Abenteuer Informatik – Zusammenfassung

Kapitel 2 – Ordnung muss sein

- Wie lösen Computer die Aufgabe, eine unsortierte Menge von Daten so zu organisieren, dass der Zugriff auf einen einzelnen Datensatz sehr schnell geht?
- **Problemgröße:**
  - Maß, wie schwierig bzw. umfangreich die Aufgabe bzw. das Problem ist
  - der zu leistende Aufwand hängt von ihr ab
  - *n* = Problemgröße (z.B. Sortieren von 1000 Karten entspricht *n = 1000*
  - die besten Sortierverfahren laufen proportional zu *Problemgröße * log_2 Problemgröße*
- **„divide et impera“:** Gesamtproblem wird in mehrere, voneinander unabhängig lösbare Stücke geteilt; Teillösungen werden dann nur noch zusammengefasst
- **Computerspeicher:**
  - im Computer kann jede Speicheradresse genau einen Wert annehmen → wird ein neuer Wert eingegeben, so wird der alte Wert überschrieben
  - viele Vorgänge benötigen **Zwischenspeicher** i.e. Speicheradressen, die besonders schnell und komfortabel abgefragt werden können
  - ein Computer kann zwar auf seinen gesamten Speicher zugreifen, aber immer nur eine Adresse gleichzeitig auslesen
- **Selection-Sort:**
  - der gesamte Speicher wird ab der ersten Adresse durchlaufen und es wird der niedrigste Wert gesucht → dieser Wert wird dann an den Anfang gestellt → dann wird ab der zweiten Adresse der Speicher erneut durchsucht und wieder der niedrigste Wert auf die zweite Adresse gesetzt → dies wird so lange fortgeführt, bis die letzte Karte sortiert wurde
  - pro Durchlauf wird **eine Karte** an den richtigen (vordersten) Platz gebracht
  - Aufwand pro Karte steigt mit der Anzahl der Karten; das Verfahren ist **nicht linear**
  - **Aufwand = 0.6 * Problemgröße^2** (beim Sortieren von z.B. 10 Karten bräuchte man 60 Elementaroperationen)
  - besser als Bubble-Sort
  - Aufwand pro Karte steigt linear mit der Gesamtanzahl der Karten → bei kleiner Anzahl besser, bei großer schlechter
- **Bubble-Sort:**
  - der gesamte Speicher wird durchlaufen, aber es werden immer nur die beiden nebeneinanderliegenden Werte miteinander verglichen → sind diese richtig sortiert, wird nichts getan, ansonsten werden sie vertauscht → damit stellt jeder Tauschvorgang ein bisschen mehr Ordnung her, bis alle kleinen Teillösungen das Gesamtproblem gelöst haben
  - pro Durchlauf wird **eine Karte** an den richtigen (hintersten) Platz gebracht
  - Aufwand pro Karte steigt mit der Anzahl der Karten; das Verfahren ist **nicht linear**
  - **Aufwand = 1.3 * Problemgröße^2** (beim Sortieren von z.B. 10 Karten bräuchte man 130 Elementaroperationen)

- **Algorithmen:**
  - für ein Problem muss immer ein Algorithmus gefunden werden und unter vielen Algorithmen muss der beste (oft schnellste) gefunden werden
  - **Operationen:** Arbeitsschritte eines Algorithmus, die dem Computer Zeit kosten
  - **Elementaroperationen:** Arbeitsschritte, die bei der Bestimmung der Laufzeit eines Algorithmus für wichtig bzw. zeitkritisch erachtet werden (z.B. Speichervorgänge)
  - **Aufwandsabschätzung:**
    - Qualität von Algorithmen
    - wie stark wächst die notwendige Rechenzeit in Bezug auf die Problemgröße
    - **konstante Faktoren werden nicht berücksichtigt**
    - wirklicher Aufwand (Komplexität) hängt von vielen, schwer zu bestimmenden Faktoren ab → **Ordnung eines Aufwands O** (z.B. Selection- und Bubble-Sort = O (n^2); Tournament-Sort = O (n log n)
- **Tournament-Sort:**
  - ein Turnierraster wird mit Werten aufgefüllt → es treten jeweils die benachbarten Werte gegeneinander an und der höhere Wert steigt in die nächste K.-o.-Runde auf → sobald leere Felder in den K.-o.-Runden entstehen, rutscht die nächste Karte aus der unteren Runde nach → wenn der Sieger (i.e. der höchste Wert) ermittelt wurde, wird dieser am Ende der sortierten Liste abgelegt → die unteren Karten rutschen nach und es werden weiter die Sieger bestimmt und in die Liste eingegliedert, bis alle Karten das Raster durchlaufen haben und die Liste sortiert ist
  - auf den Siegerebenen (K.-o.-Ebenen) gibt es n/2 + n/4 + n/8 + … 1 (Siegerfeld) Felder → geometrische Reihe ≈ *n* → Anzahl der K.-o.-Plätze immer ca. gleich der Anzahl der Teilnehmer
  - Anzahl der Ebenen (K.-o.-Runden) = *h*
  - **Aufwand = *n* * *h*** (Aufwand = *n* Durchläufe mal *h* Verschiebeoperationen)
  - $n \approx 2^h$ (n = Anzahl der Karten, h = Anzahl der Ebenen) → $\log_2 n \approx h$ → je mehr Karten vorhanden sind, desto langsamer wächst die Anzahl der Ebenen
  - **Herleitung der Rechenzeit:**
    - Aufwand zum Aufbau der K.-o.-Tabelle: n * h
    - Höhe h der Tabelle: $\log_2 n$
    - → **Aufwand des ersten Teils des Algorithmus:** $n * \log_2 n = n \log n$
    - zweiter Teil: Siegerkarte nach oben, leeres Feld aufgefüllt → passiert so oft, wie es Karten gibt → n-mal
    - Auffüllung der leeren Felder = Anzahl der Verschiebungen entspricht maximal der Höhe h → $\log_2 n$
    - → **Aufwand des zweiten Teils des Algorithmus:** $n * \log_2 n = n \log n$
    - → **Gesamtaufwand: $2 * n * \log_2 * n$** → (ohne Konstante) **n log n** → viel bessere Laufzeit
  - Aufwand pro Karte steigt logarithmisch mit der Gesamtanzahl der Karten → bei kleiner Anzahl schlechter, bei großer sehr gut (viel weniger Elementaroperationen)
- **Proxmap-Sort:**
  - Sortierverfahren mit geänderten Voraussetzungen: **braucht Informationen über Verteilung der Daten**
  - für jeden Wert wird geraten, wo er in einer Liste hinkommt und er wird dort abgelegt → sollte diese Stelle bereits besetzt sein, so muss man nur wenige Werte verschieben → im Normalfall wird jeder Wert nur einmal betrachtet → um Kollisionen zu vermeiden, ist der Speicher größer als n → es entstehen am Ende Lücken zwischen den Werten, die durch Zusammenrücken letztlich beseitigt werden

→ jeder Wert kommt mit jeweils zwei Operationen aus: Einsortieren und Zusammenrutschen → **O (2n)** → **O (n)**
    - Proxmap-Sort funktioniert bei Gleichverteilung sehr gut → gleichmäßige Verteilung der Werte auf die Speicherplätze → garantiert wenige Kollisionen
    - bei **Normalverteilung** funktioniert Proxmap-Sort nur, wenn die Verteilung der zu sortierenden Größen bekannt ist und man dadurch für jede Größe die Position im Speicher erraten kann
    - in der Praxis werden zuerst alle Daten analysiert, um die gegebene Verteilung zu erkennen
- **Fazit:**
    - **klassische Sortieralgorithmen** (Selection-, Bubble-, und Tournament-Sort) **funktionieren mit nur einer Grundvoraussetzung** (welcher Wert ist größer?) **maximal zur Ordnung O (n log n)**
    - **andere Sortieralgorithmen** (Proxmap-Sort) **funktionieren mit mehreren Grundvoraussetzungen** (Wissen um die Verteilung der Daten) **zur Ordnung O (n)**

Kapitel 3 – Ich packe meinen Koffer

- **Das *Rucksackproblem*:** eine Kiste mit einer bestimmten Größe soll mit Gegenständen unterschiedlicher Größe und Wertes so gefüllt werden, dass die Gegenstände insgesamt den größtmöglichen Wert bei optimaler Ausnutzung der Größe annehmen
    - stures Herumprobieren würde viel zu lange dauern; der Aufwand zum Probieren wächst sehr stark mit der Größe der Kiste → Problem wird sehr unübersichtlich
    - **divide et impera:** Problemgröße = Größe der Kiste + Anzahl unterschiedlicher Gegenstände
    - **Dynamische Programmierung:** Strategie zur Lösung komplexer Probleme (Optimierungsaufgaben)
        - man löst viele Teilprobleme und verwendet diese, um die nächstgrößeren Probleme zu lösen bis das Gesamtproblem gelöst ist
        - es existiert eine Verankerung, bei der das Problem für eine kleine Problemgröße gelöst wird → danach wird jeweils von einer kleineren auf die nächstgrößere Lösung abgeleitet
    - es muss jede Kiste jeder Größe mit jeder Art von Gegenständen vollgefüllt werden, um das Optimum zu bestimmen
    - **Vollständige Induktion:** zweistufiges mathematisches Beweisverfahren (insbesondere für Aussagen in Bezug auf ganze Zahlen):
        - man beweist, dass die Aussage für eine bestimmte kleine Zahl *a* gilt
        - ferner beweise man: falls die Aussage für die Zahl *n – 1* gilt, dann gilt sie auch für die Zahl *n*
        - → damit ist bewiesen, dass die Aussage für alle ganzen Zahlen größer oder gleich *a* gilt
    - **polynomielle Laufzeit:**
        - Laufzeit steigt im Verhältnis zur Problemgröße in einem durch ein Polynom ausdrückbaren Verhältnis (Problemgröße steht bei Potenzen als Basis)
        - In polynomieller Laufzeit lösbare Probleme gehören zur **Klasse P**
        - **Klasse NP** (nicht praktisch berechenbare Probleme): Probleme haben eine Laufzeit, die sich nicht durch ein Polynom ausdrücken lässt (Problemgröße steht bei Potenzen als Exponent)
        - **P = NP:** sind alle Probleme der Klasse NP auch in polynomieller Zeit lösbar?

- das ganzzahlige Rucksackproblem ist durch vollständige Induktion **lösbar**
- Bei nicht ganzzahligen Problemgrößen versagt die vollständige Induktion → für das Problem existiert keine Lösung in polynomieller Zeit → ***NP-vollständiges Problem***

Kapitel 6 – Paketpost

- **LAN** – Local Area Network**, WAN** – Wide Area Network, viele weitere Hierarchiestufen
- **IP-Adressen:** *Internet Protocol* ist ein eindeutiges System zur Identifikation eines Rechners
  - Computer haben IP-Nummern zur eindeutigen Identifikation (durch andere Computer) und IP-Namen zur menschlichen Identifikation (Benutzer)
  - **IANA:** Organisation, die IP-Nummernblöcke an andere Organisationen verteilt
    - **Class-A-Adressen:** erste Zahl wird von der IANA festgelegt; ergibt 256^3 Rechner mit individueller IP-Nummer
    - **Class-B-Adressen:** ersten beiden Zahlen werden von der IANA festgelegt; ergibt 256^2 Rechner mit individueller IP-Nummer
    - **Class-C-Adressen:** ersten drei Zahlen werden von der IANA festgelegt; ergibt 256 Rechner mit individueller IP-Nummer
  - **ICANN:** Organisation, die Domains (Namensbereiche) an Personen oder Organisationen vergibt (max. 255 Zeichen)
- **DNS-Server:** *Domain Name Service* schlüsselt IP-Namen in IP-Nummern auf
  - alle Organisationen, die Domains weiter administrieren, betreiben einen DNS-Server → dieser enthält die Name-zu-Nummer-Tabelle für das eigene Netzwerk und Teile des Internets
  - die DNS-Übersetzungen nimmt teilweise lange Wege in Anspruch und kann länger dauern
- **Router:** spezialisierte Rechner, die Datenpakete im Internet in die richtige Richtung weiterleiten
  - besitzen mehrere IP-Adressen, da sie in mehreren Netzen fungieren
  - **Gateway:** *zugeordneter Router* eines Netzwerks: an diesen werden alle Nachrichten eines Netzwerks geschickt und er verteilt diese auf das weitere Internet
  - besitzen Tabellen, in denen steht, wo sie welche Nachrichten hinschicken müssen, um diese einen Schritt weiter zum Zielort zu bringen (je nachdem wie viele Netzwerke auf einmal verbunden werden, werden die Tabellen komplizierter) → **Routing-Tabellen**
  - das Internet funktioniert im Prinzip durch das Zusammenspiel sehr vieler Router → diese vermitteln Nachrichten anhand festgelegter Routing-Tabellen
  - jeder Router deckt nur einen kleinen Ausschnitt des Internets ab → divide et impera
- **OSI-Modell** *(Open Systems Interconnection Reference Model)***:** Kommunikation wird über unterschiedlichste technische Systeme hinweg ermöglicht und die Weiterentwicklung begünstigt
  - **OSI-Layer 1 (Physical Layer – Physikalische Schicht):**
    - alle Aspekte der physikalisch vorhandenen Übertragungsmedien werden festgelegt
    - mechanische, elektrische und weitere funktionale Hilfsmittel werden zur Verfügung gestellt, um physische Verbindungen zu aktivieren/deaktivieren, sie aufrechtzuerhalten und Daten darüber zu übertragen
    - digitale Datenübertragung auf einer leitungsgebundenen oder leitungslosen Übertragungsstrecke
    - Hardware: Repeater, Hubs, Leitungen, Stecker, etc.

- **OSI-Layer 2 (Data-link Layer – Datenverbindungsschicht):**
  - Datenpakete können verloren gehen oder verfälscht ankommen → es werden zusätzliche Daten wie z.B. Nummern oder Prüfsummen im Rahmen der Kanalkodierung eingefügt, um die Vollständigkeit eines Datenpaketes zu kontrollieren → wenn es unvollständig ist, wird es nochmals angefordert
  - zuverlässige, weitgehend fehlerfreie Übertragung wird gewährleistet und der Zugriff auf das Übertragungsmedium wird geregelt
  - Hardware: Bridge, Switch
- **OSI-Layer 3 (Network Layer – Netzwerkschicht):**
  - prinzipielle Kommunikation der Computer im Internet wird hier geregelt
  - Datenübertragung geht über das gesamte Kommunikationsnetz hinweg und schließt die Wegsuche (Routing) zwischen den Netzwerkknoten ein → direkte Kommunikation zwischen Absender und Ziel möglich
  - Aufgaben der Vermittlungsschicht: Bereitstellen netzwerkübergreifender Adressen (IP), Routing bzw. Aufbau und Aktualisierung von Routingtabellen, Fragmentierung von Datenpaketen
  - Hardware: Router
- **OSI-Layer 4 (Transport Layer – Transportschicht):**
  - komplette Vermittlung der Datenpakete vom Absender zum Ziel wird geregelt
  - es erfolgt eine Priorisierung von Datenpaketen, Segmentierung des Datenstroms und eine Stauvermeidung
  - bietet den anwendungsorientierten Schichten 5 bis 7 einen einheitlichen Zugriff, so dass diese die Eigenschaften des Kommunikationsnetzes nicht zu berücksichtigen brauchen
- **OSI-Layer 5 (Session Layer – Sitzungsschicht):**
  - gesamte Übertragung von Datenpaketen einer bestimmten Sitzung findet statt (Prozesskommunikation zwischen zwei Systemen) → Zugehörigkeit eines Datenpaketes zu einer Sitzung wird durch eine eindeutige Nummer beschrieben
  - um Zusammenbrüche der Sitzung zu beheben, stellt die Sitzungsschicht Dienste für einen organisierten und synchronisierten Datenaustausch zur Verfügung → es werden Wiederaufsetzpunkte (Fixpunkte oder Check Points) eingeführt, an denen die Sitzung nach einem Ausfall einer Transportverbindung wieder synchronisiert werden kann
- **OSI-Layer 6 (Presentation Layer – Darstellungsschicht):**
  - die ersten 5 Schichten kommunizieren untereinander nur in Zahlen → Layer 6 legt fest, wie diese Zahlen in für Menschen verständliche Systeme übersetzt werden
  - gewährleistet, dass Daten, die von der Anwendungsschicht eines Systems gesendet werden, von der Anwendungsschicht eines anderen Systems gelesen werden können → agiert als Datenübersetzer
  - setzt die systemabhängige Darstellung der Daten (z.B. ASCII, Unicode) in eine unabhängige Form um und ermöglicht somit den syntaktisch korrekten Datenaustausch zwischen unterschiedlichen Systemen
  - weitere Aufgaben: Datenkompression, Verschlüsselung
- **OSI-Layer 7 (Application Layer – Anwendungsschicht):**
  - Computer kommunizieren in bestimmten Fachgebieten mit bestimmten Protokollen → diese sind in Layer 7 spezifiziert
  - Dienste, Anwendungen und Netzmanagement

  - stellt Funktionen für die Anwendungen zur Verfügung
  - stellt die Verbindung zu den unteren Schichten her
  - auf dieser Ebene findet auch die Dateneingabe und -ausgabe statt.
  - die Anwendungen selbst gehören nicht zur Schicht.
  - Anwendungen: Webbrowser, E-Mail-Programm
- **Internet:** bekannte Strukturen aus der Realität werden verwendet, um ein technisches System nach dem gleichen Modell aufzubauen → **Paradigmenbildung** (**Paradigma:** Denkmuster/Vorbild, anhand dessen ein technisches System oder abstraktes Konzept konstruiert wird, wodurch es für die Anwender leichter zu verstehen ist)

Kapitel 8 – Ordnung im Chaos

- Kann ein Computer mit einer scheinbar unsortierten Folge an Daten etwas anfangen?
- **Sequentielle Suche:** will man in einem **chaotischen Speicherinhalt** einen bestimmten Wert suchen, so kann man einen Wert nach dem anderen durchsuchen (solange der gesuchte Wert definitiv im Speicher ist) → durchschnittlich muss die Hälfte aller Werte durchsucht werden, bis der gesuchte Wert gefunden wird → bei doppelt so vielen Werten braucht man doppelt so viele Vergleiche
- **Binäre Suche:** will man in einem **sortierten Speicherinhalt** einen bestimmten Wert suchen, so kann man in der Mitte beginnen und halbiert dann immer die Anzahl noch möglicher Zahlen → so wird immer eine Hälfte der Werte ausgeschlossen → durchschnittlich braucht man 4-5 Vergleiche → bei doppelt so vielen Werten braucht man nur einen Vergleich mehr
- **Modulo:** ist eine Bezeichnung für eine eigene Rechenart = die Berechnung des Restes einer Division
- **Sortierkriterien:** Daten können nach vielen verschiedenen Kriterien sortiert werden
  - um Menschen etwas zu präsentieren, werden nur menschliche Sortierkriterien verwendet
  - wenn die Daten nur im Computer gespeichert werden und dort verbleiben, werden *seltsame* Sortierkriterien verwendet
- **Hashing:** aufgrund einer Formel wird der Speicherort für einen Datensatz festgelegt; dieser kann aufgrund derselben Formel leicht wiedergefunden werden
  - nach einem bestimmten Eintrag wird nicht gesucht, sondern seine Position wird errechnet
  - zum Speichern und Abrufen der Daten genügt jeweils ein einziger Vorgang
  - **benötigt Platz: Hashing wird eingesetzt, wenn der schnelle Zugriff auf Daten wichtiger als der kleine Speicherverbrauch ist**
  - aufgrund der bestimmten Verteilung (Häufung) von Daten, kann keine sortierte Folge entstehen → um die Verteilung herauszufinden, müsste zuerst eine statistische Analyse aller Daten erfolgen → oft unmöglich → Sortierkriterium wird geändert, um Kollisionen von Daten zu vermeiden
  - **Kollisionsbehandlung:** wird durchgeführt, sobald ein Datensatz nicht an seiner eigentlichen Stelle in den Speicher geschrieben werden kann, da dieser bereits belegt ist → ist der Speicherplatz zu dicht belegt, nimmt man eine klassische Sortiermethode zusammen mit der binären Suche
  - **Belegungsgrad:** gibt an, wie viel Speicher prozentual ausgenutzt ist
  - **je kleiner der Belegungsgrad, desto besser funktioniert das Hashing → durchschnittlich bei 85%** (bei 20% wird nur noch ca. 1 Zugriff benötigt)
  - **Schlüssel *s*:** Wert oder Zeichenfolge, nach der sortiert wird → daraus wird direkt über eine Berechnung die Speicherposition bestimmt

- **Hashfunktion *h(s)*:** Berechnungsvorschrift (Funktion), anhand welcher die Daten sortiert werden
  - **Position = h(s)**
  - Durch die Hashfunktion werden die Datensätze einsortiert, aber auch wiedergefunden
  - **Kriterien für sinnvolle Hashfunktion:** vom Wertebereich der Hashfunktion ist die Größe des zur Verfügung gestellten Speichers abhängig → Speicherbereich muss groß genug sein, um alle Schlüssel bei einem Belegungsfaktor von unter 0,8 aufzunehmen
    - um Häufungen zu vermeiden, wird oft *modulo* verwendet → *modulo* durch eine Primzahl arbeitet sogar unabhängig jeglicher Zahlensysteme (dezimal, binär, etc.) → sorgen für Entkopplung der Speicherpositionen von ihren dezimalen Abhängigkeiten
  - **Qualität eines Hashs wird durch seine Hashfunktion und durch seine Kollisionsbehandlung bestimmt**
    - Startfunktion: z.B. $h_0(s) = s \bmod 103$
    - Kollisionsbehandlung: z.B. $h_i(s) = (h_0(s) + i) \bmod 103$ → die Kollisionsbehandlung durchsucht die Speicherplätze nacheinander, bis ein freier Platz gefunden wird
    - *„für den i-ten Versuch, s unterzubringen, gehe vom Speicherplatz, den die Hashfunktion ausgesucht hat, i Plätze nach recht, fange nach dem Erreichen des Endes wieder von vorne an*“
    - gute Kollisionsbehandlung ist enorm wichtig, da sonst nicht alle Daten eingeordnet werden könnten
    - kommt es doch zu Häufungen bei der Hashfunktion, werden die Kollisionen immer wieder auf den gleichen Speicherstellen stattfinden → Kollisionsbehandlung muss vom Schlüssel abhängig gemacht werden → oft sehr schwierig, eine Funktion zu finden, die alle Speicherstellen adressiert
    - in der Praxis: fast immer *modulo* mit simpler Kollisionsbehandlung (wie oben)
  - **Anwendungen:** überall dort, wo man Daten sehr schnell ablegen und wieder hervorholen muss
    - Datenbanken, Betriebssysteme, Echtzeitsysteme (z.B. Fließband = schnelle Entscheidungen müssen getroffen werden)

- **Erzeugung von Zufallszahlen:**
  - Computer nutzen Zufallszahlen, um konkrete Berechnungen anzustellen
    - man kann z.B. die Zahl π anhand ihrer geometrischen Definition bestimmen (siehe Seite 246f)
  - Computer können allerdings gar keine Zufallszahlen erzeugen
    - Der Hauptzweck von Computern besteht darin, Abläufe immer wieder so zu wiederholen, wie sie einmal programmiert wurden → Zufall ist dadurch kultivierte Unzuverlässigkeit → Computer simulieren den Zufall → alle Zufallszahlen von Computer sind berechnet und dadurch **Pseudozufallszahlen**
    - bei der Berechnung von PZZ geht man rekursiv vor: es wird mit einer bestimmten Zahl als *Zufallszahl 0* begonnen → der Rest wird anhand einer Funktion bestimmt
    - eine wichtige Anforderung für Pseudozufallszahlgeneratoren ist es, irgendwann einmal alle Zahlen hervorzubringen → wie beim Hashing

  - die meisten Generatoren arbeiten mit Zahlen zwischen 0 und 18 Trilliarden → braucht man kleinere PZZ, so werden einfach die hinteren Stellen von größeren PZZ verwendet
  - zur ersten Initialisierung seines PZZ-Generators die Zeit, die seit dem Starten des Computers in Millisekunden vergangen ist als echte Zufallszahl

Kapitel 9 – Mit Sicherheit

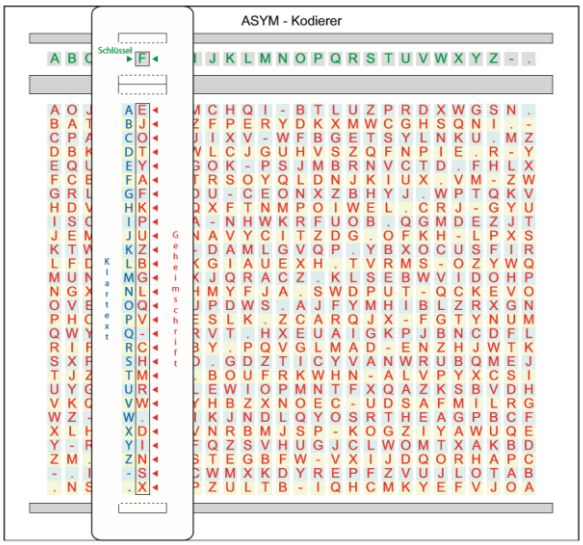
- Welche Möglichkeiten zum Schutz von Daten existieren?
- **Sicher**: Nachrichten kommen unversehrt ausschließlich beim vorgesehenen Empfänger an
- **Skytalen**: Methode zur Verschlüsselung von Daten der Spartaner; paarweise für je einen Kommunikationspartner; Holzstäbe mit identischen Durchmessern, Papyrus wird spiralförmig drumgewickelt → erste **Chiffriermaschine** der Welt → erzeugt eine **Transpositions-Chiffre** (Geheimbotschaften, die genau dieselben Zeichen enthalten wie der Klartext, aber in veränderter Reihenfolge)
- **Cäsar-Scheibe:** jeder Buchstabe des Klartextes wird durch einen anderen Buchstaben ersetzt, der im Alphabet eine entsprechende Zahl von Positionen später kommt; ist man am Ende des Alphabets angelangt, fängt man wieder von vorne an → erzeugt eine **Substitutions-Chiffre** (Geheimbotschaften, in denen die Zeichen gegen andere ausgetauscht werden, die Position im Text aber gleich bleibt)
- Experten beschäftigen sich eher mit dem Brechen bekannter Verschlüsselungsverfahren als mit der Entwicklung neuer Methoden → dadurch sollen Systeme sicherer gemacht und besser vor unberechtigtem Zugriff geschützt sein werden
- Kryptographen nutzen die **unterschiedliche Häufigkeit einzelner Buchstaben**, um Geheimbotschaften zu entschlüsseln → Häufigkeit einzelner Buchstaben innerhalb der Botschaft muss ermittelt werden und mit der allgemeinen Häufigkeit dieser Buchstaben in unserer Sprache verglichen werden → viele Verfahren zur Entschlüsselung von Geheimbotschaften basieren auf einer **statistischen Analyse**
  - z.B. kommt der Buchstabe „E“ sehr häufig vor und ist somit leicht zu identifizieren → man kann für häufige Buchstaben **mehrere Codes verwenden** oder die **Daten vor der Verschlüsselung mit dem Huffman’schen Verfahren komprimieren** → dadurch werden die Häufigkeiten ca. gleich verteilt
  - mit Hilfe von Computern können deutlich komplexere Verschlüsselungen umgesetzt werden: z.B. Cäsar-Scheibe, bei der immer die Kombination mehrerer Buchstaben einem individuellen Code zugeordnet werden
- **sichere Kommunikation mit einem Freund:**
  - Schlüsselaustausch (durch persönliches Treffen)
  - der Bote kann dann Nachrichten weder lesen, verfälschen noch austauschen
  - → moderne Verschlüsselungsverfahren können dies gewährleisten
- **sichere Kommunikation im Internet:**
  - persönliches Treffen zum Schlüsselaustausch nicht möglich → findet ohne Schutz über das Internet statt → durch Hacker leicht abfangbar → **Hauptproblem:** ein und derselbe Schlüssel wird für die Ver- und Entschlüsselung verwendet → **symmetrische Verschlüsselung**
  - **asymmetrische Verschlüsselung:** zwei unabhängige Schlüssel werden wechselweise zur Codierung und Decodierung von Nachrichten verwendet
- **ASYM-Codierer:** asymmetrische Chiffriermaschine

- pro Schlüsselbuchstaben gibt es eine asymmetrische Liste an Substitutionsbuchstaben
- ein bestimmter Schlüssel (z.B. EMU) wird so oft wie notwendig wiederholend über den zu verschlüsselnden Klartext geschrieben und dadurch eine verschlüsselte Nachricht erzeugt

EMUEMUEMU EMUEMUEMUE
ABENTEUER INFORMATIK

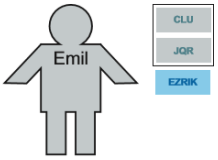
FRCZFCJPE YJUCQBFFQH

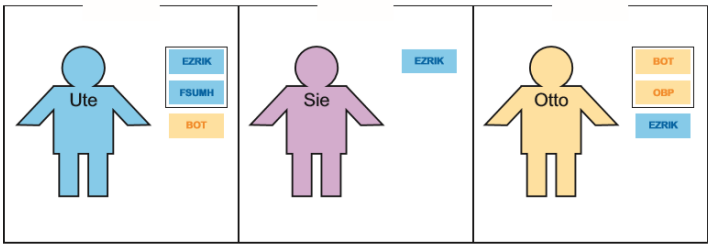
- **der Schlüssel zum Decodieren ist nicht derselbe** (z.B. EMU → FIR) → es gibt immer zwei zusammenhängende Schlüsselpaare
- **ASYM-Codierer erlaubt es, einen Code zu benutzen, der zwei unterschiedliche Schlüssel hat → verwendet man beide Schlüssel hintereinander, so entsteht wieder dieselbe Nachricht (Reihenfolge der Anwendung egal) → eine einmal verschlüsselte Nachricht ist ausschließlich mit ihrem Gegenschlüssel zu decodieren**



- **Experiment zur Nachrichtenverschlüsselung (ASYM-Codierer, Schlüsselaustausch, Authentifizierung, Zertifizierung):**
  - Ich (Kunde) möchte meine Kreditkartennummer (ZWEIVIERDREIACHT) durch Emil (Bote) an Otto (Anbieter) schicken
  - Emil darf keine Chance erhalten, meine Nachrichten auszuspionieren
  - zur Nachrichtenübermittlung muss vorher ein Schlüssel ausgetauscht werden
  - Ich als normaler Benutzer im Internet habe keine Schlüssel, Otto als Anbieter hat einige Schlüsselpaare, Emil als Hacker hat sich auch einige Schlüssel errechnet
  - Otto nimmt ein Schlüsselpaar (z.B. BOT und OBP) und markiert sie:
    - BOT ist **öffentlich**: jeder, der ihn wissen möchte, kann ihn bekommen
    - OBP ist **privat**: bleibt geheim
  - **Schlüsselübergabe 1:** Ich schicke Otto eine Nachricht „Wie lautet dein Schlüssel?“ in Klartext → Emil überbringt diese an Otto → Otto schickt „BOT“ unverschlüsselt über Emil an mich → Ich verschlüssele meine Kreditkartennummer mit BOT und schicke sie über Emil an Otto → Otto decodiert die verschlüsselte Nummer mit seinem privaten Schlüssel OBP → Otto schickt eine unverschlüsselte Empfangsbestätigung über Emil an mich
    - **Sabotage 1:** Emil könnte die verschlüsselte Nachricht mit meiner KKN nie an Otto weiter, sondern überbringt mir direkt eine Empfangsbestätigung → Otto erhält meine KKN nicht und es kommt kein Geschäft zustande
    - → **Authentifizierung:** auch die Empfangsbestätigung von Otto muss durch seinen privaten Schlüssel codiert werden → **Bestätigung, dass eine Nachricht auch von dem kommt, der als Absender angegeben ist** (Emil kann dadurch die Nachricht mit dem öffentlichen Schlüssel decodieren, aber selbst so eine Nachricht nicht erzeugen, da ihm der private Schlüssel fehlt)
  - Zwischenresümee:
    - wenn jemand ein Schlüsselpaar besitzt und einen Schlüssel als öffentlichen Schlüssel bekannt macht, dann kann jeder ihm geheime Nachrichten schicken
    - er selbst kann niemandem geheime Nachrichten schicken, da diese durch den öffentlichen Schlüssel decodiert werden könnten
    - er kann aber seine Nachrichten authentifizieren

- **Schlüsselübergabe 2:** Ich frage Otto nach dem Schlüssel → Emil erhält die Antwort BOT, schickt mir allerdings seinen eigenen öffentlichen Schlüssel CLU → fällt nicht auf, da es noch keine Möglichkeit zur Authentifizierung gab → ich codiere meine KKN mit CLU und schicke es an Otto → Emil kann meine KKN mit seinem eigenen privaten Schlüssel decodieren, lesen, und, um keinen Verdacht zu erwecken, mit Ottos öffentlichem Schlüssel BOT codiert an Otto weitersenden
  - **Sabotage 2:** Emil kann also bei der ersten Schlüsselübergabe bereits die Schlüssel gegen seine eigenen tauschen und bildet dadurch eine Art Zwischeninstanz und kann alle Nachrichten lesen
  - **→ die asymmetrische Verschlüsselung erlaubt eine grundsätzlich sichere Kommunikation mit einem Partner → die Schlüsselübergabe ist aber nicht sicher**
- **Zertifizierungsinstanz (Certification Authority):** Instanzen, die die sichere Aufbewahrung und Weitergabe von Schlüsseln gewährleisten
  - KKN-Übermittlung im Internet: Browser prüft anhand einer CA den korrekten öffentlichen Schlüssel des Anbieters → Browser haben die öffentlichen Schlüssel der wichtigsten CAs bereits fest eingebaut → Verbindung ist von Anfang an sicher
  - Bekannte CAs: DST, Equifax, Deutsche Telekom, Verisign
- **Ausgangssituation mit CA:** CA Ute übergibt jedem persönlich ihren öffentlichen Schlüssel EZRIK, um die Echtheit zu garantieren (keine Verfälschung möglich); Ute bietet an, dass man sich mit ihr treffen kann, um einen Schlüssel sicher zu übergeben → Otto übergibt ihr seinen Schlüssel BOT → Ute prüft die Authentizität seiner Identität und archiviert den Schlüssel →





- **Fortsetzung:** jeder kennt sein eigenes Schlüsselpaar; jeder kennt den öffentlichen Schlüssel der CA EZRIK; **sicher:** persönliche Schlüssel und öffentlicher Schlüssel von CA (weil er persönlich übergeben wurde) → mit CA kann man sicher kommunizieren
- **Schlüsselübergabe 3 (mit CA):** Ich schicke eine Anfrage an die CA mit der Frage nach Ottos Schlüssel (mit öffentlichem Schlüssel von CA codiert) → Emil kann diese Nachricht nicht entschlüsseln (er bräuchte den privaten Schlüssel von CA) → CA schickt mir Ottos Schlüssel BOT (mit privatem Schlüssel von CA codiert) → jeder kann diese Nachricht mit öffentlichem Schlüssel von CA entschlüsseln → ich kenne nun Ottos Schlüssel und kann mit ihm kommunizieren → ich schicke meine KKN und er schickt eine Bestätigung mit seinem privaten Schlüssel codiert zur Authentifizierung
  - Emil kann trotzdem jede Nachricht abfangen und durch eine andere ersetzen → darf nicht zulassen, dass ich Ottos öffentlichen Schlüssel bekomme, da er dann chancenlos wäre

        - **Sabotage 3:** Emil trifft sich auch mit der CA und hinterlegt dort seinen öffentlichen Schlüssel → nun kann die CA auch seinen Schlüssel auf Anfrage weitergeben → wenn ich die CA nach Ottos Schlüssel frage, kann Emil die Nachricht zwar nicht decodieren (er bräuchte den privaten Schlüssel der CA), aber er rät, dass ich Ottos Schlüssel erfrage → Emil fängt meine Nachricht ab und ersetzt sie durch die Anfrage auf Emils Schlüssel → ich kann meine Nachrichten ohne eigenes Schlüsselpaar nicht authentifizieren → CA schickt mir Emils öffentlichen Schlüssel und authentifiziert sie durch die Verschlüsselung mit ihrem privaten Schlüssel → ich bin von Echtheit überzeugt und kommuniziere von hier an nur noch mit Emil → er kann seine Spuren verwischen
        - **Schwachstelle 1: CA kann nicht feststellen, ob die Anfrage nach einem Schlüssel echt ist oder von einer dritten Partei gefälscht ist → daran kann man systematisch nichts ändern**
        - **Schwachstelle 2: ich kann nicht feststellen, ob ich den Schlüssel von Emil oder von Otto bekommen habe →**
    - **Schlüsselübergabe 4:** gleich wie Schlüsselübergabe 3, aber die CA sendet nicht nur den Schlüssel, sondern in der gleichen Nachricht auch die Information, zu wem dieser Schlüssel passt → ich registriere immer, welchen Schlüssel ich bekommen habe
    - **Resümee:**
        - bei der Verwendung der asymmetrischen Verschlüsselung hat immer eine Partei ein Schlüsselpaar und eine andere nicht (Realität: Banken, Geschäfte, andere Anbieter haben Schlüssel, Benutzer oder Kunden haben keine)
        - **Kommunikation vom Kunden zum Anbieter ist immer geheim, aber nicht authentifiziert**
        - **Kommunikation vom Anbieter zum Kunden ist immer offen, aber authentifiziert**
    - bei der asymmetrischen Verschlüsselung kann jeder Hacker heimlich alles lesen, was vom Anbieter an mich verschickt wird → Kommunikation ist authentifiziert, aber nicht geheim → **Geheimhaltung in beide Richtungen ist erforderlich → durch symmetrische Verschlüsselung gewährleistet** → sichere Schlüsselübergabe durch asymmetrische Verschlüsselung
        - nachdem ich von der CA Ottos öffentlichen Schlüssel erhalten habe, schicke ich Otto einen Schlüssel für die Cäsar-Scheibe → Emil kann diese Nachricht nicht decodieren (er bräuchte Ottos privaten Schlüssel) → Otto antwortet mit einer symmetrisch verschlüsselten Nachricht → ab diesem Zeitpunkt ist die Kommunikation sicher
        - → so funktioniert die sichere Kommunikation im Internet: asymmetrische Verschlüsselung wird nur zu Beginn verwendet, um einen Schlüssel für ein symmetrisches Verfahren sicher auszutauschen → dieser Schlüssel wird für die Sitzung verwendet
- die im Experiment durchgeführten Verfahren sind nicht wirklich sicher, sondern dienen nur zur Veranschaulichung der wichtigsten Prinzipien der Sicherheitstechnologie im Internet
- sichere Verschlüsselungsverfahren müssen die **statistischen Besonderheiten der Dateien verschleiern**
    - Schemata der Darstellung von Daten auf Webseiten von Anbietern (Banken, Online-Geschäfte, etc.) können analysiert werden und dadurch dürfen keine Schlüssel errechnet werden können

  - einige Methoden, die das verhindern: DES (Data Encryption Standard), AES (Advanced Encryption Standard), RC4 (Ron's Cipher 4)
- **Länge des Schlüssels ist entscheidend für die Sicherheit der verwendeten Codierungsmethode**
  - Es gibt 2^{48} (ca. 281 Billionen) Kombinationen einer Zahl der Länge von 48 Bit → die meisten Verfahren können nicht einfach irgendwelche Zahlen als Schlüssel verwenden → Länge des Schlüssels nur ein Anhaltspunkt für die Sicherheit

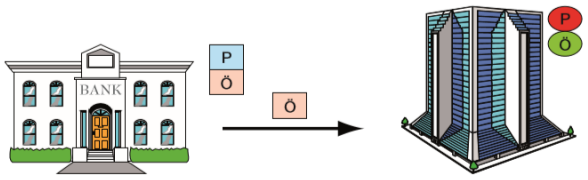
Länge (symmetrisches Verfahren)	Länge (asymmetrisches Verfahren)	Dauer zum Knacken
56 Bit	384 Bit	Minuten bis Stunden
64 Bit	512 Bit	Tage bis Wochen
112 Bit	1792 Bit	mehrere Jahre
128 Bit	2304 Bit	Jahrzehnte

- **Sichere Schlüsselübergabe: mithilfe eines asymmetrischen Verfahrens wird ein Schlüssel für das symmetrische Verfahren ausgetauscht → Kommunikation erfolgt über das symmetrische Verfahren**
- **RSA-Verfahren:** Methode zur asymmetrischen Verschlüsselung
  - eine sehr große Zahl wird in ihre Primfaktoren zerlegt → sehr aufwendig → Zahl muss durch alle möglichen Primzahlen dividiert werden, bis ein glattes Ergebnis herauskommt
  - aus zwei zufällig bestimmten, sehr großen Primzahlen werden nach einem mathematischen Verfahren zwei Schlüssel abgeleitet → je ein öffentlicher und ein privater
  - nach Vernichtung der Primzahlen und der Zwischenberechnungen ist es sehr schwer, aus dem öffentlichen Schlüssel den privaten zu errechnen
- **Zertifizierung:** Serverbetreiber hinterlegen ihren öffentlichen Schlüssel bei einer CA → diese signiert den Schlüssel: er wird mit Namen des Eigentümers, Gültigkeitsdauer + anderen Informationen als neuer Datensatz abgelegt → Datensatz wird mit privatem Schlüssel der CA codiert und an jeden weitergegeben, der ihn haben möchte → **sicher**
- **Digitale Unterschrift:**
  - jeder, der im Besitz eines privaten Schlüssels ist, kann einen beliebigen Datensatz unterschreiben → verschlüsselt mit privatem Schlüssel → jeder kann die Nachricht mit öffentlichem Schlüssel lesen, aber **nicht manipulieren,** da man sie nicht neu codieren kann
  - **Zeitstempel:** Nachweis, wann die Nachricht gesendet wurde, um Manipulation vorzubeugen
- die öffentlichen Schlüssel der wichtigsten CAs sind bereits in Browsern oder Betriebssystemen vorinstalliert
  - CAs, die ihre Schlüssel noch nicht vorinstalliert haben, haben ihre öffentlichen Schlüssel bei bekannteren CAs zertifizieren lassen → Kreis der vertrauenswürdigen Kommunikationspartner vergrößert sich kontinuierlich
- **Fingerabdruck:** für jeden Schlüssel eindeutiger Prüfcode
- **Vertrauensverhältnisse bei der Kommunikation im Internet (z.B. Netbanking):**
  - **Benutzer gegenüber Bank:**
    - die Bank muss in der Lage sein, Angreifer aus ihrem System fernzuhalten → wenn nicht, sind alle Kundendaten unsicher
    - Banken investieren viel Geld und Energie in Abwehrsysteme gegen Hacker
  - **Benutzer gegenüber CA:**
    - CA ist der Dreh- und Angelpunkt der sicheren Kommunikation
    - bei einem erfolgreichen Angriff, kann ein Hacker falsche Zertifikate ausstellen
    - CAs gelten allgemein als sicher, sind staatlich überwacht
    - CAs können bei Ausspähung eines Schlüssel diesen sperren

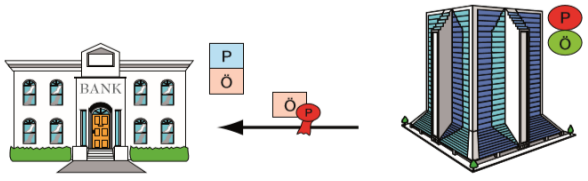
- **Benutzer gegenüber asymmetrischem Verfahren:**
  - erste sichere Kommunikation erfolgt mit RSA → bisher sicherstes Verfahren
  - man braucht einen ausreichend großen Schlüssel (1024 Bit aufwärts)
- **Benutzer gegenüber symmetrischem Verfahren:**
  - bei großen Schlüssellängen (128 Bit aufwärts) kann man von einer sicheren Funktion ausgehen
- **Benutzer gegenüber Computer:**
  - wenn Malware installiert wurde, ist der Computer nicht mehr sicher und eine sichere Kommunikation – egal durch welche Verschlüsselungsverfahren – ist nicht mehr gewährleistet
  - Betriebssysteme bieten unzureichenden Schutz → Virenabwehr muss installiert sein
- **Benutzer gegenüber Benutzer:**
  - **Phishing:** Identitätsfälschung bei E-Mails, etc.
  - Malware-Installation

- geheime Botschaften können in der immer wachsenden Datenflut des Internets versteckt werden → die Sicherung des geistigen Eigentums kann durch unsichtbare **digitale Wasserzeichen** (Markierungen in dem zu sichernden Material) gewährleistet werden → Urheberschaft kann vom Autor nachgewiesen werden → Vorteil: man kann das digitale Wasserzeichen erst entfernen, wenn man es gefunden hat → sehr schwierig
- **Resümee der sicheren Kommunikation durch asymmetrische und symmetrische Verfahren:**
  - **Wie kommt die Bank zu ihrem Zertifikat, das die Voraussetzung für die sichere Kommunikation mit einem Kunden ist?**

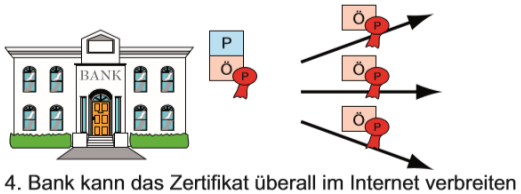
1. Schlüsselpaar generieren



2. Öffentlichen Schlüssel mit Nachweis der Identität zu einer CA bringen

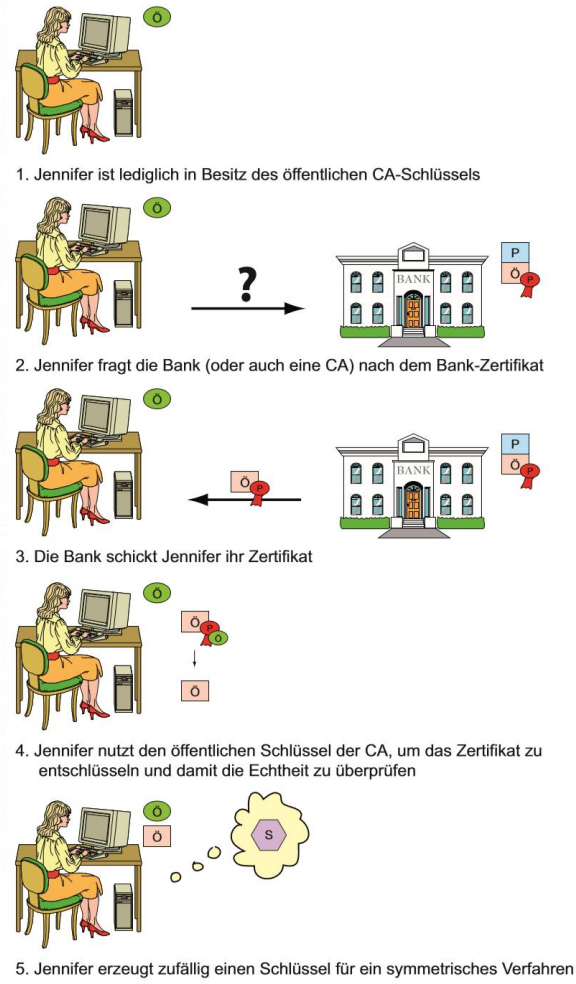


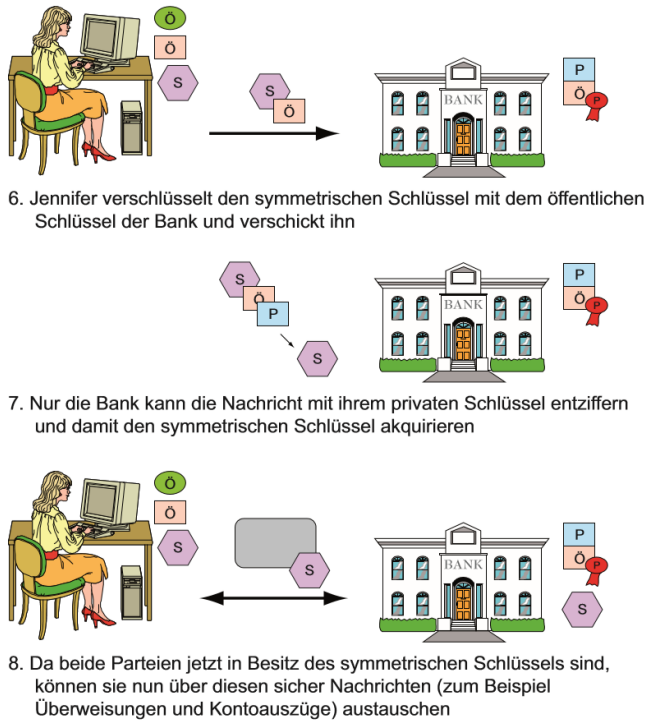
3. CA verschlüsselt den öffentlichen Schlüssel der Bank mit eigenem privaten Schlüssel (das ist das Zertifikat)



4. Bank kann das Zertifikat überall im Internet verbreiten

- **Wie kann ein symmetrischer Schlüssel ausgetauscht werden, damit der Kunde (Jennifer) sicher mit der Bank Überweisungen austauschen kann?**





Kapitel II – InformaGik

- ein Feld aus X und O enthält 25 unabhängige Informationseinheiten → diese Informationen können nach einer festen Regel (jede Zeile und Spalte hat eine gerade Anzahl an X) auf 36 ergänzt werden → **Informationsgehalt ändert sich dadurch nicht**
- Informationen lassen sich nach derselben Regel auch wieder entfernen → beliebige Zeilen und Spalten können entfernt werden
- Zusätzliche Karten ermöglichen die Identifikation eines willkürlich gesetzten Fehlers
- Fehlerreduktion ist besonders in Systemen, die sehr große Datenmengen auf engstem Raum speichern, schwierig (z.B. Blu-Rays)
- Sicherung von Daten gegen **unbeabsichtigtes Verfälschen:** redundante Daten erhöhen den Informationsgehalt eines Datensatzes nicht, aber sie ermöglichen, dass eine gewisse Zahl von Fehlern erkannt und unter Umständen auch korrigiert wird
- Feld aus X und O:
  - ein Fehler: kann erkannt und korrigiert werden
  - zwei Fehler: kann mit gleicher Wahrscheinlichkeit beseitigt oder verschlimmert werden (2 → 4 Fehler) → gar nicht korrigieren
  - drei Fehler: kann erkannt werden

- Verfahren, die mit Hilfe zusätzlicher Daten Informationen toleranter gegenüber Fehler machen: **Man muss sich jeweils entscheiden, ob man Fehler nur erkennen oder auch korrigieren möchte → die Korrektur funktioniert nur mit einer kleineren Anzahl von Fehlern → man muss bei mehr Fehlern eine falsche Korrektur in Kauf nehmen**
- **die Zahl korrigierbarer bzw. erkennbarer Fehler lässt sich durch mehr redundante Informationen steigern**
- Feld aus X und O: fügt man 2 Zeilen und Spalten hinzu, so lassen sich die Fehlerquellen besser auswerten → ist eine Fehlerquelle sowohl in einer Spalte als auch Zeile, so ist der Fehler dort am wahrscheinlichsten → sollte korrigiert werden → bis zu zwei Fehler lassen sich erkennen → Vertauschen von Karten wird vorgebeugt
- Fehlerkorrekturverfahren: bestimmte Fehlerarten lassen sich besser korrigieren als andere → Vertauschen von Stellen soll vermieden werden → z.B. bei Kreditkarten oder Ausweisen wurden **Prüfziffern** eingeführt → wird so ergänzt, dass die gesamte Zahlenkolonne eine festgelegte Regel einhält → wird durch Vertauschen gestört
    - bei Kreditkarten: abwechselnd wird jede Ziffer mit 2 und 1 multipliziert → Summe muss ein Vielfaches von 10 sein
        - Fehler: Vertauschen bei 0 und 9 fällt nicht auf
    - bei Reisepass: letzte Ziffer eines Blocks (Geburtsdatum, Ausweisnummer, etc.) ist eine Prüfziffer → Ziffern werden nacheinander mit 7, 3 und 1 multipliziert → Prüfziffer ist die letzte Ziffer dieser Summe
        - Fehler: 0, 1, 5, 6 haben beim Vertauschen keinen Effekt auf das Ergebnis
    - → Kombination aus beiden Verfahren würde besser funktionieren
- **Codierung:** Darstellung einer Informationseinheit durch eine Folge von Symbolen (z.B. Binärsystem)
    - jede mögliche Folge von Symbolen kann einer bestimmten Informationseinheit zugeordnet werden

A	B	C	D	E	F	G	H
000	001	010	011	100	101	110	111

    - 011000010111 = DACH → jedes einzelne Bit ist relevant; wird es falsch übertragen, ist die Nachricht falsch
    - Richard Hamming: **ein Code ist gegen Fehler sicherer, wenn die verwendeten Symbolfolgen möglichst unterschiedlich sind**
    - DACH benötigt doppelt so viele Bits wie vorher

A	B	C	D	E	F	G	H
000000	000111	011001	011110	101010	101101	110011	110100

    → passiert ein Fehler, so wird die fehlerhafte Codepassage mit den möglichen Codes verglichen und derjenige als Korrektur ausgewählt, bei dem die wenigsten Zeichen verschieden sind
    - in der verwendeten Code-Tabelle

A	B	C	D	E	F	G	H
000000 001000	000111 001000	011001 001000	011110 001000	101010 001000	101101 001000	110011 001000	110100 001000

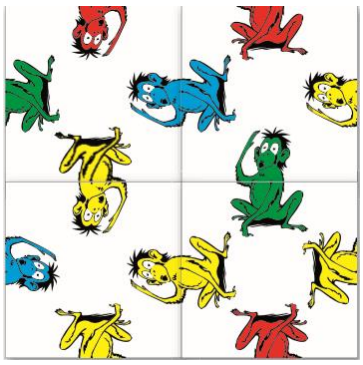
    unterscheiden sich zwei beliebige Codes immer an mindestens drei Stellen → **Hamming-Distanz von 3** → bei einem einzelnen Bitfehler gibt es immer einen einzelnen Code, der die höchste Übereinstimmung mit der fehlerhaften Nachricht hat
    - die kleinste Hamming-Distanz, die zwei beliebige Codes einer Code-Tabelle haben, bestimmt die Hamming-Distanz der Tabelle
    - Codes mit einer höheren Hamming-Distanz sind noch robuster gegenüber Fehlern, da die korrekten Codes so unterschiedlich sind, dass gegebenenfalls auch mehrere Fehler erkannt und korrigiert werden können
    - Hamming-Distanz von 6: pro Zeichen dürfen 2 Bits falsch übertragen werden, 3 Fehler können erkannt werden, aber eventuell nicht korrigiert werden, mehr als 3 Fehler könnten nicht mehr korrigiert werden

A	B	C	D	E	F	G	H
00000000000	00000111111	00111000111	00111111000	11001001011	11001110100	11110001100	11110110011

  - **die Wahl der richtigen Hamming-Distanz und damit Fehlertoleranz eines Codes kann individuell je nach erwarteter Fehlerwahrscheinlichkeit und der Fatalität eines Fehlers angepasst werden**
- **Redundanz ist sinnvoll und manchmal sogar notwendig für eine sichere Übermittlung und Speicherung von Daten → natürliche Redundanz ist ungeeignet → muss zuerst eliminiert werden und danach gezielt wieder hinzugefügt werden → maximal mögliche Sicherheit für die zur Verfügung stehende Datenmenge wird erreicht**

Kapitel 12 – Allmächtiger Computer!?

- allgemeine Meinung: Computer kann prinzipiell alles, wird nur durch den Programmierer limitiert
- Affenpuzzle: 2x2 kann man recht schnell lösen → 3x3 braucht mehr als die 16-fache Zeit → 5x5 ohne Computer kaum lösbar
  - es existiert kein Lösungsalgorithmus → Affenpuzzle gehört zu den Problemen der Klasse NP



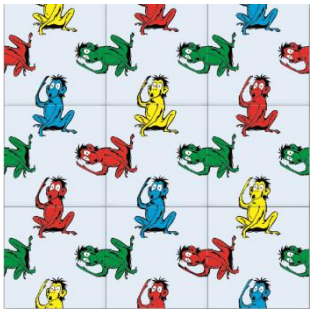
- **Problem skaliert nicht** → zur Lösung einer doppelt so großen Aufgabe benötigt man weit mehr als doppelt so viel Zeit (oft muss die Aufgabengröße quadriert werden) → z.B. auch beim Brückenbau zu finden
- manche Probleme beanspruchen bei Verdopplung der Aufgabengröße unermesslich viel mehr Lösungszeit → ein einziges Puzzleteil kann das Affenpuzzle unlösbar machen, weil man eine absurd lange Zeit zum Lösen bräuchte → nicht praktisch berechenbares Problem
- Gesamtaufwandsberechnung des Affenpuzzles:
  - *n* Puzzleteile können in *n!* Permutationen gelegt werden bis alle zusammenpassen
  - jedes Teil passt nur in eine von vier möglichen Drehrichtungen: $n! * 4^n$ Möglichkeiten
  - im Durchschnitt muss ein Achtel der Anordnungen durchprobiert werden
  - für das Probieren einer Lösung müssen jeweils *n* Karten gelegt werden
  - → $Gesamtaufwand = \frac{n! \times 4^n \times n}{8}$
- → bereits das 4x4 Puzzle ist nicht lösbar, da selbst der Computer eine absurd lange Zeit zum Lösen bräuchte
- kann nach unten korrigiert werden, da man von Hand in Wirklichkeit weitaus weniger Zeit braucht

Kantenlänge Puzzle	Puzzle-teile	Aufwand nach Formel	Dauer beim Kartenlegen per Hand in 0,5 s	Dauer beim Kartenlegen per Computer in 0,01 ns
2	4	3072	25 Minuten	< 1 s
3	9	47563407360	754 Jahre	< 1 s
4	16	4493134915501975142400	59.365 mal Universumsalter	14.247 Jahre
5	25	873203497140136544891232311889100800000		
6	36	878344409141902190815781553750844988457254774772439449600000000		
7	49	96386108756373187 483760857339381516779271342702313473398026579323858050146925281280000000000		
8	64	215886928077746357619490786108395517664283918049359780937304852706570515190751032811176279545169584371759224717312000000000000000		
9	81	16945018342271720384 528887431389738998662893500484164086563712258501015982964610508987534926237096946316995820224179955795664986535133464281958276136960000000000000000000		
10	100	74984723061792395304341371831967689215252543707966986354013192594178 138233002394645831845136345079452453791988578895816786883974949629650018907834782316014907611993173137977852173056997360402432000000000000000000000000		

- **Branch and Bound:** bezieht sich auf das Ausprobieren
  - **Branch:** man verzweigt zu allen potentiellen Lösungen, um die korrekte zu finden
  - **Bound:** man versucht möglichst frühzeitig zu erkennen, dass man beim Ausprobieren einen definitiv falschen Zweig erreicht hat und abbrechen muss
  - Legevorschrift: Startteil ist links oben (alle Möglichkeiten) → danach das Teil rechts davon (nur eines von acht Symbolen passt → Zahl der Züge schränkt sich auf 1/8 der noch vorhandenen Kombinationen) → danach vom Startteil nach unten (wieder nur 1/8)
  - Branch and Bound zielt darauf ab, Situationen zu erzeugen, in denen mit hoher Wahrscheinlichkeit eine Sackgasse erkannt wird
  - Tabelle mit Branch and Bound (viel besser):

Kantenlänge Puzzle	Puzzle-teile	Aufwand mit „Branch and Bound“	Dauer beim Kartenlegen per Hand in 0,5 s	Dauer beim Kartenlegen per Computer in 0,01 ns
2	4	129	1 Minute	<< 1 s
3	9	6773	56 Minuten	<< 1 s
4	16	1965501	11 Tage	<< 1 s
5	25	11385972942	180 Jahre	0,1 s
6	36	4482592451432549	0,005 mal Alter des Universums	12 Minuten
7	49	319714732333844702623086	422.420 mal Alter des Universums	101.380 Jahre
8	64	884602008329553372563942730961 4807		
9	81	1759513182528675008238484492087984287742051 94798		
10	100	426532633057333514229385424028342895195867106697261390996837 5819		

- **Problem des Fliesenlegers: Domino-Problem**
  - es wird eine Fläche aufgelegt, bei der die obere Kante nahtlos an die untere Kante passt, sowie die recht an die linke → dadurch kann z.B. ein 3x3 Feld eine beliebig große Fläche ausfüllen



  - Verfahren, das für beliebige Sätze von Puzzleteilen bestimmt, ob man damit beliebig große Flächen auslegen kann:

```
Versuche, die Kacheln regelkonform aneinander zu legen.

Falls dabei ein Rechteck entsteht, bei dem die obere
und untere sowie die linke und rechte Kante
aneinanderpassen
      → Antwort JA.

Falls trotz Ausreizung aller Möglichkeiten eine Karte
nicht passt
      → Antwort NEIN.
```

  - es gibt spezielle Puzzleteile, bei denen man prinzipiell unendlich weiterpuzzeln kann, aber nie ein Rechteck findet, das sich periodisch wiederholt → ein Programm würde daher nie fertig werden und niemals ein Ergebnis präsentieren können → es handelt sich um eine **unendliche, nichtperiodische Kachelung**
  - ein Computer kann auch nicht beweisen, dass dieses Puzzle unendlich ist, da er mit dem Berechnen nie fertig wäre → **nicht entscheidbares Problem**
- **Terminierung:**
  - Computerprogramme sollen im Normalfall irgendwann mit ihrer Berechnung fertig werden und ein Ergebnis präsentieren → Terminierung
  - nicht terminierende Programme → Abstürze → Computer führt immer wieder den gleichen Code in einer Endlosschleife aus und reagiert nicht mehr auf Eingaben
- **Halteproblem:**
  - ein Programm sollte immer terminieren → man will die Entscheidung „terminiert es irgendwann“ oder „terminiert nicht immer“ automatisch treffen lassen
  - das Programm HÄLTS? Prüft ein anderes Programm, ob dieses immer anhält

Zur Veranschaulichung verwenden wir zwei einfache Programme:

```
PROGRAMM  A
Zeile 10: FALLS X=U GEHE ZU Zeile 10
Zeile 20: ENDE
```

```
PROGRAMM B
Zeile 10: FALLS X=X GEHE ZU Zeile 10
Zeile 20: ENDE
```

Sie sehen, dass Programm A testet, ob X und U identisch sind. Nur falls das zutrifft, springt die bedingte Ausführung wieder zur Zeile 10. Da sich der Computer kein X für ein U vormachen lässt, geht er im Programm einfach weiter und führt Zeile 20 aus, die das Programm beendet. Ein Aufruf von HÄLTS? mit diesem Programm führt also zu:

```
HÄLTS?(PROGRAMM A)=JA
```

Anders sieht die Sache beim zweiten Programm aus. Hier wird in Zeile 10 wieder zu Zeile 10 verzweigt, falls X identisch mit X ist (was natürlich immer stimmt). Damit verliert sich das Programm in einer Endlosschleife.

```
HÄLTS?(PROGRAMM B)=NEIN
```

Der Trick ist nun, unser eigenes Superprogramm in diese bedingte Ausführung einzubauen:

```
PROGRAMM C
Zeile 10: FALLS HÄLTS?(PROGRAMM C)=JA GEHE ZU Zeile 10
Zeile 20: ENDE
```

Wir haben ja bereits festgestellt, dass in Zeile 10 die entscheidende Bedingung steckt: Ist sie wahr, verliert sich das Programm in einer Endlosschleife, ist sie falsch, terminiert es.

- These 1: Programm C hält an:
  - Folge: Bedingung HÄLTS? (Programm C) = JA ist immer wahr → Programm C geht bei Zeile 10 in eine Endlosschleife
  - Konsequenz: Programm C hält nicht an, was These 1 widerspricht, die daher falsch sein muss
- These 2: Programm C hält nicht an:
  - Folge: Bedingung HÄLTS? (Programm C) = JA ist immer falsch → Programm C überspringt Zeile 10 und hält bei Zeile 20 an
  - Konsequenz: Programm C hält an, was These 2 widerspricht, die daher falsch sein muss
- beide Thesen sind falsch → die Voraussetzung, dass wir ein Programm HÄLTS? Entwickeln können ist daher falsch
- **ein Computer kann für eine beliebige Software nicht entscheiden, ob diese terminiert oder nicht**
- **Validierung:** Fehlerfreiheit eines Teilprogrammes muss bewiesen werden

- bei automatischen Programmbeweisen gibt es auch teilautomatische Programmbeweiser: Computer ist ein sehr schnelles und konsequentes Werkzeug → ein teilautomatischer Beweiser kann über lange Strecken Software auf Fehler hin analysieren, muss allerdings irgendwann ein Teilproblem an den Benutzer weitergeben → gesamter Beweis wird durch Zusammenspiel von Intelligenz des Benutzers und Rechenleistung des Computers erzielt