- Fragen?
- Relationen

  A40, 41, 43

- Algebraische Strukturen

| Zahlenmenge | stets ausführbare Operationen | inverse Operationen | neutrale Elemente | „verlorene" Eigenschaft |
|---|---|---|---|---|
| $\mathbb{N}$ | | | | |
| $\mathbb{Z}$ | | | | |
| $\mathbb{Q}$ | | | | |
| $\mathbb{Q} \setminus \{0\}$ | | | | |
| $\mathbb{R}$ | | | | |
| $\mathbb{R} \setminus \{0\}$ | | | | |
| $\mathbb{C}$ | | | | |
| $\mathbb{C} \setminus \{0\}$ | | | | |

- Wiederholung

  $5 : 2 =$

  $12 : 3 =$

  $0 : 6 =$

  $-1 : 9 =$

  $50 : -10 =$

(Mini-Test-Aufgabensammlung)

**4.1.14** Wie ist der Standartrepräsentant einer Äquivalenzklasse einer Äquivalenzrelation modulo $m$ definiert? (mit $m \in \mathbb{N}$)

(Mini-Test-Aufgabensammlung) A47, 48, 49, 50

**5.2.4** Berechnen Sie:

- $[7]_5 + [4]_5 =$
- $[-3]_8 + [10]_8 \cdot [9]_8 =$
- $([2]_{11}^{-1} + [5]_{11}) \cdot [3]_{11} =$
- $[5]_{17}^{-1} \cdot [5]_{17} - [9]_{17} \cdot [9]_{17}^{-1} =$

- RSA

Einleitung/Motivation

- Verschlüsselung aus Gründen des Datenschutzes und der Privatsphäre
  -> symmetrische Verfahren wie (klassisch) Cäsar-, Verschiebe-, Substitutionschiffre, Skytale oder (modern) DES, AES und ChaCha20 [Analogie: Holztruhe mit Schloss]
- Problem: Schlüsselaustausch (z.B. Kommunikation mit Hochschule auf anderem Kontinent)
- Idee: asymmetrische Verfahren (ca. ab den 1970er Jahren)
  -> zweigeteilter Schlüssel: öffentlicher Schlüssel (der beliebig veröffentlicht werden kann; zum Verschlüsseln) und privater Schlüssel (der unbedingt geheim gehalten werden muss; zum Entschlüsseln) [Analogie: Kiste mit Vorhängeschloss]
  => RSA (benannt nach den Erfindern Ronald L. *R*ivest, Adi *S*hamir und Leonard *A*dleman; 1978)

Mathematische Grundkonzepte

- **Division mit Rest**
  Für beliebige ganze Zahlen a und b mit b > 0 gibt es eindeutig bestimmte ganze Zahlen q und r, sodass gilt: $a = q*b + r$ und $0 \leq r < b$.
  Man schreibt auch: $[r]_b = [a]_b$ oder $r \equiv a \mod b$ ("a reduziert modulo b").
  (Achtung: In der Mathematik (und in manchen Programmiersprachen wie Python) ist der Rest immer $\geq 0$, in anderen Programmiersprachen (wie z.B. Java) kann der Modulo-Operator % auch negative Werte liefern.)

  **Bemerkung und Definition 3.11** Es sei $m \in \mathbb{Z} \setminus \{0\}$, dann heißt

  $$R_m := \{(x, y) \in \mathbb{Z} \times \mathbb{Z} : x - y \text{ ist ohne Rest durch } m \text{ teilbar}\}$$

  die **Kongruenz-Relation modulo** $m$. Wie sich relativ einfach zeigen lässt, ist $R_m$ eine Äquivalenzrelation. Man schreibt für $(x, y) \in R_m$

  $$x \equiv y \mod m,$$

  und sagt $x$ ist **kongruent** $y$ **modulo** $m$.

  Zu einem gegebenen $x \in \mathbb{Z}$ ist

  $$\{y \in \mathbb{Z} : (x, y) \in R_m\} = \{y \in \mathbb{Z} : y = x + k\,m,\ k \in \mathbb{Z}\}. \quad (3.4)$$

- **Restklassenring**

**Bemerkung und Definition 4.8** Es sei $m \in \mathbb{Z} \setminus \{0\}$ und $R_m$ die Kongruenz-Relation modulo $m$ (vgl. 3.11). Dann heißt

$$\mathbb{Z}_m := \{[x]_m : x \in \mathbb{Z}\}$$

die Menge der **Restklassen** modulo $m$. Nach (3.4) ist

$$[x]_m := \{y \in \mathbb{Z} : y = x + k\,m,\ k \in \mathbb{Z}\}.$$

Auf $\mathbb{Z}_m$ definieren wir zwei Verknüpfungen wie folgt. Für $x, y \in \mathbb{Z}$ sei die Addition der zugehörigen Äquivalenzklassen definiert durch

$$[x]_m + [y]_m = [x + y]_m$$

und die Multiplikation sei definiert durch

$$[x]_m \cdot [y]_m = [x \cdot y]_m\,.$$

Damit ist $(\mathbb{Z}_m, +, \cdot)$ ein Ring und $\mathbb{Z}_m$ und wird als **Restklassenring** modulo $m$ bezeichnet.

- **Modulares Rechnen ("Rechenregeln in Restklassenringen")**
  Seien a, b, c $\in \mathbb{Z}_n$.
  - (a mod n) + (b mod n) = ((a + b) mod n)
  - (a mod n) - (b mod n) = ((a - b) mod n)
  - (a mod n) * (b mod n) = ((a * b) mod n)
  - (c mod n) * ((a mod n) + (b mod n)) = (c mod n) * (a mod n) + (c mod n) * (b mod n)

  => reduzieren an beliebigen Stellen möglich

- **Größte gemeinsame Teiler (ggT)**
  Für zwei natürliche Zahlen a und b ist der größte gemeinsame Teiler ggT(a, b) definiert als die größte natürliche Zahl, die a und b teilt.
  (Berechnung mithilfe des euklidischen Algorithmus)

- **Multiplikatives Inverses**
  Eine Zahl $x^{-1} \in \mathbb{Z}_n$ heißt multiplikatives Inverses zu x, wenn gilt:
  $x * x^{-1} \equiv x^{-1} * x \equiv 1 \mod n$
  (Berechnung mithilfe des erweiterten euklidischen Algorithmus (siehe Skript Anhang C) oder des Tools von Prof. Konstantin Knorr ( https://public.hochschule-trier.de/~knorr/exeuclid.php ))

  Man kann zeigen, dass das multiplikative Inverse nur für zu n teilerfremde Zahlen (ggT(x, n) = 1) definiert ist. ( https://www.youtube.com/watch?v=P1LTElH7Kjw, https://www.youtube.com/watch?v=ftm-9hgXn4M )

- **Satz von Euler**
  Für natürlichen Zahlen x und n gilt:
  ggT(x, n) = 1 $\Rightarrow$ $x^{\phi(n)} \equiv 1$ mod n.

Schlüsselgenerierung (notwendig pro Kommunikationsteilnehmer) — Beispiel

1. Wähle zufällig zwei beliebige Primzahlen p und q.
2. Berechne den RSA-Modul n = p * q.
3. Berechne die euler'sche Phi-Funktion: $\varphi(n) = (p-1)*(q-1)$.
4. Wähle eine natürliche Zahl e mit $1 < e < \varphi(n)$ und $ggT(e, \varphi(n)) = 1$.
   (Häufig ist e eine "standardisiert" Konstante, oft: e = 2**16 + 1 = 65537.)
   => Dann ist (e, n) der öffentliche Schlüssel.
5. Bestimme d mit $1 < d < \varphi(n)$ und $e*d \equiv 1 \mod \varphi(n)$ (also $d = e^{-1}$).
   => Dann ist (d, n) der private Schlüssel.

RSA-Ver- und -Entschlüsselung

Sei der zu verschlüsselnde Klartext M eine natürliche Zahl mit 0 <= M < n.
Dann berechnet sich der Chiffre-Text wie folgt:

$$C = M^e \bmod n.$$

Sei der zu verschlüsselte Chiffre-Text C eine natürliche Zahl mit 0 <= C < n.
Dann berechnet sich der Klartext wie folgt:

$$M = C^d \bmod n.$$

Korrektheitsbeweis

z. z.: $(M^e)^d \equiv M \mod n$

Annahme: ggT(M, n) = 1.

Gemäß der Konstruktion der Schlüssel gilt:
$e*d \equiv 1 \mod \varphi(n) \Leftrightarrow e*d = 1 + k*\varphi(n)$ (mit $k \in \mathbb{N}$).

$(M^e)^d \equiv M^{(e*d)} \equiv M^{1 + k*\phi(n)} \equiv M^1 * M^{k*\phi(n)} \equiv M * (M^{\phi(n)})^k \equiv M * 1^k \equiv M * 1 \equiv M \mod n$

q. e. d.

Sicherheit/Angriffe

- schwieriges Grundproblem: Faktorisierung großer Zahlen
  -> Es ist leicht, das Produkt zweier Primzahlen zu berechnen, allerdings sind keine Algorithmen mit polynomialer Laufzeit (für klassische Computer) bekannt, um eine gegebene Zahl in ihre Primfaktoren zu zerlegen.

  => Der Aufwand zur Berechnung des geheimen Exponenten aus dem öffentlichen Schlüssel ist äquivalent zur Faktorisierung des RSA-Moduls, dessen Aufwand von der Größe des RSA-Moduls (Schlüssellänge, gemessen in der Anzahl der Bits des Moduls n) abhängt

  Sei eine n-Bit Zahl gegeben.
  • Probedivision: $O(2^{n/2})$
  • Quadratisches Sieb: $O(\exp(c * n^{1/2} * (\ln(n))^{1/2}))$
  => geeignet zur Faktorisierung von Zahlen bis ca. 110 Dezimalstellen
  • Zahlenkörpersieb: $O(\exp(c * n^{1/3} * (\ln(n))^{2/3}))$
  => geeignet zur Faktorisierung von Zahlen ab ca. 110 Dezimalstellen

  **Faktorisierungen von RSA-Moduln**
  - 640 Bit RSA-Modul (November 2005)
  - 768 Bit RSA-Modul (Januar 2010)
  - 829 Bit RSA-Modul (Februar 2020)
  (Übersicht + Preisgelder: https://en.wikipedia.org/wiki/RSA_Factoring_Challenge)

  Allerdings: Mit einem Quantencomputer kann eine Zahl in Polynomialzeit faktorisiert werden
  (-> Algorithmus von Shor: https://www.youtube.com/watch?v=R_LThCHKMKo&list=PLb0zKSynM2PCGMVJiM9Z9fWAXOfb06qe5&t=251s),
  reale Quantencomputer sind derzeit aber noch weit davon entfernt, RSA mit heute üblichen Parameter „gefährlich" zu werden (Faktorisierung von 35, …).
  Da aber heute schon Daten aufgezeichnet werden können, die noch in einigen Jahren relevant sein könnten, gibt es ein zunehmendes Bestreben, kryptologische Post-Quanten-Verfahren (https://www.youtube.com/watch?v=-UrdExQW0cs&t=1062s) zu standardisieren (https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards) und zu implementieren (https://www.mozilla.org/en-US/firefox/132.0/releasenotes/#note-790478).

- sichere Auswahl von RSA-Parametern
  • RSA-Schlüssellänge (Modullänge): $\geq$ 2048 Bit ($\approx$ 609-610 Dezimalstellen)
  • Die Primzahlen p und q sollten möglichst zufällig und gleichverteilt gewählt sein.
  • Aufgrund des „Low-Exponent-Attack" sollte der öffentliche Exponent nicht zu klein sein, üblich ist die Nutzung von $e = 2^{16} + 1 = 65537$.

RSA in der Praxis:
*(Beispiel-Webseiten, die TLS-Cipher-Suites verwenden, die RSA enthalten)*

https://static-rsa.badssl.com/

https://video.hochschule-trier.de/Panopto/Pages/Home.aspx
https://alfresco.hochschule-trier.de/login.php
https://www.ionos.de/

=> Kryptografisch kann RSA zum Verschlüsseln (Geheimhaltung) (bzw. in hybriden Verschlüsselungssystemen zum Schlüsselaustusch) und Signieren (Integrität/Authentizität) verwendet werden.
Heutzutage wird RSA (aus Gründen, die an dieser Stelle nicht weiter ausgeführt werden sollen) hauptsächlich zum Signieren verwendet (https://tls-v1-2.badssl.com:1012/), wobei auf elliptischen Kurven basierende Verfahren wie der "Elliptic Curve Digital Signature Algorithm" (ECDSA) zunehmend verwendet werden (https://ecc256.badssl.com/), da sie schneller sind (Rechenzeit) und mit kürzeren Schlüssel (weniger zu übertragene Daten) als RSA-Signaturen auskommen (um jeweils dasselbe Sicherheitsniveau zu bieten).