponent2} = \left(Basis^{Exponent2}\right)^{Exponent1} \pmod p$
- Es gibt – außer aufwändigem Durchprobieren – keine Möglichkeit, vom Wert der Potenz auf den Exponenten zu schließen (Problem des diskreten Logarithmus).

Die Idee:

- Der Modul p (eine Primzahl) und die Basis g (eine Primitivwurzel von p) werden (öffentlich) ausgetauscht.
- Ich halte den zufällig erzeugten „Exponent1“ geheim, mein Kommunikationspartner erzeugt den „Exponent2“ und behält diesen für sich.
- Dafür schicke ich $Basis^{Exponent1}$ und erhalte vom Kommunikationspartner $Basis^{Exponent2}$.
- Jetzt können wir beide den Schlüssel s berechnen als $s = Basis^{Exponent1*Exponent2}$.

Ein einfaches Bεispiel:

In der Tabelle auf Seite 5 können wir die Werte nachschauen; für die Verschlüsselung mit festem Exponent waren die Spalten interessant, jetzt sind es die Zeilen: als g verwendbar sind Zahlen, in deren Zeile kein Wert doppelt vorkommt (und damit alle Zahlen < p vorkommen): 2, 6, 7 und 8. Man nennt diese Zahlen **Primitivwurzeln** der Primzahl p.

Alice	ein „Horcher“	Bob
Einer muss anfangen: p=11 g=6		
wählt geheimes a=3		
rechnet $A=6^3 \bmod 11$		
sendet A = 7	hört p=11, g=6, A=7	empfängt p, g und A
		wählt geheimes b=5
		rechnet $B=6^5 \bmod 11$
	hört B=10	sendet B = 10
empfängt B=10		
rechnet $S=B^a \bmod 11$ $= 10^3 \bmod 11 = 10$		rechnet $S=A^b \bmod 11$ $= 7^5 \bmod 11 = 10$
verwendet S=10 als Schlüssel	kennt also A und B;	verwendet S=10 als Schlüssel
	um S zu finden, muss er daraus a oder b berechnen, was bei den hier verwendeten kleinen Zahlen sehr leicht geht, bei großen Primzahlen aber unrealistisch lange dauert.	

Schwächen / Angriffspunkte des Verfahrens

- Bei zu kleinen Primzahlen kann der diskrete Logarithmus durch Probieren ermittelt werden.
- Ein „man in the middle“, der nicht nur horcht, sondern das Verfahren kennt **und** Datenpakete verändern kann, wird den Wert für A durch einen mit einem eigenen Exponenten a’ erzeugten Wert ersetzen, und ebenso B durch einen mit ihm bekannten b’ errechneten Wert ersetzen. Auf diese Weise entstehen zwei Schlüssel S_1 und S_2 und zwei Kommunikationen, zwischen denen der „man in the middle“ durch Umschlüsseln vermittelt. Dabei wird der gesamte Inhalt der verschlüsselten Kommunikation bekannt.
- Vermeiden kann man diesen Angriff, wenn die Nachrichten zusätzlich authentifiziert werden.

29.7 Aufgaben

Schreiben Sie eine Klasse „LangeZahlen“ (umfangreich, aber lehrreich; Methoden für die Grundrechenarten und „Potenzieren modulo“) oder verwenden Sie etwa „Python“ (andere Sprache, anderes Programmiersystem) und bauen Sie das Projekt „RSA-einfach“ nach, ergänzt um einen Programmteil zur Unterstützung von „Diffie-Hellmann“. Testen Sie die Kommunikation per Email mit einem Partner: Öffentliche Schlüssel austauschen, damit den „Diffie-Hellmann“ - Schlüsselaustausch verschlüsseln; den so vereinbarten Schlüssel für eine symmetrische Verschlüsselung benutzen. Welche Chancen hat jetzt noch ein „man in the middle“?