LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Grundlagenpraktikum: Rechnerarchitektur

XTEA (A502)
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 Assembler-Code anzufertigen ist, sind für die 64-Bit x86-Architektur (x86-64) unter Verwendung der SSE-Erweiterungen zu schreiben; alle anderen Bestandteile der Hauptimplementierung sind in C nach dem C11-Standard anzufertigen.

Der **Abgabetermin** ist **Sonntag 24. Juli 2022, 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 **24.08.2022 – 01.09.2022** 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 XTEA

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.

2.2 Funktionsweise

Der *XTEA*[2] Algorithmus [1] wurde entwickelt, um einige Schwachstellen in TEA [3] zu beheben. Wie *TEA* ist *XTEA* eine Feistelchiffre mit 64-Bit pro Block und 64 Runden. In einer Runde wird ein sogenanntes Delta eingebracht, das wie bei *TEA* wie folgt definiert ist:

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

Damit erhalten wir Algorithmus 1 für die Verschlüsselung eines Blocks und analog dazu Algorithmus 2 für die Entschlüsselung.

Algorithm 1 Verschlüsselung

function XTEA_ENCRYPT_BLOCK($v \in \mathbb{Z}/(32)^2$, $k \in \mathbb{Z}/(32)^4$)
  $p_1 \leftarrow v_1$
  $p_2 \leftarrow v_2$
  $s \leftarrow 0$
  for $i \leftarrow 1, \ldots, 64$ **do**
    $v_1 \leftarrow v_1 + (((v_2 \ll 4) \oplus (v_2 \gg 5)) + v_2) \oplus (s + k_{(s\&3)})$
    $s \leftarrow s + \delta$
    $v_2 \leftarrow v_2 + (((v_1 \ll 4) \oplus (v_1 \gg 5)) + v_1) \oplus (s + k_{((s \gg 11)\&3)})$
  end for
end function

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] E**x**tended **T**iny **E**ncryption **A**lgorithm
[3] **T**iny **E**ncryption **A**lgorithm

Algorithm 2 Entschlüsselung

function XTEA_DECRYPT_BLOCK($v \in \mathbb{Z}_{32}^2$, $k \in \mathbb{Z}_{32}^4$)
  $p_1 \leftarrow v_1$
  $p_2 \leftarrow v_2$
  $s \leftarrow \delta \cdot 64$
  for $i \leftarrow 1, \ldots, 64$ **do**
    $v_2 \leftarrow v_2 - (((v_1 \ll 4) \oplus (v_1 \gg 5)) + v_1) \oplus (s + k_{((s \gg 11)\&3)})$
    $s \leftarrow s - \delta$
    $v_1 \leftarrow v_1 - (((v_2 \ll 4) \oplus (v_2 \gg 5)) + v_2) \oplus (s + k_{(s\&3)})$
  end for
end function

2.3.1 Theoretischer Teil

- Beschreiben Sie in Ihrer Ausarbeitung - unter Zuhilfenahme geeigneter Sekundärliteratur - den Aufbau einer Feistelchiffre. Können Sie Vergleiche zum Aufbau von *XTEA* ziehen?
- Die Länge der zu verschlüsselnden Daten muss immer ein Vielfaches der Blocklänge sein. Ein Verfahren zum sogenannten *padding* auf Blocklänge ist *PKCS#7*. Recherchieren Sie jenes in geeigneter Sekundärliteratur und beschreiben Sie es in Ihrer Ausarbeitung.
- Der gegebene Algorithmus ver- bzw. entschlüsselt einen Block. Wie können Sie Daten verschlüsseln, deren Länge nicht genau einem Block entspricht? Suchen Sie nach Standardverfahren in geeigneter Sekundärliteratur.

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 xtea_encrypt_block(uint32_t v[2], const uint32_t key[4])
  ```

  Die Funktion bekommt einen Block (v) sowie den Key als Parameter übergeben und verschlüsselt den Block in-place.
- Implementieren Sie in der Datei mit Ihrem C-Code die Funktion:

  ```
  void xtea_decrypt_block(uint32_t v[2], const uint32_t key[4])
  ```

Die Funktion bekommt einen Block (`v`) sowie den Key als Parameter übergeben bekommt und entschlüsselt den Block in-place.

- Implementieren Sie eine Funktion in C, die die zu verschlüsselnden Daten auf ein Vielfaches der Blocklänge paddet.
- Implementieren Sie eine Funktion in C, die ein geeignetes Verfahren zum Verschlüsseln von Daten, deren Länge mehr als einen Block beträgt, ausführt.

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<int>` — Die Implementierung, die verwendet werden soll. Hierbei soll mit `-V0` Ihre Hauptimplementierung verwendet werden. Wenn diese Option nicht gesetzt wird, soll ebenfalls die Hauptimplementierung ausgeführt werden.
- `-B<int>` — Falls gesetzt, wird die Laufzeit der angegebenen Implementierung gemessen und ausgegeben. Das optionale Argument dieser Option gibt die Anzahl an Wiederholungen des Funktionsaufrufs an.
- `<file>` — Positional Argument: Eingabedatei
- `-k<int>` — Schlüssel
- `-i<int>` — Initialisierungsvektor
- `-d` — Falls gesetzt, wird entschlüsselt, ansonsten verschlüsselt
- `-o<file>` — 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. 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.

Literatur

[1] Roger M. Needham und David J. Wheeler. *Tea extensions*. Techn. Ber. University of Cambridge, Okt. 1997.