LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Praktikum Rechnerarchitektur

RC5 (A501)

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.

Der **Abgabetermin** ist der **24. Juli 2019, 23:59 Uhr (MESZ)**. Die Abgabe erfolgt per Git in das für Ihre Gruppe eingerichtete Projektrepository. Bitte beachten Sie die in der Praktikumsordnung angebene Liste von abzugebenden Dateien.

Der **erste Teil Ihrer Projektpräsentation** ist eine kurze Vorstellung Ihrer Aufgabe im Umfang von ca. 2 Minuten in einer Tutorübung in der Woche **17.06.2019 – 21.06.2019**. Erscheinen Sie bitte **mit allen Team-Mitgliedern** und wählen Sie bitte eine Übung, in der mindestens ein Team-Mitglied angemeldet ist.

Die **Abschlusspräsentationen** finden in der Zeit vom **12.08.2019 – 30.08.2019** statt. Weitere Informationen zum Ablauf und die Zuteilung der einzelnen Präsentationstermine werden noch bekannt gegeben. Beachten Sie, dass die Folien für die Präsentation am obigen Abgabetermin im PDF-Format abzugeben sind.

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

PS: Vergessen Sie nicht, sich rechtzeitig in TUMonline zur Prüfung anzumelden. Dies ist Voraussetzung für eine erfolgreiche Teilnahme am Praktikum im laufenden Semester.

[1] `https://www.caps.in.tum.de/lehre/ss19/praktika/era/`

2 RC5

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

RC5[2] ist ein von Ronald Rivest 1994 beschriebenes Verfahren zur **symmetrischen Blockverschlüsselung**. Dabei werden die Daten zunächst in Blöcke getrennt und dann in wiederholter Anwendung einfacher mathematischer Operationen, den sogenannten Runden, verschlüsselt. Schlüssellänge und Rundenzahl sind dabei (in gewissen Schranken) frei wählbar.

2.2.1 Initialisierung

Bezeichne $r \in \mathbb{N}$ die Anzahl der Runden. Bevor Daten verschlüsselt werden können, müssen zunächst aus dem (geheimen) Schlüssel die Rundenschlüssel $S_0, ..., S_{r+2}$ generiert werden. Dieses System aus erweiterten Rundenschlüsseln wird *erweiterte Schlüsseltabelle* genannt.
Für die Berechnung des Anfangszustands der Rundenschlüssel sei folgender Pseudocode gegeben:

Algorithm 1 Schlüsselexpansion

```
1: S_0 ← P
2: for i ← 1, ... 2r + 1 do
3:     S_i ← S_{i−1} + Q
4: end for
```

P und Q sind dabei ungerade Ganzzahlen, die mit der eulerschen Zahl e und dem goldenen Schnitt ϕ in Abhängigkeit der gewählten Blockgröße gewählt werden. Um diese zu berechnen, ist es notwendig, zunächst die Größe eines Halbblocks w in Bits zu

[2] **R**ivest **C**ipher **5**

bestimmen. Ein Block muss immer aus zwei gleich langen Halbblöcken bestehen. Nun können P und Q wie folgt bestimmt werden:

$$P = Odd((e-2) \cdot 2^w) \tag{1}$$
$$Q = Odd((\phi-1) \cdot 2^w) \tag{2}$$

Dabei bezeichne $Odd(x)$ aus (1) und (2) die jeweils nächste ungerade Zahl.
Bevor nun mit der Ver- und Entschlüsselung begonnen werden kann, müssen die zunächst festen Rundenschlüssel noch mit dem gewählten Schlüssel vermischt werden. Dazu wird der Schlüssel zunächst in Halbblöcke L_i aufgeteilt. Sollte sich der Schlüssel nicht gänzlich auf die Halbblöcke aufteilen lassen, wird mit Nullen aufgefüllt.

Algorithm 2 Key-Mixing

1: $k \leftarrow i \leftarrow 0$
2: $A \leftarrow B \leftarrow 0$
3: **for** $j \leftarrow 0, \ldots, 3 \cdot \max(2(r+1), \lceil \frac{b}{w/8} \rceil)$ **do**
4: $\quad A \leftarrow S_k \leftarrow (S_k + A + B) \lll 3$
5: $\quad B \leftarrow L_i \leftarrow (L_i + A + B) \lll (A + B)$
6: $\quad k \leftarrow (k+1) \mod (2(r+1)$
7: $\quad i \leftarrow (i+1) \mod (\lceil \frac{b}{w/8} \rceil)$
8: **end for**

Hierbei bezeichne w die Länge eines Halbblocks in Bit, b die Länge des Schlüssels in Byte.

2.2.2 Verschlüsselung

Für einen Block B, bestehend aus den Halbblöcken B_a und B_b, wird die Verschlüsselung wie folgt durchgeführt. Eine notwendige Vorbedingung dazu ist, dass B in *Little-Endian*-Darstellung vorliegt. S_0 und S_1 werden dabei zum sogenannten *Key-Whitening*, also einer Verknüpfung mit den ersten zwei Halbblöcken des Klartextes (S_0 mit dem Halbblock B_{0_a} und S_1 mit B_{0_b} wobei B die Menge der Blöcke und B_i den i-ten Block bezeichnet). Damit ergibt sich für die Verschlüsselung:

Algorithm 3 Verschlüsselung

1: $A \leftarrow A + S_0$ ▷ Key-Whitening durchführen
2: $B \leftarrow B + S_1$
3: **for** $k \leftarrow 1, \ldots, r$ **do**
4: $\quad A \leftarrow ((A \oplus B) \lll B) + S_{2k}$
5: $\quad B \leftarrow ((B \oplus A) \lll A) + S_{2k+1}$
6: **end for**

Somit entspricht jede Halbrunde eine Runde einer *Feistelchiffre*[3].

[3] Schlagen sie diesen Begriff in geeigneter Sekundärliteratur nach

2.2.3 Entschlüsselung

Die Entschlüsselung verläuft nun analog zur Verschlüsselung, der Pseudocode ist dargestellt in *Algorithm 4*.

Algorithm 4 Entschlüsselung

1: **for** $k \leftarrow 1, \ldots, r$ **do**
2: $\quad B \leftarrow ((B - S_{2k+1}) \ggg A) \oplus A$
3: $\quad A \leftarrow ((A - S_{2k}) \ggg B) \oplus B$
4: **end for**
5: $B \leftarrow B - S_1$
6: $A \leftarrow A - S_0$

Damit ist RC5 vollständig beschrieben. Sollten Sie weitere Sekundärliteratur benötigen, so empfehlen wir Ihnen *Angewandte Kryptographie, Schneier B., Pearson Studium, ISBN 978-3893198542*. Sie finden eine Beschreibung zu RC5 in Kapitel 14.

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. Alle Antworten auf konzeptionelle Fragen sollten in Ihrer Ausarbeitung erscheinen. Besprechen Sie nach eigenem Ermessen außerdem im Zuge Ihres Vortrags einige der konzeptionellen Fragen. Die Antworten auf die Implementierungsaufgaben werden durch Ihrem Code reflektiert.

2.3.1 Theoretischer Teil

- Als Blocklänge wählen wir 32 Bit mit 16 Runden. Berechnen Sie – auf dem Papier – in Abhängigkeit dieser P und Q. Sind 16 Runden genug, damit der Algorithmus sicher ist?
- RC5 besteht aus einer Aneinanderreihung von Feistelchiffren. Beschreiben Sie unter zuhilfename geeigneter Sekundärliteratur den prinzipiellen Aufbau eines Feistelnetzwerks. Ziegen Sie, wie die Feistelchiffre in RC5 umgesetzt ist.
- Damit auch Nachrichten verschlüsselt werden können, die nicht ohne Rest in Blöcke aufgeteilt werden können, muss die Nachricht mit sogenanntem *Padding* versehen werden. Recherchieren Sie die Funktionsweise von PKCS#7-Padding.
- Suchen Sie nach alternativen Implementation von RC5/32-CBC und ver- und entschlüsseln Sie Daten mit der jeweils anderen Implementation. Vergleichen Sie auch die Geschwindigkeit Ihrer Implementation mit der alternativen Implementierung.

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 Ihrem Rahmenprogramm eine Funktion

  ```
  void pkcs7_pad(void *buf, size_t len)
  ```

  die einen Block gemäß PKCS#7 auf die Blocklänge auffüllt. Sie können davon ausgehen, dass der Speicher von `buf` groß genug ist, um das Padding unterzurbringen. `len` gibt dabei die Länge vor dem Padding an.

  Hinweis: Sie dürfen diese Funktion in C implementieren.
- Implementieren Sie in der Datei mit dem Assemblercode eine Funktion

```
void rc5_init(unsigned char *key, size_t keylen, void *s, void *l)
```

welche einen Pointer auf einen Key sowie auf einen Speicherbereich, in dem die Rundenschlüssel abgelegt werden, erhält. Diese Funktion führt das Keysetup wie beschrieben durch.

- Implementieren Sie in der Datei mit dem Assemblercode eine Funktion

  ```
  void rc5_enc(uint16_t *buffer, void *s)
  ```

  welche einen Block verschlüsselt.

- Implementieren Sie in der Datei mit dem Assemblercode eine Funktion

  ```
  void rc5_dec(uint16_t *buffer, void *s)
  ```

  welche einen Block entschlüsselt.

- Um Ihre Implementation an einem geeigneten Beispiel zu testen, können Sie die Funktionen

  ```
  void rc5_cbc_enc(unsigned char *key, size_t keylen,
        void *buffer, size_t len, uint32_t iv)
  ```

  sowie

  ```
  void rc5_cbc_dec(unsigned char *key, size_t keylen,
        void *buffer, size_t len, uint32_t iv)
  ```

  aus dem Anhang verwenden.

2.3.3 Zusatzaufgabe

Die im Anhang gegebenen Funktionen implementieren den sogenannten *Cipher Block Chaining*-Modus. Implementieren Sie zusätzlich eine Funktion

```
void rc5_ctr(unsigned char *key, size_t keylen, void *buffer)
```

welche RC5 im Counter-Modus ausführt. Sie können dabei dieselbe Funktion zum Ver- und Entschlüsseln verwenden.

2.4 Allgemeine Bewertungshinweise

Die folgende Liste soll Ihnen als Gedächtnisstütze beim Bearbeiten der Aufgaben dienen. Beachten Sie ebenfalls die in der Praktikumsordnung angegebenen Hinweise.

- Stellen Sie unbedingt sicher, dass Ihre Abgabe auf der Referenzplattform des Praktikums (`lxhalle`) kompiliert und funktionsfähig ist.
- Fügen Sie Ihrem Projekt ein funktionierendes `Makefile` hinzu, welches durch den Aufruf von `make` Ihr Projekt kompiliert.
- Verwenden Sie keinen Inline-Assembler.
- Verwenden Sie SIMD-Befehle, wenn möglich.
- Verwenden Sie keine x87-FPU- oder MMX-Instruktionen. Sie dürfen alle SSE-Erweiterungen bis SSE4.2 benutzen. AVX-Instruktionen dürfen Sie benutzen, sofern Ihre Implementierung auch auf Prozessoren ohne AVX-Erweiterungen lauffähig ist.
- Sie dürfen die Signatur der in Assembler zu implementierenden Funktion nur dann ändern, wenn Sie dies (in Ihrer Ausarbeitung) rechtfertigen können.
- I/O-Operationen dürfen grundsätzlich in C implementiert werden.
- 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.
- Verwenden Sie für die Ausarbeitung die bereitgestellte TEX-Vorlage und legen Sie sowohl die PDF-Datei als auch sämtliche TEX-Quellen in das Repository.
- 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 4:3 als Folien-Format.
- Zusatzaufgaben (sofern vorhanden) müssen nicht implementiert werden. Es gibt keine Bonuspunkte.

3 Anhang

```
#define ROUNDS 16
#define BLOCKSIZE 4
#define HALFBLOCK BLOCKSIZE/2
void rc5_cbc_enc(unsigned char *key, size_t keylen, void *buffer, size_t
    len, uint32_t iv) {
  void *roundkeys;
  void *l;
  uint32_t *padbuf;
  uint32_t *curblock, *lastblock;
  size_t bock_count, i;
  // allokiere Speicherbereich für die Rundenschluessel
  // 2r+2 Schluessel zu je 16 Bit laenge
  roundkeys = malloc((2*ROUNDS+2)*HALFBLOCK);
  l = malloc((2*ROUNDS+2)*HALFBLOCK);

  // Keysetup
  rc5_init(key, keylen, roundkeys, l);

  // allokiere Platz fuer das padding
  padbuf = (uint32_t *)malloc(len + (len % BLOCKSIZE));
  memcpy(padbuf , buffer, len);
  block_count = sizeof(padbuf) / BLOCKSIZE;

  pkcs7_pad((padbuf + (block_count - 1)), BLOCKSIZE - (len % BLOCKSIZE));


  // benutze einen initialization vector
  // um den ersten block zu XORen
  *padbuf ^= iv;
  rc5_enc((uint16_t *)padbuf, roundkeys);
  curblock = padbuf;
  for(i = 1; i < block_count; i++) {
    lastblock = curblock++;
    *curblock ^= *lastblock;
    rc5_enc(curblock, roundkeys);
  }
  free(roundkeys);
  free(l);
}

int rc5_cbc_dec(unsigned char *key, size_t keylen, void *buffer, size_t
    len, uint32_t iv) {
  void *roundkeys;
  void *l;
  uint32_t *padbuf;
  uint32_t *curblock, *lastblock;
  uint32_t lastenc, curenc;
  size_t bock_count, i;
  // allokiere Speicherbereich für die Rundenschluessel
  // 2r+2 Schluessel zu je 16 Bit laenge
  roundkeys = malloc((2*ROUNDS+2)*HALFBLOCK);
```

```
50    l = malloc((2*ROUNDS+2)*HALFBLOCK);
51
52    // Keysetup
53    rc5_init(key, keylen, roundkeys, l);
54
55    if(len % BLOCKSIZE != 0)
56      return -1; // error
57
58    block_count = len / BLOCKSIZE;
59    lastenc = *buffer;
60    rc5_dec((uint16_t *)buffer, roundkeys);
61    *buffer ^= iv;
62
63    // benutze einen initialization vector
64    // um den ersten block zu XORen
65    for(i = 1; i < block_count; i++) {
66      curenc = *++buffer;
67      rc5_dec(buffer, roundkeys);
68      *buffer ^= lastenc;
69      lastenc = curenc;
70    }
71    free(roundkeys);
72    // gibt laenge ohne padding zurueck
73    return len - (*buffer & 0xFF);
74  }
```