LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Aspekte der systemnahen Programmierung bei der Spieleentwicklung

Gruppe team 117 – Abgabe zu Aufgabe 502: XTEA
Wintersemester 2021/22

Guo, Linfeng
Gönenc, Hazar
Özakay, Baris

1 Einleitung

Im Rahmen unseres Projektes im Fach Aspekte der systemnahen Programmierung bei der Spieleentwicklung war es unsere Aufgabe, ein Ver- und Entschlüsselungsalgorithmus XTea in Assemblercode zu programmieren. Diese Aufgabe lässt sich in folgende Bereiche aufteilen: Konzeption, die Funktionsweise des XTEA Algorithmus und Verfahren für die Optimierung des Algorithmus, verstehen; der XTEA Algorithmus in Assemblercode zu implementieren ... (Aufgabe von Baris). Die Bearbeitung dieser Teilbereiche wird im Folgenden Beschrieben.
In der Grafik zu sehen ist ein Beispiel für die Ver- und Entschlüsselung mit dem XTEA Algorithmus.

2 Lösungsansatz

2.1. Feistelchiffre

Der XTEA Algorithmus ist ein Verschlüsselungsalgorithmus, welche auf die Struktur von Feistelchiffre basiert ist. Feistelchiffre ist eine Struktur, die für symmetrische Verschlüsselung verwendet wird. Unter symmetrische Verschlüsselung ist zu verstehen, dass für die Ver- und Entschlüsselung nur ein Key verwendet wird. Wir würden daher erstmal mit Feistelchiffre, die grundstruktur der symmetrische Verschlüsselung, eingehen.
Feistelchiffre lässt sich in vier Schritten aufteilen. Zuerst hat man ein Klartextblock mit der Nachricht, weche meist 8 Bytes ist, und teilt diese in zwei gleich große Blöcke L0 und R0, die 4 Bytes entsprechen auf.

B	E	I	S	P	I	E	L

Tabelle 1: Nachricht

42	45	49	53	50	49	25	4C

Tabelle 2: Nachricht in Hex Code

42	45	49	53

Tabelle 3: L0

50	49	25	4C

Tabelle 4: R0

Danach findet die eigentliche Verschlüsselung statt. Man führt die Verschlüsselungsfunktion mit der Schlüssel,der ebenfalls 4 Byte entspricht, auf R0 aus. Die Funktion bei Feistelchiffre ist allerdings undefiniert, da Feistelchiffre nur die Struktur anbietet. Der Wert der durch die Funktion ergibt, wird anschließend mit dem linken Teil durch die XOR- Operation eingefangen. So ergibt sich das neue R1. Die ursprüngliche R0 wird dann zu L1. Nach jeder Verschlüsselung wird die Position vom linken und dem rechten Block getauscht, sowie oben beschrieben wird.
Die Entschlüsselung funktioniert im Prinzip genau wie die Verschlüsselung. Nur mit dem Unterschied, dass man die Position vom linken und rechten Teil tauscht. Also wird Ln+1 in die Funktion mit demselben Key eingesetzt und das Ergebnis wird dann mit Rn+1 per XOR abgebildet. Der Wert wird dann zu Rn und Ln+1 wird zu Ln.

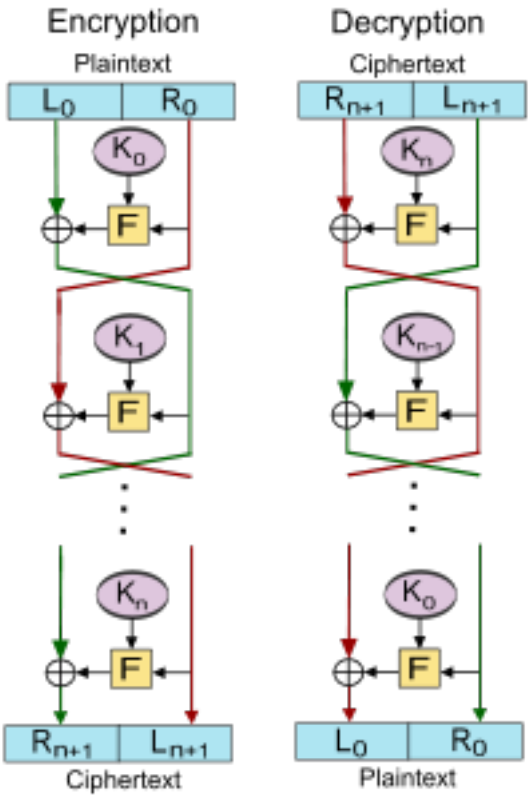

Abbildung 1: Feistelchiffre

2.2.XTEA

2.3.Padding

3 Korrektheit/Genauigkeit

I/O-Operationen in C

Implementierung in C

Implementierung in Assemblercode

4 Performanzanalyse

5 Zusammenfassung und Ausblick