LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Grundlagenpraktikum: Rechnerarchitektur

Gruppe 215 – Abgabe zu Aufgabe A500
Sommersemester 2022

David Csida Georgios Merezas Fabian Degen

1 Einleitung

1.1 Einführung

Sei es beim Herstellen einer Verbindung mit einem Server oder das Chatten mit Freunden und Bekannten rund um den Globus, nie genoss die Kryptographie mehr Relevanz als jetzt. Man möchte nicht, dass die Nachrichten, die man an seine Geliebten versendet, von einer dritten Person mitgelesen werden können. Darum existieren kryptographische Verfahren zur Verschlüsselung von Informationen. Eines dieser Verschlüsselungsverfahren ist Salsa20/20, welches es zu implementieren galt.

1.2 Funktionsweise Salsa20/20

Salsa20/20 ist eine Stromchiffre basierend auf einem sogenannten Add-Rotate-XOR-Schema, ins Leben gerufen von David J. Bernstein.
Salsa20/20 verwendet eine 4×4 Matrix bestehend aus vorzeichenlosen 32-bit Little-Endian-Ganzzahlen, welche durch Add, Rotate und XOR Operationen aus bestimmen Startwerten erzeugt wird. Auf dieser generierten Matrix wird dann mit der zu verschlüsselnden Nachricht Byte für Byte eine XOR-Operation ausgeführt.

1.2.1 Der Salsa20-Kern:

Der Salsa20-Kern ist eine Funktion, welche einen 64-Byte Block generiert, auf dem, wie oben beschrieben, auf der zu verschlüsselnden Nachricht Byte für Byte eine Add-Rotate-XOR-Operation ausgeführt wird. Der 64-Byte-Block wird in 20 sogenannten Runden aus einem 256-Bit-Key, einer 64-Bit Nonce und einem 64-Bit Counter generiert.

1.3 Aufgabenstellung

1.3.1 Theoretischer Teil:

Folgende theoretische Fragen waren zu beantworten:

- Wie könnte man das im letzten Schritt einer jeden Salsa20-Kern Runde ineffiziente, stattfindende transponieren der Matrix optimieren?
- Was bedeuten die Werte an der Diagonalen des Startzustands?

- Erklären der Funktionsweise einer Stromchiffre anhand eines Beispiels. Warum kann man dieselbe Funktion zum Ver- und Entschlüsseln verwenden? Wie müssen die Parameter gewählt werden, damit dies funktioniert? Warum ist der Counter keine EIngabe des Verschlüsselungsalgorithmus?

1.3.2 Praktischer Teil:

Zu implementieren war folgende Funktionalität:

- Ein Rahmenprogramm welches IO-Operationen unterstützt, mithilfe derer man eine ganze Datei in den Speicher einlesen und als Pointer an eine Unterfunktion übergeben kann. Selbiges soll auch zum Schreiben eines Speicherbereiches mit bekannter Länge in eine Datei möglich sein.
- void salsa20_core(uint32_t output[16] const uint32_t input[16])
  welche den oben beschriebenen Salsa20-Kern implementiert. input bezeichnet den Startzustand des Kerns, output den finalen 64-Byte-Block.
- void salsa20_crypt(size_t mlen, const uint8_t msg[mlen], uint8_t cipher [mlen], uint32_t key[8], uint64_t iv)
  welche eine Nachricht msg der Länge mlen mit einem gegebenen Schlüssel und Nonce (iv) verschlüsselt und das Ergebnis in cipher schreibt.

2 Lösungsansatz

2.1 Theoretischer Teil

2.1.1 Transponieren

Das Transponieren der Matrix ist Teil des Salsa20-Kern Algorithmus, aber ist nicht sonderlich performant. Um ihn weiter zu optimieren, kann man anstatt 20 Runden von der Kern Funktion auszuführen und nach jeder Runde die Matrix zu transponieren, nur 10 Mal durch die Schleife iterieren und in jedem Schleifendurchlauf zwei Runden nacheinander ausführen, aber mit den “transponierten” Indexen der Matrix.

2.1.2 Werte an der Diagonale

2.1.3 Funktionsweise einer Stromchiffre

Eine Stromchiffre ist ein symmetrischer kryptographischer Algorithmus, der für Ver- und Entschlüsselung von Daten verwendet wird. Dieser Algorithmus nimmt einen Klartext und ein Schlüsselstrom und führt eine bitweise XOR-Operation aus und gibt am Ende einen Geheimtext zurück. Man kann durch Benutzung von demselben Schlüsselstrom wieder den Klartext erhalten, wenn man dem Algorithmus den Geheimtext übergibt. Aufgrund der Symmetrie der XOR-Operation gilt folgendes:

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

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

Wie man sehen kann, führt der Geheimtext als Eingabe unter Verwendung desselben Schlüsselstroms wieder zum Klartext.

Damit man also für einen durch Salsa20/20 erstellten Geheimtext den dazugehörigen Klartext zurückerhält, muss man denselben Schlüsselstrom sowie denselben Initalisierungsvektor, mit dem Geheimtext als zu verschlüsselnde Nachricht, übergeben.
Der Counter ist keine Eingabe des Salsa20/20 Verschlüsselungsverfahren, sondern dient dazu den generierten 64-Byte-Block sicherer gegen Angriffe zu machen. Alle 64 verschlüsselte Bytes, wird der Counter um 1 inkrementiert und der Kern neu berechnet. Das macht es schwer, den Kern vorherzusagen und bietet somit mehr Sicherheit für das Verschlüsselungsverfahren und schützt den Geheimtext gegen Angriffe.

2.2 Praktischer Teil

3 Korrektheit

4 Performanzanalyse

5 Zusammenfassung und Ausblick

Literatur

[1] Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual*. Intel Corporation, April 2016. `http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html`, visited 2017-08-19.

[1]