SPAß MIT ALGORITHMEN

Von Listensortieren, über Matrizenmultiplikation bis zur Kryptographie

08.01.2026, Luna Nox Brüntrup

swb

FÜR HEUTE. FÜR MORGEN. FÜR MICH.

Agenda

1. ALLGEMEINES ZU ALGORITHMEN

swb
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

1.1 Was ist ein Algorithmus

Eine eindeutige Folge von Anweisungen, mit endlicher Anzahl an Schritten, um ein Problem zu lösen.

Beispiel schriftliche Multiplikation:

```
 4 2 * 6 8
-----------
     2 4
+      1 2
+      3 2
+        1 6
-----------
   2 8 5 6
```

swb
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

1.2 Eigenschafen von Algorithmen

Auf 1.2 wird später aufgebaut

Endlichkeit

- Endliche Länge der Beschreibung und endlich großer Ressourcenverbrauch

Komplexität

- Aufwand an Rechenzeit und Speicherplatz

Terminiert

- Ein Ergebnis liegt nach endlich vielen Schritten vor

Determiniertheit

- Bei gleicher Eingabe, gleiche Ausgabe; unterschiedliche Zwischenschritte möglich

swb

FÜR HEUTE. FÜR MORGEN. FÜR MICH.

1.2 Eigenschafen von Algorithmen

Deterministisch

- Bei gleicher Eingabe, gleiche Ausgabe und gleiche Abfolge von Schritten

Nicht-Deterministisch

- Bei gleicher Eingabe, nah gleicher Ausgabe, mit mehreren Möglichkeiten in den nächsten Zustand überzugehen

swb

FÜR HEUTE. FÜR MORGEN. FÜR MICH.

1.2 Eigenschafen von Algorithmen

- Veranschaulichung für Nicht-DeterministischeAlgorithmen sind probabilistische Algorithmen
- Code Beispiel: Las Vegas Algorithmus
  - Nicht-Deterministisch determiniert

```
function getRandomInt(min: number, max: number): number {  Show usages
    min = Math.ceil(min);
    max = Math.floor(max);
    return Math.floor(Math.random() * (max - min + 1)) + min;
}

function lasVegas(array: number[]): number {  Show usages
  for (let i: number = 1; i < 1000; i++) {
    const r: number = getRandomInt( min: 0, array.length - 1);
    if (array[r] === 1) {
        console.log("try:" + i)
      return r;
    }
  }
  return -1;
}

//array with the number 1-1000
const array: number[] = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
console.log(lasVegas(array))
```

1.3 Iterative Algorithmen

- Mehrfach auszuführende Arbeitsschritte werden durch Schleifen umgesetz

1.4 Rekursive Algorithmen

- Mehrfach auszuführende Arbeitsschritte werden durch den Selbstaufruf einer Methode umgesetzt
  - Direkte Rekursion: a() ruft a() auf
  - Indirekte Rekursion: a() ruft b() auf, ruft c() auf, ruft a() auf

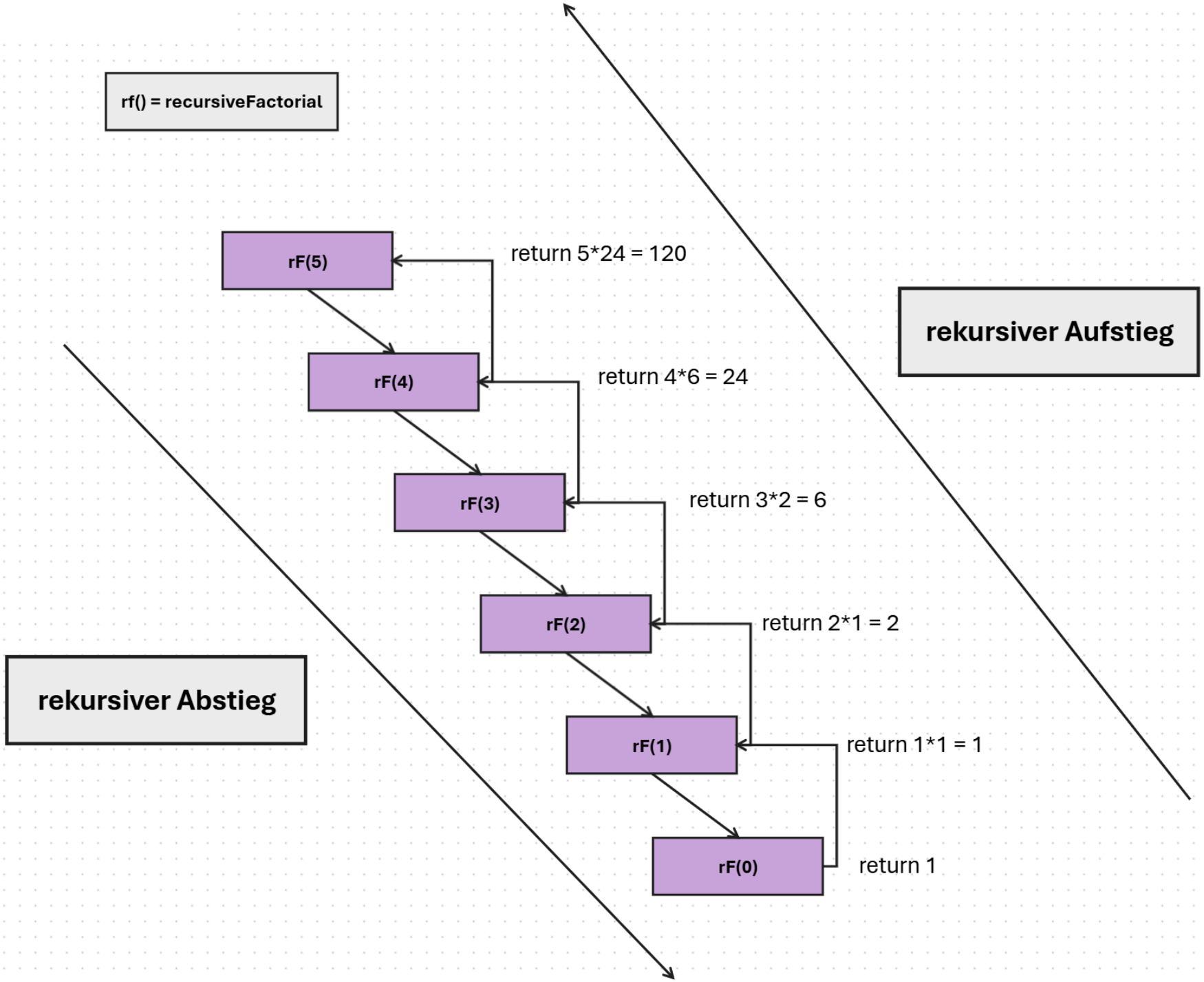
  Besteht aus
  - Base Case (Abbruch Bedingung)
  - Rekursiven Aufruf

swb
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

1.4 Rekursive Algorithmen

Codebeispiel: Berechnung n!

```
3   function iterativeFactorial(n: number): number{  Show usages
4     let answer : number  = 1;
5     for (let iteration: number = 2; iteration <= n; iteration++){
6       answer = answer * iteration;
7     }
8     return answer;
9   }
10
11  function recursiveFactorial(n: number): number{  Show usages
12    //base case
13    if(n === 0){
14      return 1;
15    }
16    //recursive call
17    return n * recursiveFactorial(n -1);
18  }
19
20  //returns 120
21  console.log(recursiveFactorial( n: 5));
22
23  //returns 120
24  console.log(iterativeFactorial( n: 5));
25
```

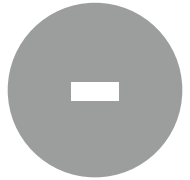


POPQUIZ!!!

Welche Vor- und Nachteile könnte Rekursion haben?

Vor- und Nachteile von Rekursion

- Weniger Zeilen Code
- Lösen komplexer Probleme
- Divide and Conquer wird implementiert
- Effizient für viele Algorithmen



- Stack Overflow
- Viel Arbeitsspeicher wird benötigt
- Funktionsaufrufe und Stack-Management schaden Performance
- Schwer zu debuggen

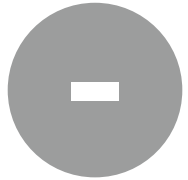
POPQUIZ!!!

swb
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

Welche Vor- und Nachteile könnte Iteration haben?

Vor- und Nachteile von Iteration

- Effizienter Umgang mit Arbeitsspeicher
- Besser Optimierbar für Performance
- Einfach zu debuggen



- Komplexität des Codes
- „off-by-one-error“
- State Management kann zu Fehlern führen

2. O-NOTATION

Auf alles aus diesem Kapitel wird später aufgebaut

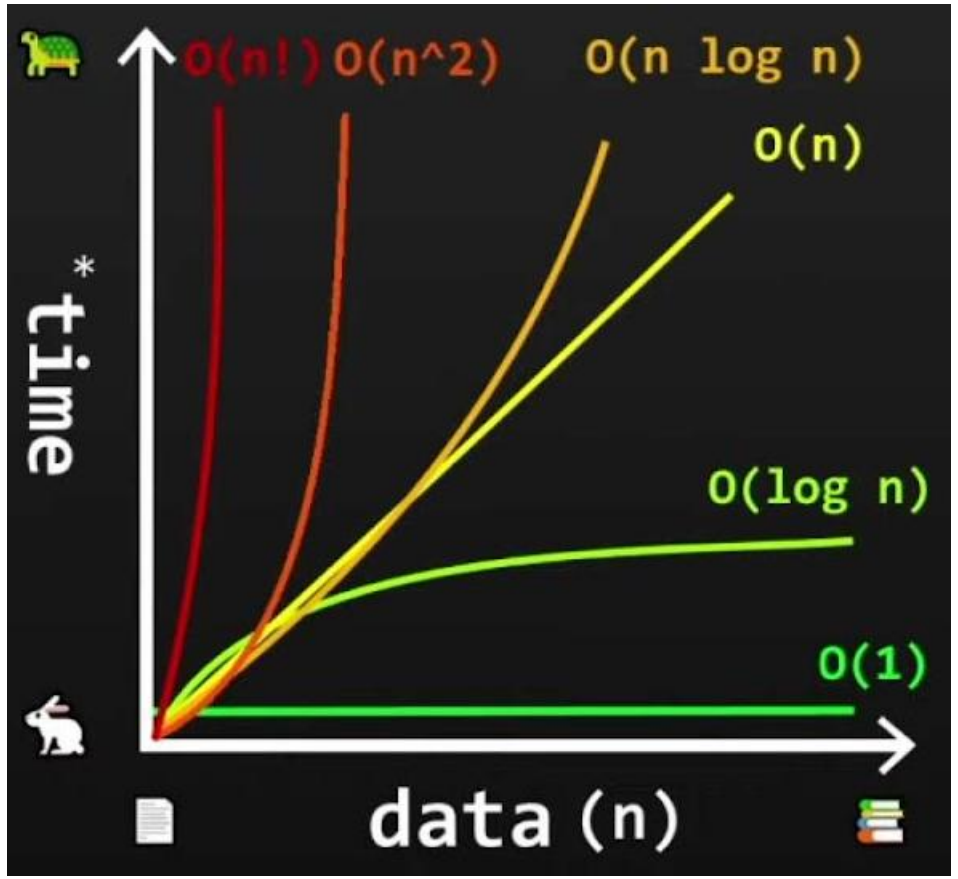
swb

FÜR HEUTE. FÜR MORGEN. FÜR MICH.

2 O-Notation

- Macht die Komplexität von Algorithmen vergleichbar
- Darstellung in Abhängigkeit von der Eingabegröße

- Arten von Komplexität
  - Zeitkomplexität
  - Speicherkomplexität



2.1 polynomielle Algorithmen

- Ein Algorithmus ist polynomiell wenn es eine natürliche Zahl k mit $t \in O(n^k)$
- Polynomialzeit gilt als Grenze zwischen praktisch lösbaren und unlösbaren Problemen
- Sinnvolles Konzept in der theoretischen Informatik

swb
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

2.2 P-NP Problem

P:

- Klasser aller Probleme, die sich auf einer deterministischen sequentiellen Maschine in Polynomialzeit lösen lassen

NP:

- Die Klasse aller Probleme, die sich von einer nichtdeterministischen Maschine in Polynomialzeit lösen lassen

$$P \subseteq NP$$

gilt, da eine deterministische Maschine eine spezielle Form der nichtdeterministischen Maschine ist

Frage: Sind P und NP voneinander echt verschieden? Also ist P ≠ NP

swb
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

3. LISTEN SORTIEREN

swb
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

3.1 Bubblesort

- Vergleicht das aktuelle Element mit dem nächsten
  - Nächstes Element ist größer: Nächstes Element wird zum aktuellem Element
  - Nächstes Element ist kleiner: Position vom aktuellem und nächstem Element werden getauscht; das aktuelle Element bleibt das aktuelle Element
- Im Ergebnis wandert das größte Element an das Ende der Liste

- Durchschnittliche Komplexität: $O(n^2)$
- Bester Fall: $O(n)$
- Schlimmster Fall: $O(n^2)$
- Speicherkomplexität: $O(1)$

```
3   function bubbleSort(array: number[]): void {  Show usages
4     for(let i: number = 0; i < array.length; i++) {
5       for(let j: number = 0; j < ( array.length - i -1 ); j++) {
6         if(array[j] > array[j+1]) {
7           let temp: number = array[j];
8           array[j] = array[j + 1];
9           array[j+1] = temp;
10        }
11      }
12    }
13  }
14
15  //unsorted array with 30 random numbers
16  const arr: number[] = [43 ,8 ,96 ,68 ,17 ,96 ,50 ,77 ,9 ,35 ,2
17  bubbleSort(arr);
18  console.log(arr);
19
```

swb
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

3.2 Quicksort

Nicht-deterministisch determiniert

- Es wird ein zufälliges Element gewählt (Pivot Element)
- Alle kleinere Elemente werde auf die Linke Seite von Pivot geschoben, alle größeren auf die rechte
  - 2 Teilbereiche entstehen
  - Das Pivot Element steht an der richtigen Stelle
- Das gleiche Muster wird rekursiv auf die beiden neuen Teilbereiche angewendet

Parallelisierbar

swb

FÜR HEUTE. FÜR MORGEN. FÜR MICH.

Legende: Pivot kleiner größer sortiert

Ausgangslage:	9	5	3	8	4	2	6	7	1
1	9	5	3	8	4	2	6	7	1
2	5	3	4	2	1	6	9	8	7
1	5	3	4	2	1	6	9	8	7
2	2	1	3	5	4	6	7	8	9
1	2	1	3	5	4	6	7	8	9
2	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	1	2	3	4	5	6	7	8	9
Fertig:	1	2	3	4	5	6	7	8	9

Funktionsweise:

1.: auswählen eines zufälligen Pivot Elementes
2.: kleinere Elemente werden an den neuen Array der linken angehangen; größere auf den der rechten Seite
3.: rekursiver Aufruf

- Durchschnittliche Komplexität: O(n * log(n))
- Bester Fall: O(n * log(n))
- Schlimmster Fall: $O(n^2)$
- Speicherkomplexität: O(n)

```typescript
function quicksort(arr: number[]): number[] {  Show usages
  //Base case
  if (arr.length <= 1) {
    return arr;
  }
  const randomIndex: number = Math.floor(Math.random() * arr.length);
  //zufälliges Pivot Element wird gewählt
  const pivot: number = arr[randomIndex];
  const left: number[] = [];
  const right: number[] = [];

  //kleiner als Pivot -> links vom Pivot einsortieren; größer als Pivot -> rechts vom Pivot einsortieren
  for (let i: number = 0; i < arr.length; i++) {
    if (i === randomIndex) continue;
    if (arr[i] < pivot) {
      left.push(arr[i]);
    } else {
      right.push(arr[i]);
    }
  }
  //kombiniert drei arrays zu einem
  return [...quicksort(left), pivot, ...quicksort(right)];
}

const unsortedArray: number[] = [3, 6, 8, 10, 1, 2, 1];
const sortedArray: number[] = quicksort(unsortedArray);
console.log(sortedArray);
```

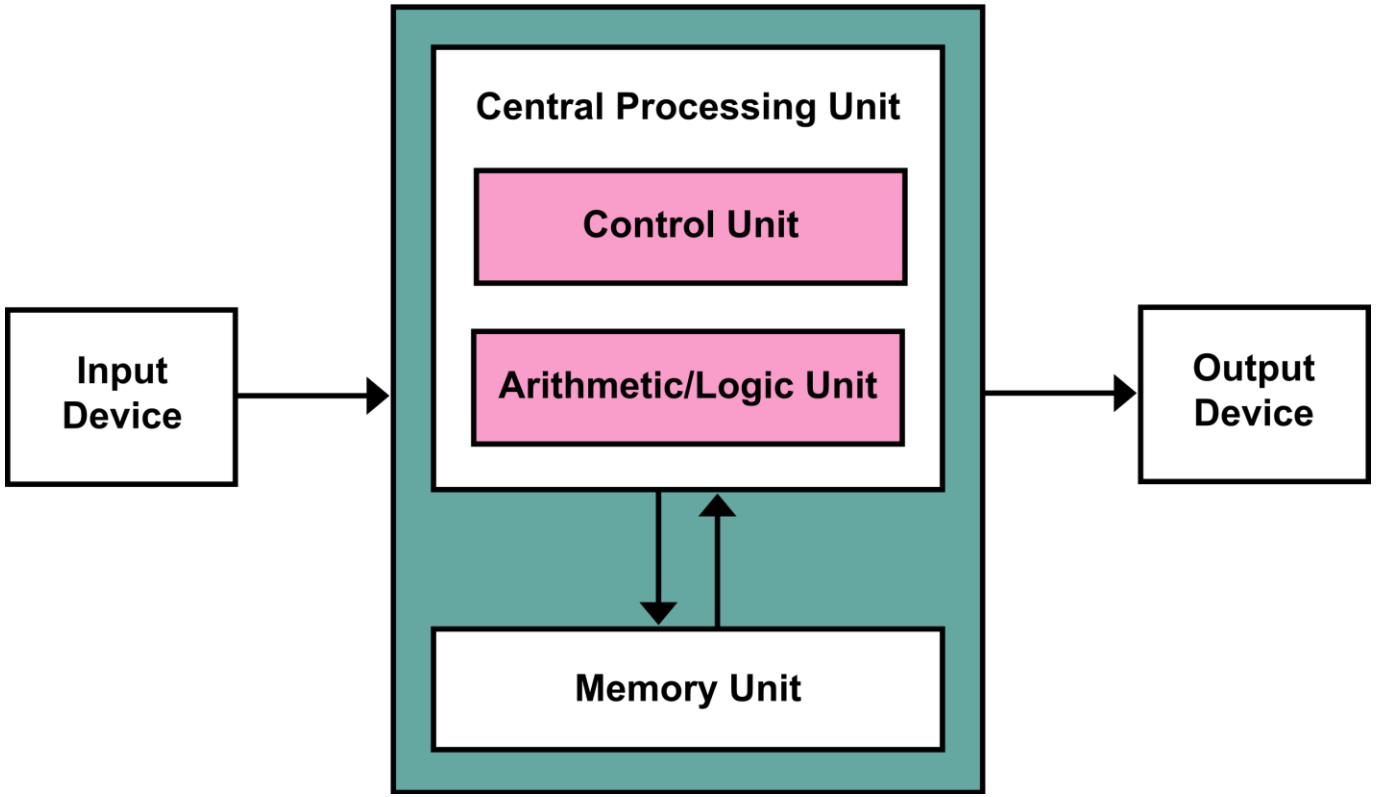
swb
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

4. CPU VS GPU

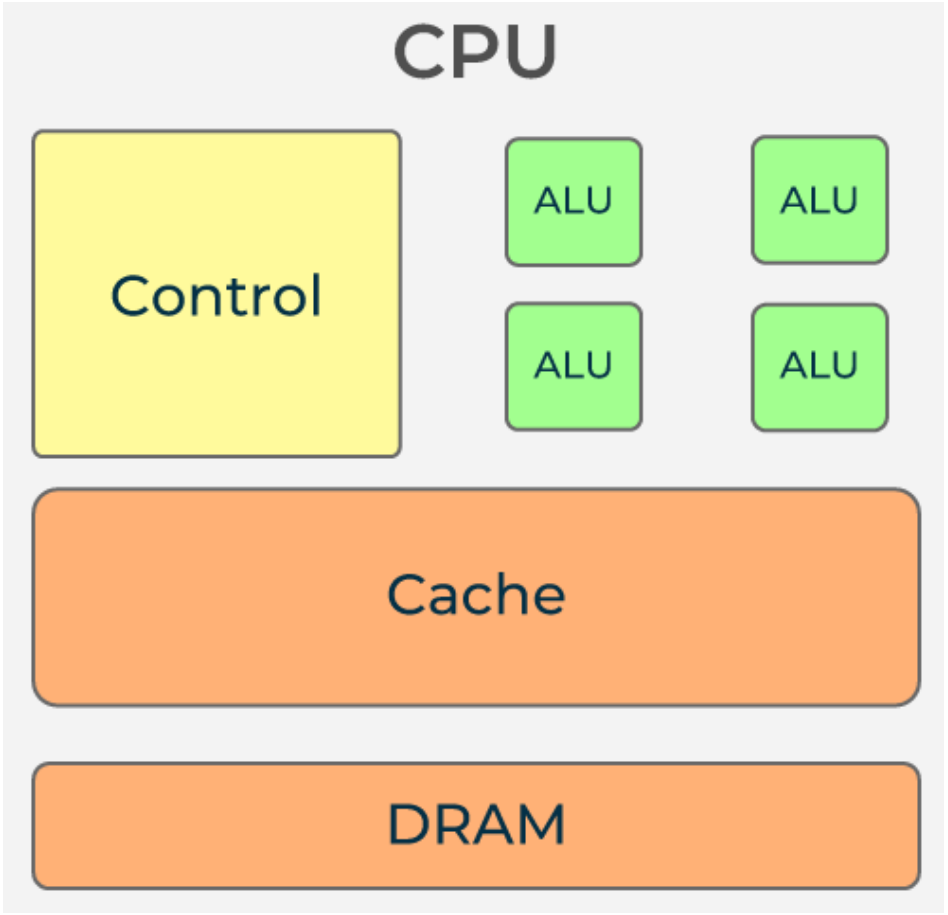
swb

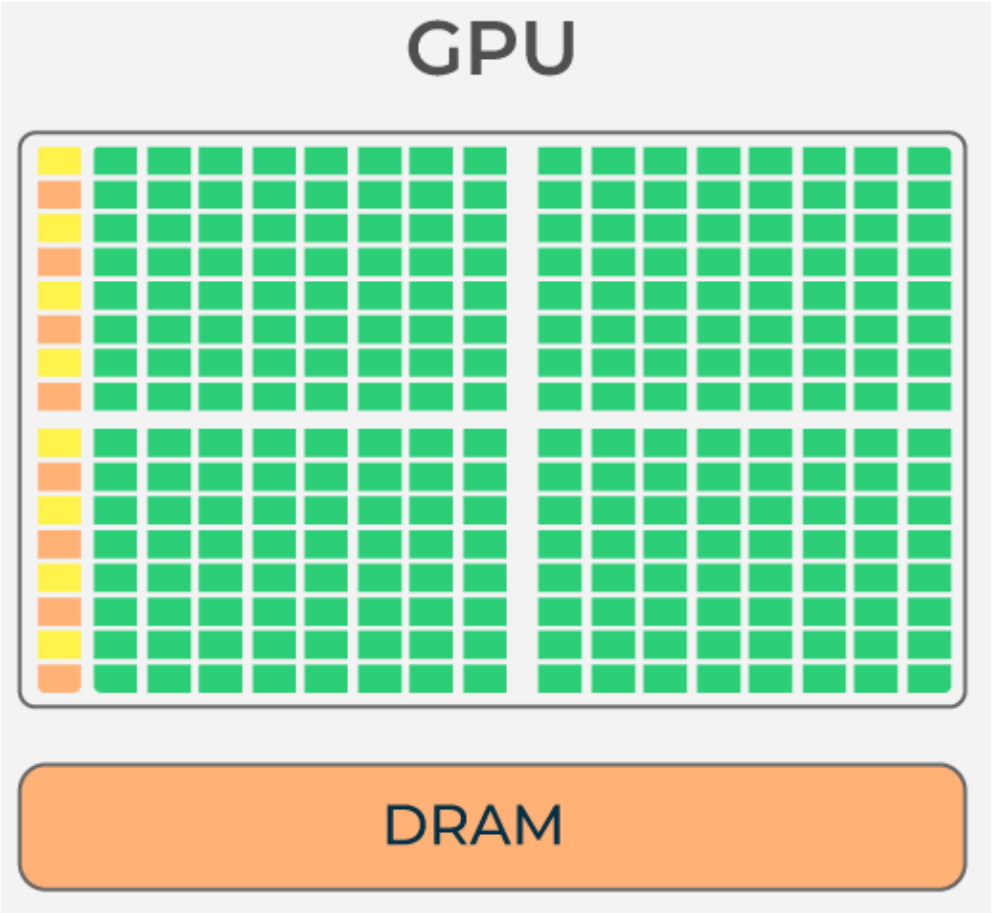
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

4.1 CPU: Von Neumann Architektur



4.2 Unterschiede zwischen CPU und GPU





swb
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

5. MATRIZEN RECHNUNG

swb
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

5.1 Was sind Matrizen

```
const arr: number[][] = [
[22, 24, 19, 23, 25, 20, 21],
[18, 22, 21, 25, 26, 24, 19],
[20, 23, 22, 27, 24, 22, 18],
[19, 21, 24, 26, 25, 23, 20],
[23, 25, 27, 26, 22, 24, 21]
];
```

Eine Matrix ist als 2D-Array oder Tabelle darstellbar

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

Matrix mit den Dimensionen m*n

swb
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

5.2 Matrizen Multiplikation

Wichtigste Matrizen Rechnung bei Computern

$$\begin{bmatrix} a_1 & b_1 \\ c_1 & d_1 \end{bmatrix} \times \begin{bmatrix} a_2 & b_2 \\ c_2 & d_2 \end{bmatrix} = \begin{bmatrix} a_1a_2+b_1c_2 & a_1b_2+b_1d_2 \\ c_1a_2+d_1c_2 & c_1b_2+d_1d_2 \end{bmatrix}$$

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \times \begin{bmatrix} j & k & l \\ m & n & o \\ p & q & r \end{bmatrix} = \begin{bmatrix} (aj + bm + cp) & (ak + bn + cq) & (al + bo + cr) \\ (dj + em + fp) & (dk + en + fq) & (dl + eo + fr) \\ (gj + hm + ip) & (gk + hn + iq) & (gl + ho + ir) \end{bmatrix}$$

swb

FÜR HEUTE. FÜR MORGEN. FÜR MICH.

5.2 Matrizen Multiplikation

$$\begin{pmatrix} 4 & 3 & 2 \\ 1 & 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix}$$

$$= \begin{pmatrix} 4 \cdot 2 + 3 \cdot 3 + 2 \cdot 4 \\ 1 \cdot 2 + 2 \cdot 3 + 3 \cdot 4 \end{pmatrix}$$

$$= \begin{pmatrix} 25 \\ 20 \end{pmatrix}$$

swb
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

5.3 Anwendungsbeispiel Transformation

Problem: Objekt im dreidimensionalen Raum soll verschoben, rotiert und/oder skaliert werden
Lösung: Transformationsmatrix

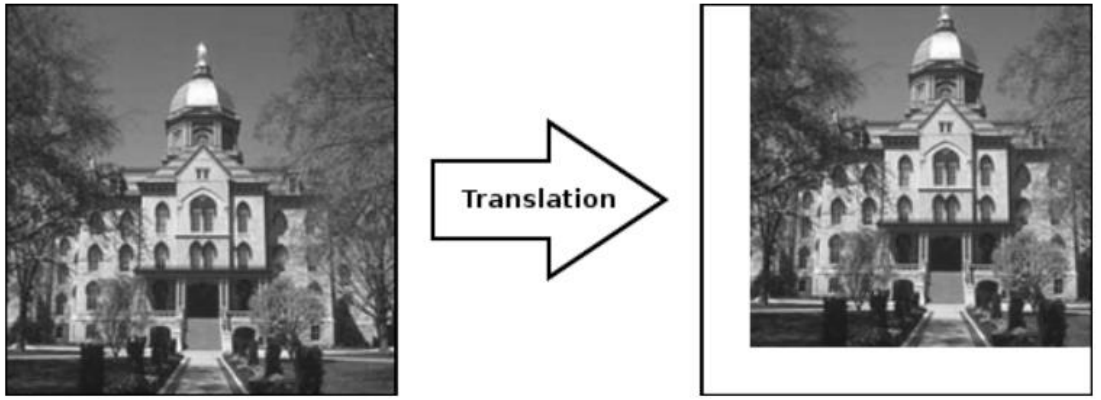
$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \times \begin{bmatrix} S_x & M_{12} & M_{23} & T_x \\ M_{21} & S_y & M_{23} & T_y \\ M_{32} & M_{32} & S_z & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- erste Matrix: Orginalkoordinaten
- zweite Matrix: transformationsmatrix
- T: Position
- M: Rotation
- S: Skalierung
- Zahlen: fixed -> nur 12 Freiheitsgrade

swb
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

verschieben

$$P' = P + d$$



$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

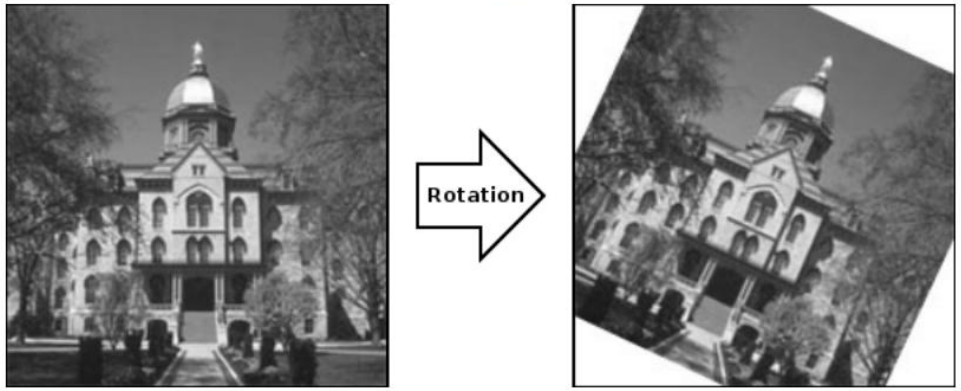
resulting coordinate — 3d translation matrix — original coordinate

swb
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

Rotation um die z-Achse

$$x' = x \cos(t) - y \sin(t)$$
$$y' = x \sin(t) + y \cos(t)$$
$$z' = z$$

$$P' = R_z(t)P$$



$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

resulting coordinate — 3d rotation matrix in Z — original coordinate

Rotation um die x-Achse

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

resulting coordinate — 3d rotation matrix in X — original coordinate

Rotation um die y-Achse

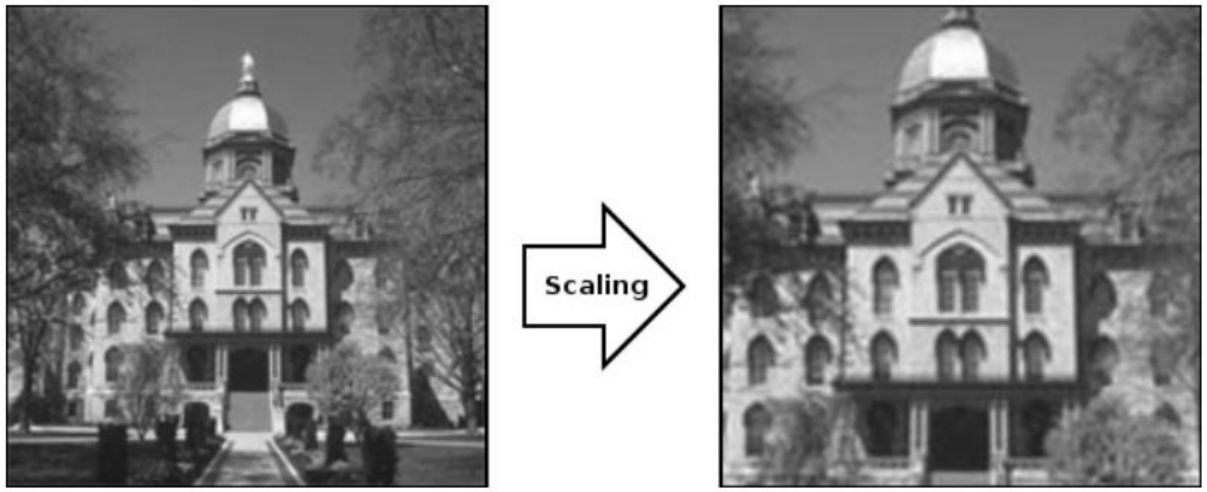
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

resulting coordinate — 3d rotation matrix in Y — original coordinate

swb
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

Skalierung

$$P' = SP$$



$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

resulting coordinate — 3d scaling matrix — original coordinate

5.3 Anwendungsbeispiel Transformation

- Die beschriebene Multiplikationen sind 4x4x4 Matrizen Multiplikation

Komplexität von 4x4x4 Matrizen Multiplikation

- Normal: 64 skalare Multiplikationen; $O(n^3)$
- Strassmann Algorithmus (von 1969): 49 skalare Multiplikationen; $O(n^{\log_2(7)})$
- Algorithmus von Googles Alpha Evolve (von 2025): 48 Skalare Multiplikation

swb
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

5.4 Anwendungsbeispiel Maschinelles lernen

- Riesige Datenmengen müssen gleichzeitig verarbeitet werden

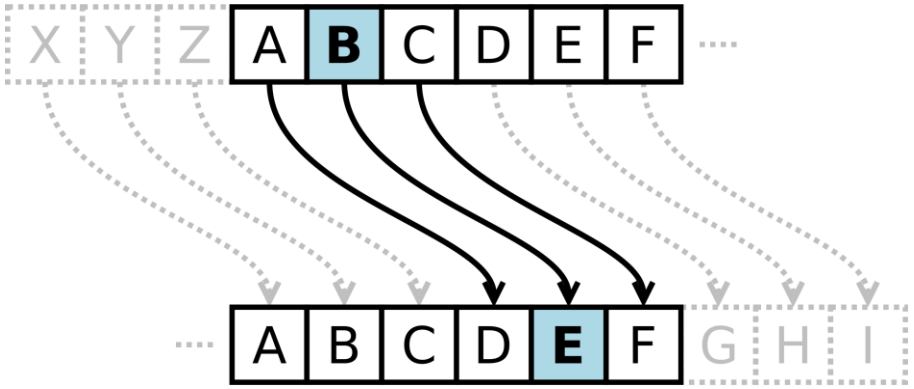
- Neuronale Netze: gewichtete Verbindungen zwischen Knoten
  - Darstellbar als Matrix
  - Daten werden als Vektoren oder Matrizen eingespeist
  - Auf jeder Ebene findet Matrix Multiplikation statt
    - lernen passiert durch das Anpassen der Gewichtung durch Matrix Rechnungen, bis das Modell einen verlässlichen Output hat

6. VERSCHLÜSSELUNG

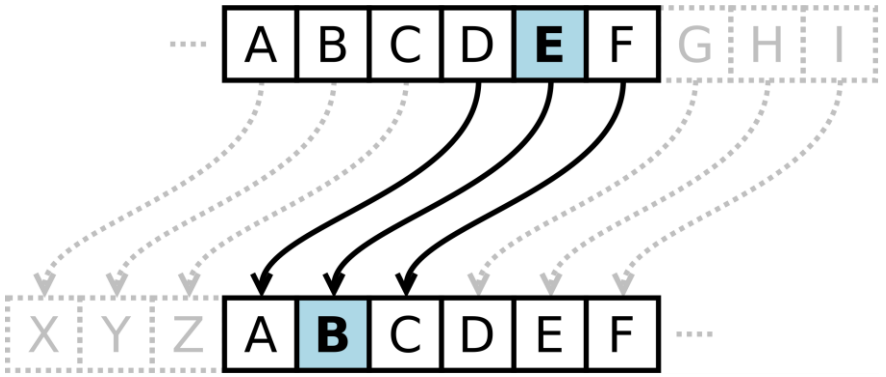
swb
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

6.1 Caeser Verschlüsselung

- Einfaches symmetrisches Verschlüsselungsverfahren
  - Symmetrisch: beide Teilnehmer verwenden den gleichen Schlüssel



Verschlüsselung mit dem Schlüssel C,
also eine Verschiebung um 3 Zeichen



Entschlüsselung mit dem Schlüssel C

Klar: 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
Geheim: D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

swb
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

6.1 Caeser Verschlüsselung

Mathematische Darstellung bei einem Alphabet mit 26 Zeichen :

$\text{Verschlüsseln}_K(P) = (P+K)\%26$

$\text{Entschlüsseln}_K(C) = (C-K)\%26$

P: Klartextbuchstabe

K: Verschiebung

C: Geheimbuchstaben

swb

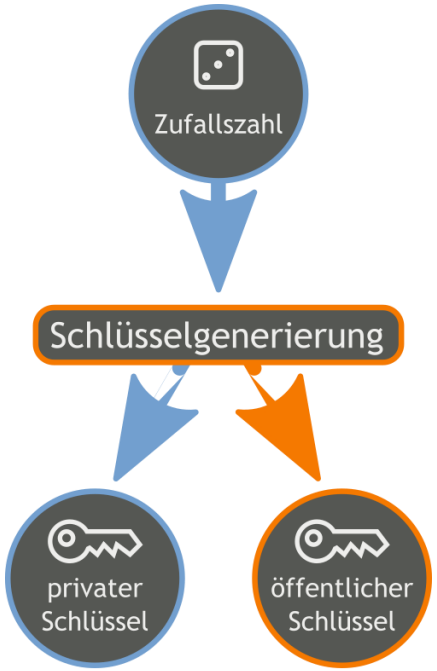
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

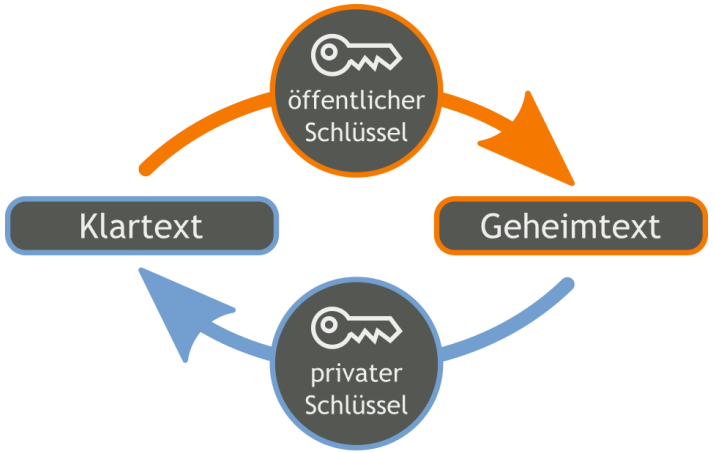
7. PUBLIC-KEY-VERSCHLÜSSELUNGSVERFAHREN

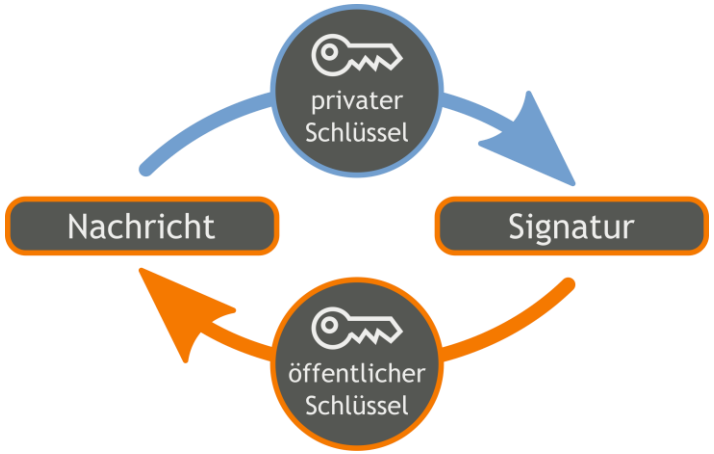
swb
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

7. Public-Key-Verschlüsselungsverfahren

Asymmetrisches Verschlüsselungsverfahren







swb
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

7.1 RSA Verschlüsselung

Rivest–**S**hamir–**A**dleman

- Erstveröffentlichung 1977
- Deterministischer Algorithmus
- Verschlüsselung basiert auf auf Einwegpermutation mit Falltür (engl. trapdoor one-way permutation, kurz TOWP)

7.1.1 Einwegfunktionen

Berechnung der Funktion einfach, Umkehren der Funktion sehr schwer

- Beispiel
- Multiplizieren zweier großer Primzahlen: 52067 * 90271 = 4.700.140.157
  - Einfach: in Polynomialzeit berechenbar
- Aus dem Ergebnis 4.700.140.157 die Primfaktoren zu berechen:
  - Schwer: kein probabilistischer Algorithmus kann die Primfaktoren in Polynomialzeit berechnen

- Problem:
  - Existenz von Einwegfunktionen nicht bewiesen
  - Beweis von Einwegfunktionen ist ein Beweise für das P-NP-Problem

7.1.2 Einwegfunktionen mit Falltür

Berechnung der Funktion einfach, Umkehren der Funktion auch einfach, wenn man eine gewisse Zusatzinformation besitzt

Beispiel:

- Multiplizieren zweier großer Primzahlen hat das Ergebnis 4.700.140.157
- Zusatzinformation: ein Primfaktor ist 52067

> 4.700.140.157/ 52067 = 90271

Ergebnis ist zweiter Primfaktor

swb
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

7.2 Beispiel für RSA Verschlüsselung

Öffentlicher Schlüssel: (e, N)

Privater Schlüssel: (d, N)

Öffentlicher Schlüssel: (5, 14)

Privater Schlüssel: (11, 14)

Klartext m: „B“ -> 2

Verschlüsseln:

M^e % N = c

2^5 % 14 = 32 % 14 = 4

Geheimtext c: „D“ -> 4

Entschlüsseln:

C^d % N = m

4^{11} % 14 = 4194304 % 14 = 2

Klartext m: „B“ -> 2

swb

FÜR HEUTE. FÜR MORGEN. FÜR MICH.

7.3 Beispiel für RSA Schlüsselgenerierung

1.:

Wähle zwei Primzahlen p und q

p = 2, q = 7

2.:

Berechne N = p * q

N =14

swb

FÜR HEUTE. FÜR MORGEN. FÜR MICH.

7.3 Beispiel für RSA Schlüsselgenerierung

3.:

Berechne die Anzahl an Teilfremden (coprimes) von 14

[1, ~~2~~, 3, ~~4~~, 5, ~~6~~, ~~7~~, ~~8~~, 9, ~~10~~, 11, ~~12~~, 13, ~~14~~]

Beispiel:

Primfaktoren von 14 = 2 * 7

Primfaktoren von 6 = 2 * 3

Folgerung: 2 und 7 können nicht Teilfremd sein, da sie die Primfaktoren von 14 sind. Und 6 kann nicht Teilfremd sein, weil einer der Primfaktoren sich mit 14 geteilt wird

Einfacher: Phi Funktion von N: $\Phi(N) = (p-1)*(q-1)$

$\Phi(N) = 6$

swb

FÜR HEUTE. FÜR MORGEN. FÜR MICH.

7.3 Beispiel für RSA Schlüsselgenerierung

4.:

Wähle Zahl e

Bedingung 1: $1 < e < \Phi(N)$

Bedingung 2: Teilfremd mit N und $\Phi(N)$

Bedingung 1 erfüllen: 2, 3, 4, 5

2 und 3 sind Primfaktoren von 6; 4 teilt sich Primfaktoren mit 6

Bedingung 2 erfüllt: 5

swb

FÜR HEUTE. FÜR MORGEN. FÜR MICH.

7.3 Beispiel für RSA Schlüsselgenerierung

5.:

Wähle d

Bedingung: d * e % Φ(N) = 1

5d % 6 = 1

5d könnte 5, 10, 15, 20, 25, 30 , 35

Modulo 6: 5, 4, 3, 2, 1, 0, 5

Überraschung! 5*6 %6 = 0 = 5*12 % 6

d = 11

Weitere mögliche Werte für d: 5, 17, 23, 29

swb

FÜR HEUTE. FÜR MORGEN. FÜR MICH.

8. AUFGABEN

swb
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

Aufgabe 1

Erinnerst du dich an die Aufgabe Fibonacci-Folge? Wie wäre es mit einer Tribonacci Folge?

Bei einer Tribonacci Folge wird das nächste Element aus den drei vorherigen berechnet.

Schreibe nun eine iterative und eine rekursive Methode, die mit einem Array [0, 1, 1] startet und n-te Tribonacci Zahl zurück gibt.

Beispiel für die 4. Fibonacci Zahl: 0 + 1 + 1 = 2
Beispiel für die 5. Fibonacci Zahl: 1 + 1 + 2 = 4

Aufgabe 2

Bubblesort ist unfassbar ineffizient. Deine Aufgabe ist es nun, Bubblesort minimal besser zu machen. Identifiziere dazu mindestens ein Problem von Bubblesort. Es kann danach die gleiche Zeitkomplexität haben.

Eine ausgedruckte Version bitte über Brieftaube abgeben

swb
FÜR HEUTE. FÜR MORGEN. FÜR MICH.

Aufgabe 3

Du solltest nie in deiner Berufslaufbahn Kryptographie-Verfahren selber schreiben, sondern lieber auf bereits etablierte Systeme zurückgreifen, *aber* um dein Verständnis von der RSA Verschlüsselung zu stärken, sollst folgende Methoden (für kleine natürliche Zahlen) implementieren:

Zunächst errechne per Hand einen öffentlichen und einen privaten Key

Methode 1: bekommt den öffentlichen Key, verschlüsselt damit ein einzelnes Zeichen der ASCII Tabelle, und gibt dieses zurück.

Methode 2: bekommt das verschlüsselte ASCII Zeichen und den privaten Key. Damit soll das ursprüngliche ASCII Zeichen ermittelt und zurückgegeben werden.

Danke, dass ihr Teil dieser Reise wart

swb

FÜR HEUTE. FÜR MORGEN. FÜR MICH.