LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Grundlagenpraktikum: Rechnerarchitektur

Salsa20/20 (A500)
Projektaufgabe – Aufgabenbereich Kryptographie

1 Organisatorisches

Auf den folgenden Seiten finden Sie die Aufgabenstellung zu Ihrer Projektaufgabe für das Praktikum. Die Rahmenbedingungen für die Bearbeitung werden in der Praktikumsordnung festgesetzt, die Sie über die Praktikumshomepage[1] aufrufen können.

Wie in der Praktikumsordnung beschrieben, sind die Aufgaben relativ offen gestellt. Besprechen Sie diese innerhalb Ihrer Gruppe und konkretisieren Sie die Aufgabenstellung. Die Teile der Aufgabe, in denen C-Code anzufertigen ist, sind in C nach dem C17-Standard zu schreiben.

Der **Abgabetermin** ist **Sonntag 16. Juli 2023, 23:59 Uhr (CEST)**. Die Abgabe erfolgt per Git in das für Ihre Gruppe eingerichtete Projektrepository. Bitte beachten Sie die in der `README.md` angegebene Liste von abzugebenden Dateien.

Die **Abschlusspräsentationen** finden in der Zeit vom **21.08.2023 – 01.09.2023** statt. Weitere Informationen werden noch bekannt gegeben. Beachten Sie, dass die Folien für die Präsentation am obigen Abgabetermin im PDF-Format abzugeben sind und keine nachträglichen Änderungen akzeptiert werden können.

Bei Fragen/Unklarheiten in Bezug auf den Ablauf und die Aufgabenstellung wenden Sie sich bitte an Ihren Tutor.

Wir wünschen Ihnen viel Erfolg und Freude bei der Bearbeitung Ihrer Aufgabe!

Mit freundlichen Grüßen
Die Praktikumsleitung

[1] `https://gra.caps.in.tum.de`

2 Salsa20/20

2.1 Überblick

Kryptographie ist ein essentieller Bestandteil unserer heutigen Kommunikation, beispielsweise beim Surfen im Internet über `HTTPS` oder beim Versenden Ende-zu-Ende verschlüsselter E-Mails mit PGP. In Ihrer Projektaufgabe werden Sie ein kryptographisches Verfahren ganz oder teilweise implementieren.
Für das Verständnis der folgenden Aufgabe ist wichtig, zunächst einige Definitionen zu tätigen:

1. $x \lll y$ bezeichne die Linksrotation von x um y Bit.
2. $x \ggg y$ bezeichne die Rechtsrotation von x um y Bit.
3. $\oplus$ bezeichne den binären XOR-Operator.

2.2 Funktionsweise

Salsa20/20 ist eine von Daniel J. Bernstein (alias *djb*) erfundene Stromchiffre, die auf einem sogenannten *Add-Rotate-XOR* (kurz *ARX*)- Schema aufbaut.

2.2.1 Der Salsa20-Kern

Der Salsa20-Kern ist eine Funktion, die aus einem 256-Bit-Key (32 Byte), einer 64-Bit-Nonce (8 Byte) und einem 64-Bit-Block-Counter in insgesamt 20 sog. Runden einen 64-Byte-Block erzeugt. Den Anfangszustand bildet dabei eine 4×4-Matrix aus 32-Bit vorzeichenlosen Little-Endian-Ganzzahlen. Bezeichne K_i den i-ten Teil des Schlüssels (K_0 steht also für die ersten 4 Byte in Little-Endian-Reihenfolge des Schlüssels, K_1 für die nächsten 4 Byte und so weiter). Dasselbe gilt analog für die Nonce N_i sowie den Counter C_i. Dann ist der Anfangszustand S gegeben durch:

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

Die Konstanten auf der Diagonalen sind für jeden Key und jede Nonce dieselben.
Nun wird mit der eigentlichen Erezugung des Keystreams begonnen. Im Folgenden wird der Ablauf einer *Runde* dargestellt. Für den finalen Salsa20/20- Algorithmus werden, bevor ein Block ausgegeben wird, 20 dieser Runden ausgeführt.

Wir wählen wie üblich die Schreibweise $a_{i,j} \in A$ um einen einzelnen Eintrag der obigen Matrix anzugeben. Dann wird jedes Wort unter der Diagonalen wie folgt modifiziert:

$$\begin{aligned} a_{2,1} &= ((a_{1,1} + a_{4,1}) \lll 7) \oplus a_{2,1} \\ a_{3,2} &= ((a_{2,2} + a_{1,2}) \lll 7) \oplus a_{3,2} \\ a_{4,3} &= ((a_{3,3} + a_{2,3}) \lll 7) \oplus a_{4,3} \\ a_{1,4} &= ((a_{4,4} + a_{3,4}) \lll 7) \oplus a_{1,4} \end{aligned}$$

Nun fahren wir ähnlich fort, addieren jedoch immer das Wort unter der Diaognalen mit der Diagonalen und rotieren dabei um 9 Bit:

$$\begin{aligned} a_{3,1} &= ((a_{1,1} + a_{2,1}) \lll 9) \oplus a_{3,1} \\ a_{4,2} &= ((a_{2,2} + a_{3,2}) \lll 9) \oplus a_{4,2} \\ a_{1,3} &= ((a_{3,3} + a_{4,3}) \lll 9) \oplus a_{1,3} \\ a_{2,4} &= ((a_{4,4} + a_{1,4}) \lll 9) \oplus a_{2,4} \end{aligned}$$

Dasselbe passiert nun eine Zeile nach unten versetzt. Nun wird um 13 Bit rotiert:

$$\begin{aligned} a_{4,1} &= ((a_{3,1} + a_{2,1}) \lll 13) \oplus a_{4,1} \\ a_{1,2} &= ((a_{4,2} + a_{3,2}) \lll 13) \oplus a_{1,2} \\ a_{2,3} &= ((a_{1,3} + a_{4,3}) \lll 13) \oplus a_{2,3} \\ a_{3,4} &= ((a_{2,4} + a_{1,4}) \lll 13) \oplus a_{3,4} \end{aligned}$$

Als vorletzten Schritt wird nun die Diagonale selbst modifiziert:

$$\begin{aligned} a_{1,1} &= ((a_{4,1} + a_{3,1}) \lll 18) \oplus a_{1,1} \\ a_{2,2} &= ((a_{1,2} + a_{4,2}) \lll 18) \oplus a_{2,2} \\ a_{3,3} &= ((a_{2,3} + a_{1,3}) \lll 18) \oplus a_{3,3} \\ a_{4,4} &= ((a_{3,4} + a_{2,4}) \lll 18) \oplus a_{4,4} \end{aligned}$$

Das finale Ergebnis einer Runde entsteht, indem die Matrix A transponiert wird:

$$A = A^T$$

Am Ende aller Runden wird die aktuelle Matrix A noch auf den Startzustand S addiert. Somit ergibt sich für den 64-Byte-Ausgabeblock O:

$$O = A + S$$

Alle Additionen werden auf dem Restklassenring $\mathbb{Z}/(32)$ durchgeführt.

2.3 Aufgabenstellungen

Ihre Aufgaben lassen sich in die Bereiche Konzeption (theoretisch) und Implementierung (praktisch) aufteilen. Sie können (müssen aber nicht) dies bei der Verteilung der Aufgaben innerhalb Ihrer Arbeitsgruppe ausnutzen. Antworten auf konzeptionelle Fragen sollten an den passenden Stellen in Ihrer Ausarbeitung in angemessenem Umfang erscheinen. Entscheiden Sie nach eigenem Ermessen, ob Sie im Rahmen Ihres Abschlussvortrags auch auf konzeptionelle Fragen eingehen. Die Antworten auf die Implementierungsaufgaben werden durch Ihren Code reflektiert.

2.3.1 Theoretischer Teil

- Das Transponieren der Matrix im letzten Schritt ist nicht sonderlich performant. Wie könnte an dieser Stelle weiter optimiert werden?
- Was bedeuten die Werte an der Diagonale?
- Erklären Sie anhand eines Beispiels die Funktionsweise einer Stromchiffre. Warum können Sie dieselbe Funktion zum Ver- und Entschlüsseln verwenden? Wie müssen die Parameter gewählt werden, damit dies funktioniert? Warum ist der Counter *keine* Eingabe des Verschlüsselungsalgorithmus?

2.3.2 Praktischer Teil

- Implementieren Sie in Ihrem Rahmenprogramm I/O-Operationen in C, mithilfe derer Sie eine ganze Datei in den Speicher einlesen und als Pointer an eine Unterfunktion übergeben können. Implementieren Sie selbiges ebenfalls zum Schreiben eines Speicherbereiches mit bekannter Länge in eine Datei.
- Implementieren Sie in der Datei mit Ihrem C-Code die Funktion:

  ```
  void salsa20_core(uint32_t output[16], const uint32_t input[16])
  ```

  Die Funktion implementiert den oben beschriebenen Salsa20-Kern. `input` bezeichne dabei den Startzustand S, `output` den finalen 64-Byte-Block O.
- Implementieren Sie in der Datei mit Ihrem C-Code die Funktion:

  ```
  void salsa20_crypt(size_t mlen, const uint8_t msg[mlen], uint8_t
            cipher[mlen], uint32_t key[8], uint64_t iv)
  ```

  Die Funktion verschlüsselt eine Nachricht `msg` der Länge `mlen` mit einem gegebenen Schlüssel und Nonce (`iv`) und schreibt das Ergebnis in `cipher`.

2.3.3 Rahmenprogramm

Ihr Rahmenprogramm muss bei einem Aufruf die folgenden Optionen entgegennehmen und verarbeiten können. Wenn möglich soll das Programm sinnvolle Standardwerte definieren, sodass nicht immer alle Optionen gesetzt werden müssen.

- `-V<Zahl>` — Die Implementierung, die verwendet werden soll. Hierbei soll mit `-V 0` Ihre Hauptimplementierung verwendet werden. Wenn diese Option nicht gesetzt wird, soll ebenfalls die Hauptimplementierung ausgeführt werden.
- `-B<Zahl>` — Falls gesetzt, wird die Laufzeit der angegebenen Implementierung gemessen und ausgegeben. Das *optionale* Argument dieser Option gibt die Anzahl an Wiederholungen des Funktionsaufrufs an.
- `<Dateiname>` — Positional Argument: Eingabedatei
- `-k<Zahl>` — Schlüssel
- `-i<Zahl>` — Initialisierungsvektor
- `-o<Dateiname>` — Ausgabedatei
- `-h` — Eine Beschreibung aller Optionen des Programms und Verwendungsbeispiele werden ausgegeben und das Programm danach beendet.
- `--help` — Eine Beschreibung aller Optionen des Programms und Verwendungsbeispiele werden ausgegeben und das Programm danach beendet.

Sie dürfen weitere Optionen implementieren, beispielsweise um vordefinierte Testfälle zu verwenden. Ihr Programm muss jedoch nur unter Verwendung der oben genannten Optionen verwendbar sein. Beachten Sie ebenfalls, dass Ihr Rahmenprogramm etwaige Randfälle korrekt abfangen muss und im Falle eines Fehlers mit einer aussagekräftigen Fehlermeldung auf `stderr` und einer kurzen Erläuterung zur Benutzung terminieren sollte.

2.4 Allgemeine Bewertungshinweise

Beachten Sie grundsätzlich alle in der Praktikumsordnung angegebenen Hinweise. Die folgende Liste konkretisiert einige der Bewertungspunkte:

- Stellen Sie unbedingt sicher, dass *sowohl* Ihre Implementierung *als auch* Ihre Ausarbeitung auf der Referenzplattform des Praktikums (`lxhalle`) kompilieren und vollständig korrekt bzw. funktionsfähig sind.
- Die Implementierung soll mit GCC/GNU as kompilieren. Verwenden Sie keinen Inline-Assembler. Achten Sie darauf, dass Ihr Programm keine x87-FPU- oder MMX-Instruktionen und SSE-Erweiterungen nur bis SSE4.2 verwendet. Andere ISA-Erweiterungen (z.B. AVX, BMI1) dürfen Sie nur benutzen, sofern Ihre Implementierung auch auf Prozessoren ohne derartige Erweiterungen lauffähig ist.

- Sie dürfen die angegebenen Funktionssignaturen (nur dann) ändern, wenn Sie dies (in Ihrer Ausarbeitung) begründen.
- Verwenden Sie die angegebenen Funktionsnamen für Ihre Hauptimplementierung. Falls Sie mehrere Implementierungen schreiben, legen wir Ihnen nahe, für die Benennung der alternativen Implementierungen mit dem Suffix „`_V1`“, „`_V2`“ etc. zu arbeiten.
- Denken Sie daran, das Laufzeitverhalten Ihres Codes zu testen (Sichere Programmierung, Performanz) und behandeln Sie *alle möglichen Eingaben*, auch Randfälle. Ziehen Sie ggf. alternative Implementierungen als Vergleich heran.
- Eingabedateien, welche Sie generieren, um Ihre Implementierungen zu testen, sollten mit abgegeben werden; größere Eingaben sollten stattdessen stark komprimiert oder (bevorzugt) über ein abgegebenes Skript generierbar sein.
- Stellen Sie Performanz-Ergebnisse nach Möglichkeit grafisch dar.
- Vermeiden Sie unscharfe Grafiken und Screenshots von Code.
- Geben Sie die Folien für Ihre Abschlusspräsentation im PDF-Format ab. Achten Sie auf hinreichenden Kontrast (schwarzer Text auf weißem Grund!) und eine angemessene Schriftgröße. Verwenden Sie 16:9 als Folien-Format.