8.10 Das RSA-Verfahren

1977 fanden Ronald Rivest, Adi Shamir und Leonard Adleman eine solche Falltürfunktion. Das nach ihnen benannte **RSA-Verfahren** wird bis heute eingesetzt.

8.10.1 Vorbereitung: Rechnen mit Resten

a mod n ist der Rest, den a bei Division durch n lässt. Beispiele: 17 mod $5 = 2$, 24 mod $6 = 0$, -23 mod $75 = 52$ (denn $-23 = (-1) \cdot 75 + 52$).

Die **eulersche Phi-Funktion** $\varphi(n)$ gibt an, wie viele Zahlen t mit $1 \leq t < n$ teilerfremd zu n sind:

n	**teilerfremde** t	$\varphi(n)$
5	1, 2, 3, 4	4
6	1, 5	2
15	1, 2, 4, 7, 8, 11, 13, 14	8

Für RSA genügen zwei Regeln:

- Ist p eine Primzahl, so ist $\varphi(p) = p - 1$.
- Sind $p \neq q$ Primzahlen und $n = p \cdot q$, so ist $\varphi(n) = (p-1)(q-1)$.

Der **Satz von Euler** besagt: Sind m und n teilerfremd, so gilt $m^{\varphi(n)}$ mod $n = 1$. Daraus folgt die Idee von RSA: Man zerlegt den Exponenten $\varphi(n)$ geschickt in zwei Faktoren – einen zum Verschlüsseln, einen zum Entschlüsseln.

Arbeitsauftrag 9

35. Rechnen Sie: 24 mod 5, 232 mod 7, $3\,492\,392$ mod 2.
36. Bestimmen Sie $\varphi(11)$, $\varphi(35)$ und $\varphi(77)$. Prüfen Sie bei $\varphi(35)$ und $\varphi(77)$ die zweite Regel nach, indem Sie zusätzlich abzählen.

8.10.2 Schlüsselerzeugung

1. Zwei verschiedene Primzahlen p und q wählen.
2. $n = p \cdot q$ berechnen.
3. $\varphi(n) = (p-1)(q-1)$ berechnen.
4. Ein e mit $1 < e < \varphi(n)$ wählen, das teilerfremd zu $\varphi(n)$ ist.
5. Das d bestimmen mit $e \cdot d$ mod $\varphi(n) = 1$ (erweiterter euklidischer Algorithmus).

> LK
>
> Schritt 5 ist LK-Stoff: Der GK-Lehrplan verlangt das RSA-Verfahren ausdrücklich *„ohne die Berechnung des privaten Schlüssels"*, der LK dagegen fordert den euklidischen und den erweiterten euklidischen Algorithmus. Im Grundkurs bekommen Sie d also vorgegeben und prüfen nur die Bedingung $e \cdot d$ mod $\varphi(n) = 1$ nach — rechnen Sie Schritt 5 trotzdem einmal mit, er ist der schönste Teil.

Öffentlicher Schlüssel: (e, n). Privater Schlüssel: (d, n). Die Zahlen p, q und $\varphi(n)$ werden vernichtet – wer sie kennt, kann d berechnen.

Verschlüsseln: $c = m^e$ mod n Entschlüsseln: $m = c^d$ mod n

8.10.3 Durchgerechnetes Beispiel

Wir wählen $p = 7$ und $q = 13$.

1. $n = 7 \cdot 13 = 91$
2. $\varphi(n) = 6 \cdot 12 = 72$
3. Wahl von $e = 5$. Probe: ggT$(5, 72) = 1$, also teilerfremd.
4. Gesucht ist d mit $5d$ mod $72 = 1$. Euklidischer Algorithmus: $72 = 14 \cdot 5 + 2$ und $5 = 2 \cdot 2 + 1$, der ggT ist also 1.
5. Rückwärts eingesetzt: $1 = 5 - 2 \cdot 2 = 5 - 2 \cdot (72 - 14 \cdot 5) = 29 \cdot 5 - 2 \cdot 72$. Also ist $d = 29$. Probe: $5 \cdot 29 = 145 = 2 \cdot 72 + 1$.

Öffentlicher Schlüssel $(5, 91)$, privater Schlüssel $(29, 91)$.

Verschlüsselung der Nachricht $m = 3$:

$$c = 3^5 \text{ mod } 91 = 243 \text{ mod } 91 = 61$$

Entschlüsselung:

$$m = 61^{29} \text{ mod } 91 = 3$$

> In der Webfassung steht hier ein Bedienfeld zum Ausprobieren: **RSA-Rechner**.

Im Bedienfeld können Sie p und q verändern. Probieren Sie größere Primzahlen und beobachten Sie zweierlei: Die Rechnung bleibt gleich schnell, der Angriff durch Faktorisieren wird dagegen rasch mühsam.

Die zweite Rechnung führt man nicht mit dem Taschenrechner aus – 61^{29} hat 52 Stellen. Man rechnet **modular**, also nach jedem Schritt den Rest, oder nutzt die schnelle Exponentiation:

```
1 # Modulare Exponentiation - in Python eingebaut
2 print(pow(3, 5, 91))     # 61   Verschluesseln
3 print(pow(61, 29, 91))   # 3    Entschluesseln
```

8.10.4 Sicherheit von RSA

Die Sicherheit steht und fällt mit dem **Faktorisierungsproblem**: Wer n in p und q zerlegen kann, berechnet $\varphi(n)$ und daraus d – RSA ist gebrochen. Ein schnelles Faktorisierungsverfahren ist bis heute nicht bekannt, aber es ist auch nicht bewiesen, dass es keines gibt.

In der Praxis benutzt man deshalb sehr große Zahlen: Schlüssel unter 2048 Bit gelten als überholt, empfohlen werden heute 3072 Bit oder mehr. Unser $n = 91$ ist 7 Bit lang.

Ein Nachtrag, den ältere Darstellungen noch nicht enthalten: Ein hinreichend großer Quantencomputer könnte mit dem Algorithmus von Peter Shor (1994) in Polynomialzeit faktorisieren und damit RSA brechen. Solche Rechner existieren bisher nicht in nutzbarer Größe. Trotzdem werden seit 2024 schrittweise **post-quanten-sichere** Verfahren eingeführt, die auf anderen mathematischen Problemen beruhen; Messenger und Browser setzen sie bereits parallel zu RSA ein.

Arbeitsauftrag 10

37. Führen Sie die Schlüsselerzeugung für $p = 11$ und $q = 13$ mit $e = 7$ vollständig durch und bestimmen Sie d über den erweiterten euklidischen Algorithmus.
38. Verschlüsseln Sie mit Ihrem Schlüssel aus Aufgabe 37 die Zahl $m = 9$ und entschlüsseln Sie das Ergebnis wieder. Kontrollieren Sie mit `pow(...)`.
39. Erklären Sie, warum p und q nach der Schlüsselerzeugung gelöscht werden müssen.
40. Warum darf m nicht größer als n sein? Was folgt daraus für längere Nachrichten?
41. Schreiben Sie ein Programm, das zu gegebenen p, q und e den privaten Schlüssel d berechnet und einen Text zeichenweise ver- und entschlüsselt.