Serie 10.

Woche: **23.11 – 26.11.2026**

Heute beschäftigen wir uns mit dem RSA-Public-Key-Kryptosystem. RSA ist ein so genanntes asymmetrisches Verschlüsselungsverfahren, welches 1977 von **R**ivest, **S**hamir und **A**dleman am MIT entwickelt wurde. Man nutzt aus, dass es Funktionen gibt, bei denen eine Richtung leicht, aber die andere Richtung sehr schwer zu berechnen ist (sogenannte Einwegfunktionen). In diesem Fall beschreibt die Funktion die Faktorisierung einer grossen Zahl, also die Zerlegung dieser Zahl in ihre Primfaktoren. Man erzeugt die Zahlen e, d (Verschlüsselungs- und Entschlüsselungsexponent) und M (RSA-Modul). Dabei bildet das Paar (e, M) den öffentlichen Schlüssel (*public key*) und (d, M) den privaten Schlüssel (*private/secret key*). Dabei geht man wie folgt vor:

I. Generiere die Schlüssel.

1. Wähle drei beliebige Zahlen a, b und c.
2. Definiere p bzw. q als die auf a bzw. b folgenden Primzahlen.
3. Berechne den RSA-Modul $M = p \cdot q$.
4. Berechne die Eulersche φ-Funktion von M, wobei $\varphi(M) = (p-1) \cdot (q-1)$.
5. Sind $\varphi(M)$ und c nicht teilerfremd (also $\text{ggT}(c, \varphi(M)) \neq 1$), vergrössere c *solange* um 1, *bis* $\text{ggT}(c, \varphi(M)) = 1$. Setze dann den *public key* $e = c$.
6. Finde einen *private key* d. Dabei muss d erfüllen, dass $e \cdot d \equiv 1 \bmod \varphi(M)$.

II. Verschlüsselung.

1. Wenn jemand eine Nachricht x verschlüsseln will, die nur du entschlüsseln können sollst, übergib ihm deine *public keys* M und e (den *private key* d darf nur derjenige kennen, der die Schlüssel generiert hat).
2. Die verschlüsselte Nachricht y ist dann $y = x^e \bmod M$.

III. Entschlüsselung.

1. Erhältst du eine mit deinen *public keys* verschlüsselte Nachricht y, benötigst du nur noch deinen *private key* d.
2. Die entschlüsselte Nachricht ist dann $x = y^d \bmod M$.

Bemerkung: Dass die Entschlüsselung dann tatsächlich wieder den Klartext liefert, kann man mit der folgenden Rechnung einfach einsehen:

$$\begin{aligned}
y^d \bmod M &= (x^e)^d \bmod M \\
&= x^{ed} \bmod M \\
&= x^{\varphi(M)k+1} \bmod M \quad (ed \equiv 1 \bmod \varphi(M),\ k \text{ passend gewählt}) \\
&= (x^{\varphi(M)})^k \cdot x \bmod M \\
&= 1^k \cdot x \bmod M \quad (x^{\varphi(M)} \equiv 1 \bmod M \text{ (Satz von Euler)}) \\
&= x \bmod M \\
&= x \quad (x \in \{1, \ldots, M-1\})
\end{aligned}$$

Aufgabe 1 (RSA-Verfahren | 6 Punkte).

Finde heraus was die Matlab-Befehle `nextprime`, `mod` und `gcd` tun, wie man sie richtig benutzt (siehe z.B. `doc mod` und `doc gcd`) und teste sie dann anhand eines Beispiels. Was liefert der Befehl `[G,U,V] = gcd(A,B)`?

Schreibe eine Funktion `[M,e,d] = generateKey(a,b,c)`, die (gemäss Teil I. oben) drei beliebige Zahlen a, b und c als Input nimmt und daraus die Schlüssel M, e und d des RSA-Verfahrens generiert. Um im letzten Schritt d zu bestimmen, verwende, dass $e \cdot d + \varphi(M) \cdot k = \text{ggT}(e, \varphi(M))$ (für ein passendes $k \in \mathbb{N}$) ist. Beachte ausserdem, dass $d \geq 0$ sein soll. Teste deine Funktion für $a = 17$, $b = 12$, $c = 2022$ (Ergebnis: $M = 221$, $e = 2023$, $d = 151$).

Um den Ausdruck $x^y \bmod n$ zu berechnen, wird Matlab zuerst x^y auswerten, was bei sehr grossen Zahlen, wie wir sie hier verwenden werden, zu Ungenauigkeiten führt. Um dies zu vermeiden, verwenden wir, dass

$$ab \bmod n = (a \bmod n)(b \bmod n) \bmod n$$

gilt. Daraus folgt, dass

$$x^y \bmod n = \underbrace{(\dots(((x \bmod n)x \bmod n)x \bmod n)\dots)x \bmod n}_{y\text{-mal mod anwenden}}.$$

Schreibe eine Funktion `z = myModulo(x,y,n)`, die $x^y \bmod n$ für grosse Zahlen exakt berechnet. Benutze dazu eine `for`-Schleife. Beachte, dass das Eingabeargument `x` auch ein Vektor sein kann! Teste deine Funktion für $x = \left(10^6, 10^7, 10^8\right)$, $y = 5$, $n = 19$ (Ergebnis: $z = (7, 2, 6)$).

Schreibe zwei Funktionen `y = encrypt(t,e,M)` und `t = decrypt(y,d,M)`, die eine Nachricht x wie oben angegeben (Teile II. und III.) ver- bzw. entschlüsseln können. Dabei musst du `double` verwenden, um einen Text t in einen Vektor x umzuformen und `char`, um einen Vektor x in einen Text t umzuformen. Zum Beispiel ist

```
  double('Mathe') = [77,97,116,104, 101] und
char([80,114,97,107,116,105,107,117,109]) = 'Praktikum'.
```

Verwende ausserdem deine Funktion `myModulo`, um Ausdrücke der Form $x^y \bmod M$ zu berechnen.

Um dein Programm abschliessend zu testen, erzeuge mit `generateKey` eigene Schlüssel, lass deinen Sitznachbarn eine Nachricht mit deinen *public keys* M und e verschlüsseln und versuche, sie mit deinem *private key* zu entschlüsseln!