ASP Team 117 – Aufgabe 502: eXtended Tiny Encryption Algrotihm

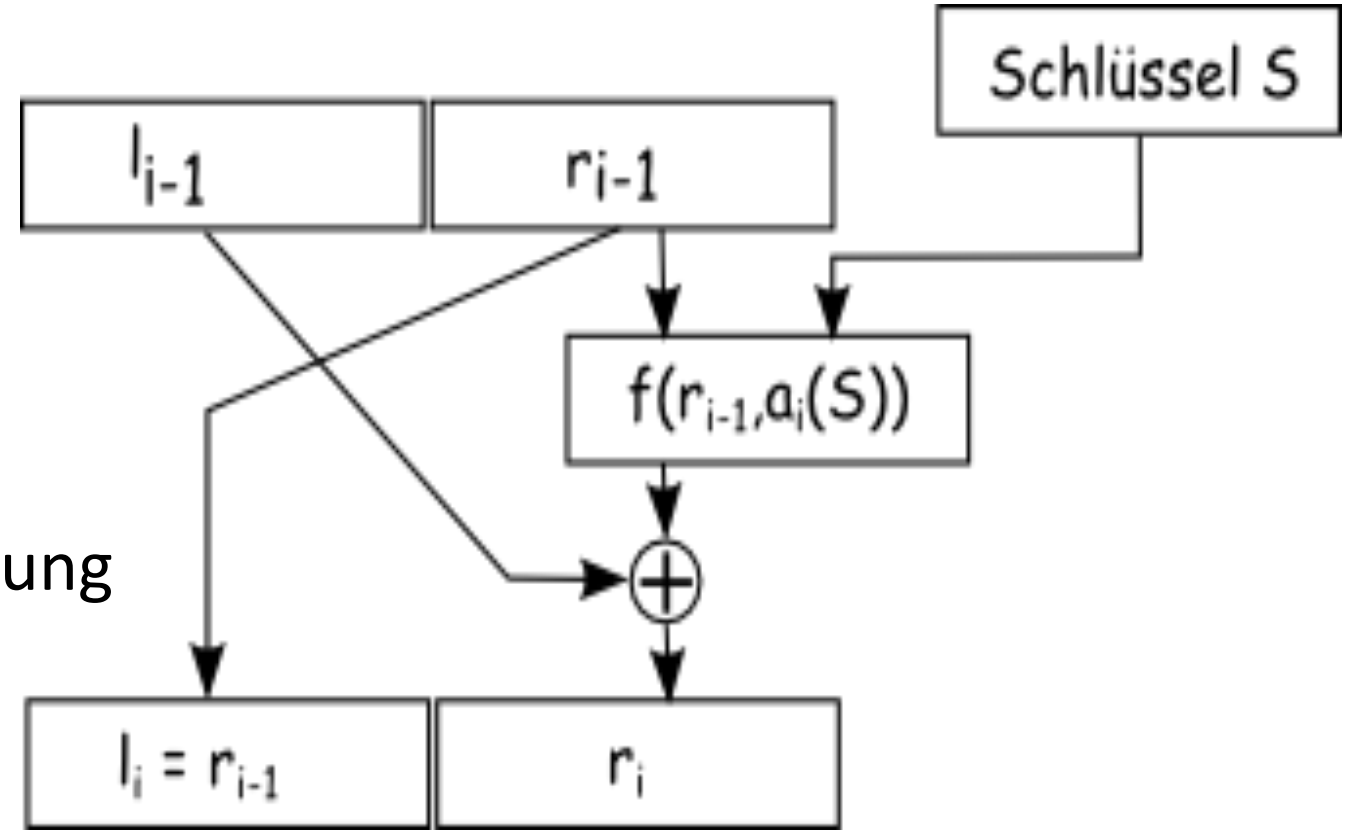
Teammitglieder:

- Linfeng Guo
- Barış Özakay
- Hazar Gönenç

Inhaltsverzeichnis

Feistelchiffre

- Eine symmetrische Verschlüsselungsstruktur
- Zwei gleichgroße Blocks
- Jede Runde ein Block
- Für die Ver– und Entschlüsselung nur ein Key



Eine Verschlüsselungsrunde der abstrakten Feistelchiffre

Abbildung 1: www.hsg-kl.de/faecher/inf/krypto/feistel/index.php
(Zugriffsdatum: 23.01.2022)

Feistelchiffre

B	E	I	S	P	I	E	L

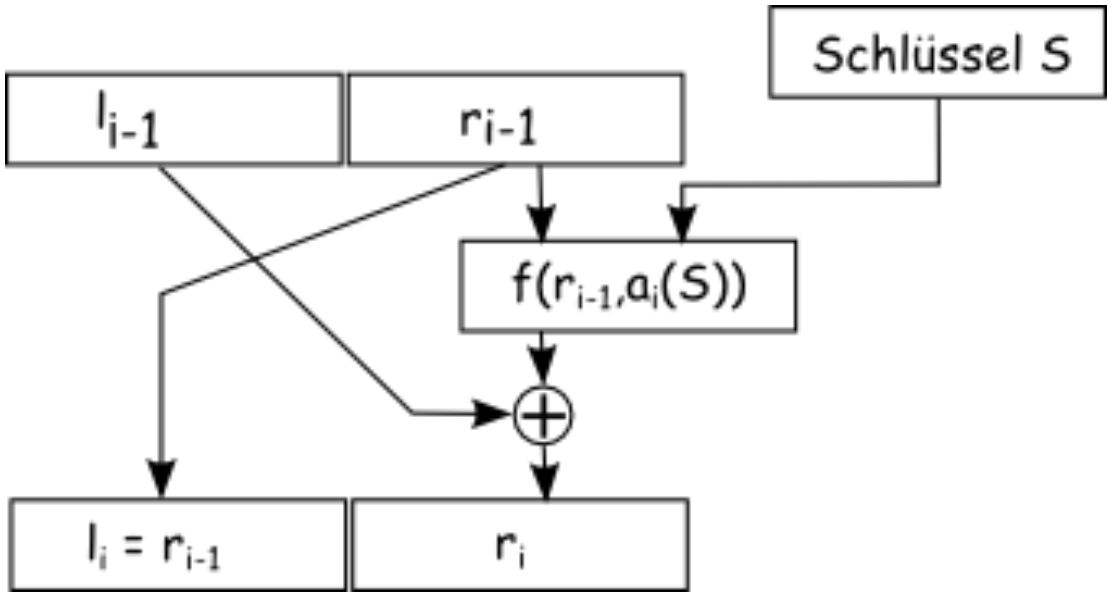
42	45	49	53	50	49	45	4C

42	45	49	53

50	49	45	4C

41	53	50	41	53	50	41	53	50	41	53	50	41	53	50	41

41	53	50	41



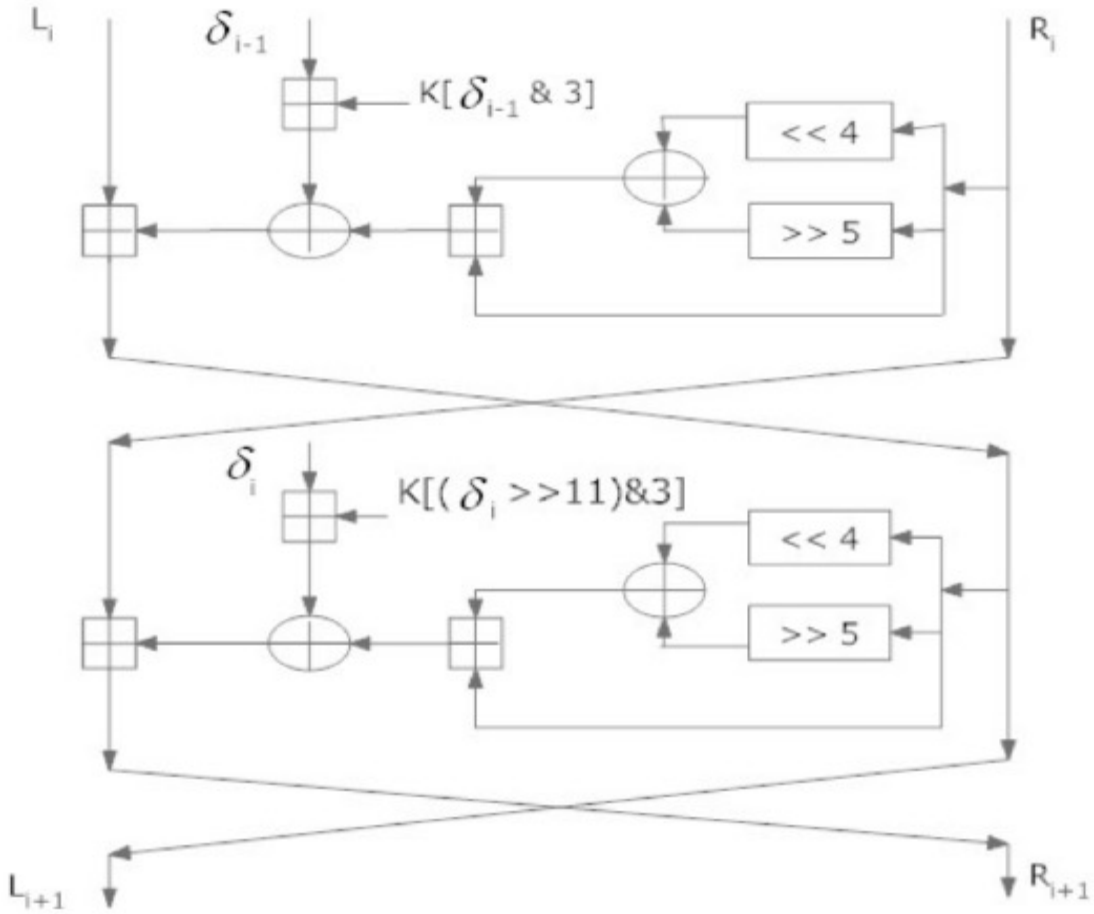
- 64-Bit Message
- Message geteilt in 2 * 4 Byte Blocks
- 128-Bit Schlüssel
- Schlüssel geteilt in 4 * 4 Byte Blocks

XTEA

- Auf Feistelchiffre basiert
- Doppelte Verschlüsselung in jeder Runde
- Eine weitere Variable s und die magische Zahl δ

$$\delta = \lfloor (\sqrt{5} - 1) \cdot 2^{31} \rfloor$$

- Bits von R_i und L_i geshiftet



Eine Verschlüsselungsrunde des XTEAs

Abbildung 2: www.researchgate.net/figure/Fig-1Two-rounds-one-cycle-of-XTEA-III-EXISTING-ATTACKS-ON-XTEA-There-exist_fig2_229480139 (Zugriffsdatum: 27.01.2022)

Beispiel für XTEA

Ein Teil der ersten Verschlüsselungsrunde des XTEAs

A
41

$$((((V2 \ll 4) \oplus (V2 \gg 5)) + V2) \oplus (s + K[s\&\ 3])) + V1 =$$

59	51	7D	CA

Abbildung 3: Alle Schritte des ersten Teils vom XTEA

Padding

- Wie werden Nachrichten deren Länge nicht einem Block betragen verschlüsselt?
- PKCS#7
- Nachrichten mit Padding-Bytes ausschreiben und erst beim Entschlüsseln Padding-Bytes strippen

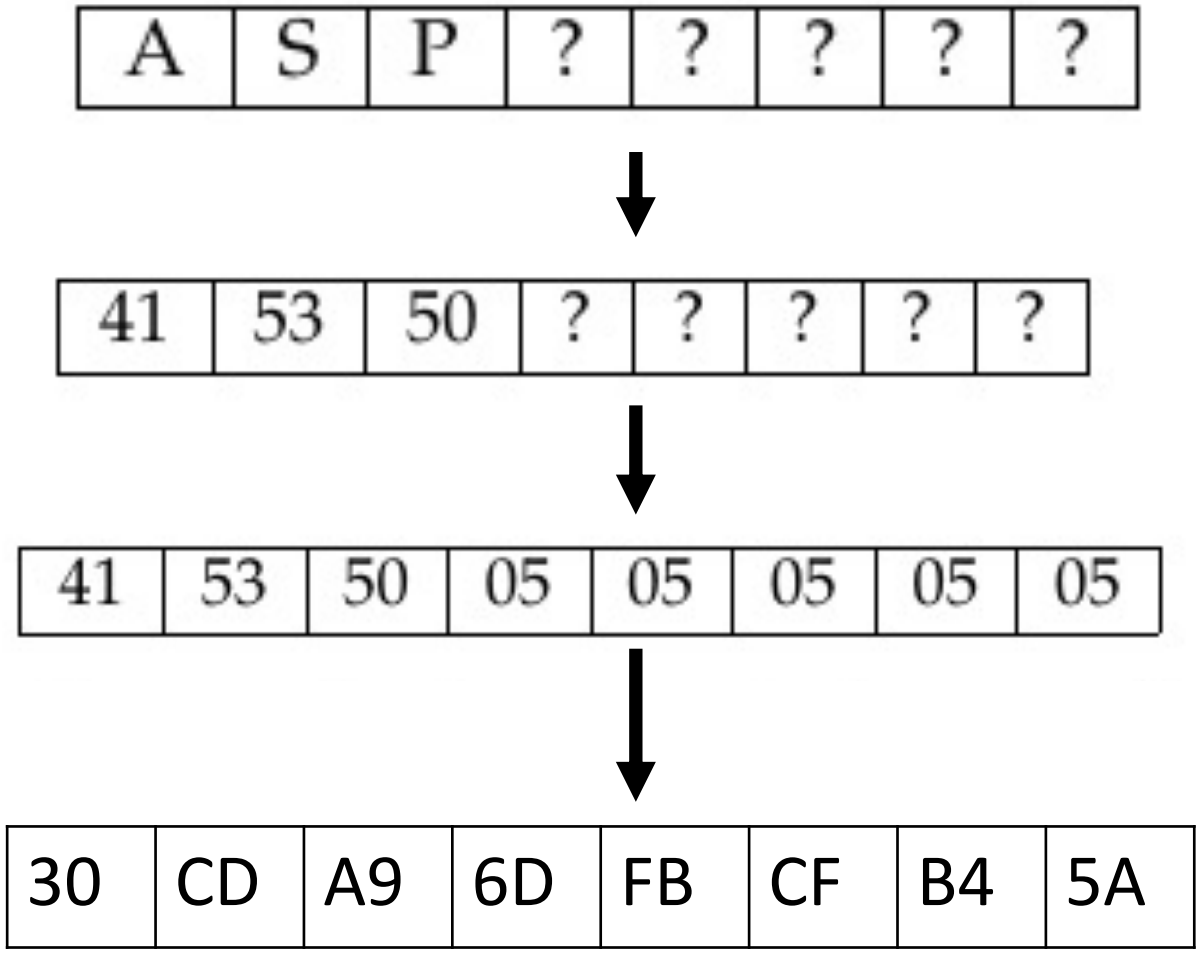


Abbildung 4: Verschlüsselung einer 3-Byte Nachricht

Implementierung

- 3 Implementierungen von uns
  + die Referenz-C-Implementierung von Needham und Wheeler
  + Maschinengenerierte Assemblerimplementierung eines unseren C-Codes
- Einfache Implementierung, wenige Zeilen Code
- Auch einfache Padding-Implementierung ohne weitere Hilfsfunktionen

Korrektheit

- Ergebniswerte von allen unseren Varianten Identisch mit der von Needham und Wheeler
- Vergleich des Werts vom linken Teil nach der ersten Runde per Hand und durch unsere C-Implementierung
- Für die Assemblerimplementierung nur Caller-saved Registers genutzt und keine unnötige Speicherzugriffe

Performanzanalyse

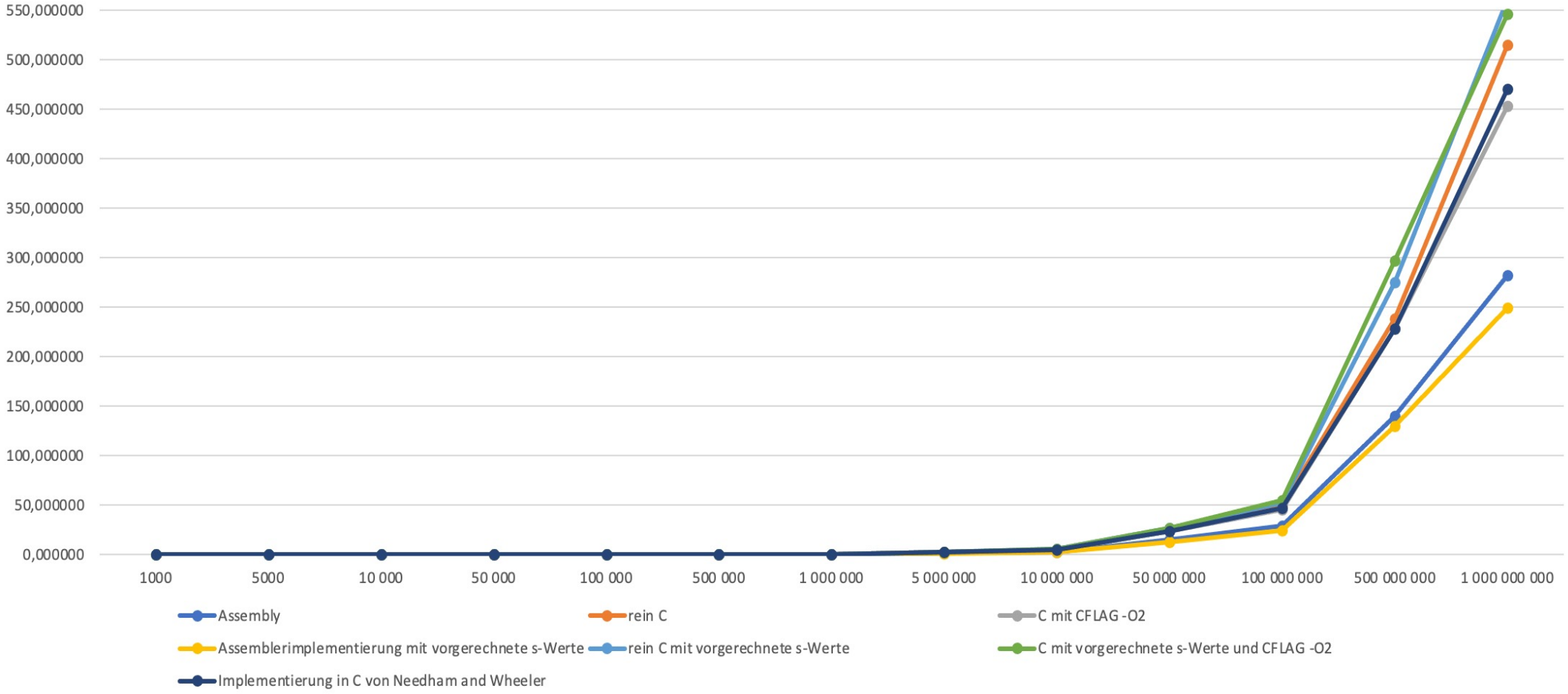
- Beide Assemblerimplementierungen deutlich schneller als gcc 9.3.0 -O2
- V3 hat 11% weniger Laufzeit als V0, wie unsere Behauptung
- C-Implementierungen mit gcc 9.3.0 fast verdoppelte Laufzeiten

	V0	V1	V1 + O2	V2	V3	V4
1 Mrd.	282,54	515,40	453,51	536,21	249,62	470,82

Tablle 1: Laufzeit aller Varianten in Sekunden in 1 Mrd. Rundenzahl

Performanzanalyse

Graph 1: Laufzeit aller Varianten in Rundenzahl von 1000 - 1 Mrd. in Sekunden



Schwierigkeiten bei der Implementierung*

*was wir leider nicht schaffen könnten

- Kein Keyparser im Rahmenprogramm wegen 128-Bits und zu vielen Integerdarstellungsarten
- Padding als eigene Funktion wegen Spaghetticode
- Zeitmanagement

  Für Keyparser verbrachte Tage: 5 (nicht geschafft)

  Für alles andere: 4
- Implementierung mit einer vorgerechneten s-Variable in C-Code

Zusammenfassung

Bilderverzeichnis

Danke fürs Zuhören!