

# Salsa20/20

Gruppe 215

A500

David Csida • Georgios Merezas • Fabian Degen

---

# Gliederung

---

- Was ist Salsa20/20 ?
  - Wie funktioniert eine Stromchiffre ?
  - Salsa20-Crypt
  - Salsa20-Core
    - Was bedeuten die Werte an der Diagonalen ?
- Korrektheit
- Performanz
  - Implementierungsunterschiede
    - Optimierung der Transponierung
    - SIMD
- Zusammenfassung

# Was ist Salsa20/20 ?

---

- ein kryptographisches Verfahren zur Ver - und Entschlüsselung von Daten
- Stromchiffre basierend auf einem *Add-Rotate-XOR-Schema*
- verwendet eine 4x4-Matrix bestehend aus vorzeichenlosen Little-Endian-Ganzzahlen
- Matrix wird erzeugt durch ARX-Operationen auf Startwerten (siehe Salsa20-Kern)
- Ersteller: David J. Bernstein

# Wie funktioniert eine Stromchiffre ?

---

- symmetrischer, kryptographischer Algorithmus
- nimmt Klartext und Schlüsselstrom entgegen
- führt bitweise XOR-Operation zwischen Klartext und Schlüsselstrom aus => Geheimtext
- selber Schlüsselstrom und Geheimtext als Eingabe => Klartext

Klartext	Schlüsselstrom	Geheimtext	Geheimtext	Schlüsselstrom	Klartext
0	0	0	0	0	0
0	1	1	1	1	0
1	0	1	1	0	1
1	1	0	0	1	1

# Salsa20-Crypt

---

- nimmt Eingabe des Verfahrens (256bit Key, 64bit Nonce, Nachricht beliebiger Länge)
- ruft die Kernfunktion auf und bekommt ein 64 Byte Schlüsselstrom
- führt die Verschlüsselung von 64 Bytes der Nachricht mit dem Strom durch XOR Operationen aus
- wiederholt die letzten zwei Schritte bis zum Ende der Nachricht

# Salsa20-Core

---

- Funktion die vorhin beschriebene 4x4 Matrix erzeugt
- Input: Startzustand der 4x4 Matrix
- Output: 64 Byte Schlüsselstrom zum verschlüsseln/entschlüsseln
- besteht aus 20 Runden
- eine Runde beinhaltet 4 Viertelrunden
- eine Viertelrunde besteht aus je 4 Add, Rotate und XOR Operationen
- am Ende jeder Runde wird die Matrix transponiert

# Salsa20-Core

---

- Startzustand der Matrix

$$\begin{pmatrix} 0x61707865 & K_0 & K_1 & K_2 \\ K_3 & 0x3320646e & N_0 & N_1 \\ C_0 & C_1 & 0x79622d32 & K_4 \\ K_5 & K_6 & K_7 & 0x6b206574 \end{pmatrix}$$

- K -> 256bit Key
- N -> 64bit Nonce
- C -> 64bit Counter

# Was bedeuten die Werte an der Diagonalen ?

---

- durch diagonale Verschiebungen und Bitrotationen der Elementen der Matrix können Teile von sie wiederhergestellt werden
- Diagonalwerte stellen sicher, dass dies nicht passiert
- somit haben zwei verschiedene Key-Nonce-Eingaben niemals dieselbe Matrix
- nicht alle Werte sind dafür geeignet



# Korrektheit

---

- Testen der Core Funktion:
  - Referenzimplementierung von David J. Bernstein
  - aufrufen der Funktion mit eigens erstellten und zufällig generierten Eingaben
  - vergleichen von Output mit Output der Referenzimplementierung
- Testen der Crypt Funktion:
  - zweimaliges Aufrufen der Crypt Funktion
  - beim zweiten Aufruf Geheimtext als input
  - Output nach zweitem Aufruf muss wieder Klartext sein
- durch automatisierte Tests beliebig oft Kern und Crypt testen

# Performanz

---

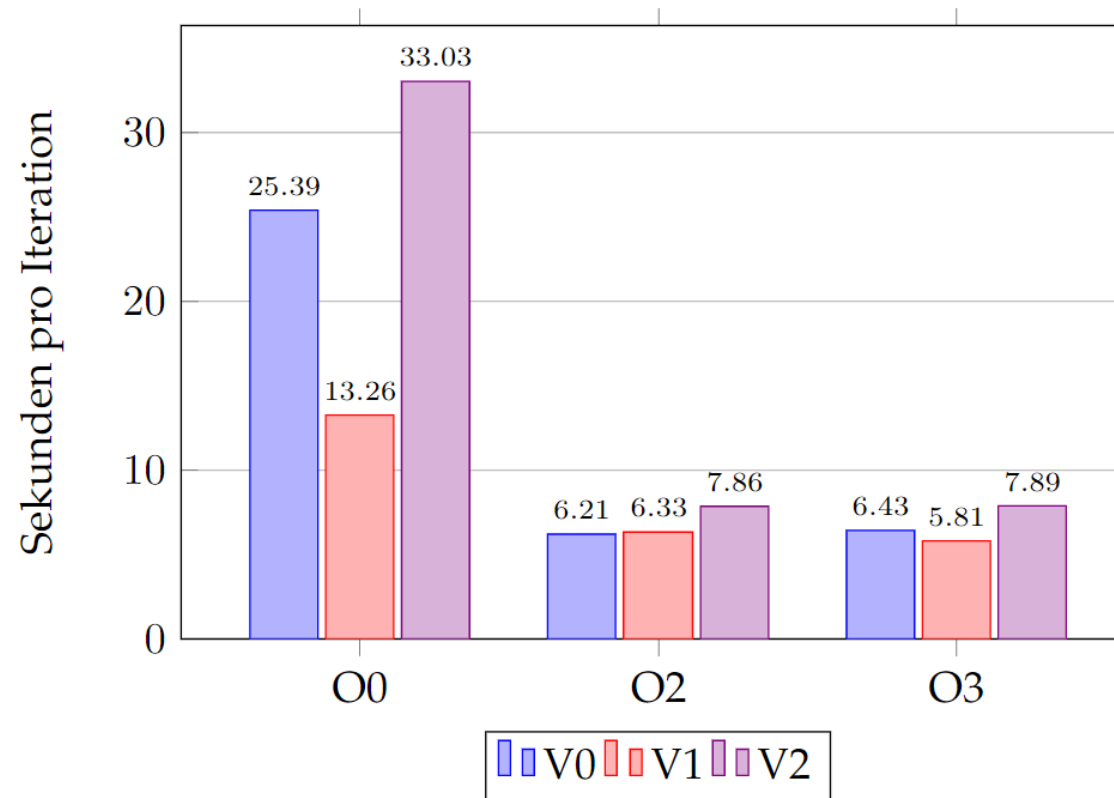


Abbildung 1: Laufzeiten der Implementierungen

# Implementierungsunterschiede

---

- drei Implementierungen: V0, V1, V2
- V0: 1:1 Implementierung der Aufgabenstellung
- V1: Optimierung der Transponierung und andere Berechnung der Matrixindizes
- V2: V1 mit zusätzlicher Vektorisierung der XOR- und Rotate-Operationen

# Optimierung der Transponierung

---

- 10 Runden anstatt 20
- in jeder Runde 8 Viertelrunden anstatt 4
- die ersten 4 Viertelrunden normal
- die zweiten 4 Viertelrunden indizieren die Matrix so, als wäre sie transponiert
- => loop unrolling + Einsparen des Transponierens
  - => Signifikanter Speedup

# SIMD

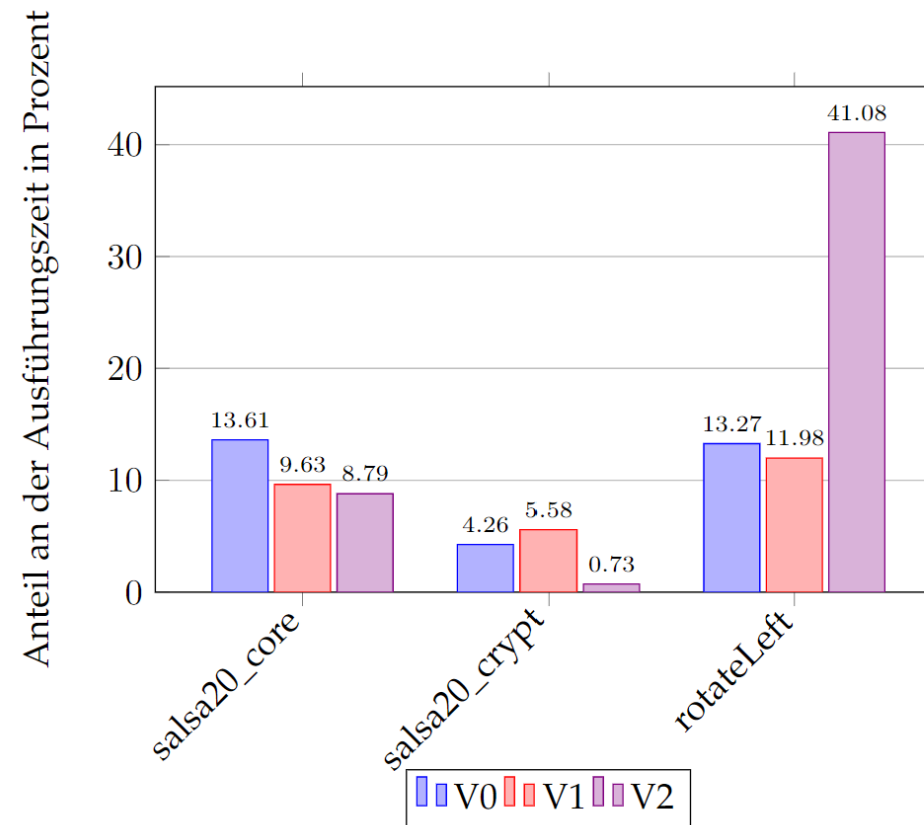


Abbildung 2: Relative Laufzeiten der Implementierungen

# SIMD

---

- Vektorisierung der XOR-Operation zwischen Schlüsselstrom und Klartext
- Vektorisierung der Rotate-Operation
- langsamer als V1 durch Overhead von Laden und Schreiben
  - => besonders gravierend bei rotateLeft (siehe perf-Diagramm)
- durch klügere und weniger Lade - und Schreiboperationen potentiell höherer Speedup möglich
  - => V2 auf O2 und O3 deutlich schneller, da Compiler vermutlich genau das getan hat

# Zusammenfassung

---

- Ziel: Implementierung des Salsa20/20-Verschlüsselungsverfahrens
- Verifizierung der Korrektheit der Implementierung durch automatisierte und manuelle Tests
- Optimierung der Hauptimplementierung durch Einsparen des Transponierens
- Speedup durch Vektorisierung unerfolgreich
- mögliche Verbesserungen: TUI, GUI, schnellere I/O-Operationen, mehr Eingabeformate