LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Aspekte der systemnahen Programmierung bei der Spieleentwicklung

RSA (A504)
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 ARM-Architektur (AArch64) unter Verwendung der NEON-Erweiterungen zu schreiben; alle anderen Bestandteile der Hauptimplementierung sind in C nach dem C17-Standard anzufertigen.

Der **Abgabetermin** ist **Sonntag 05. Februar 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 **14.03.2023 – 24.03.2023** 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

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://asp.caps.in.tum.de`

2 RSA

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

Das RSA-Verfahren ist ein asymmetrisches Kryptosystem. Die Asymmetrie beruht dabei auf der Tatsache, dass es für jeden Teilnehmer einen öffentlichen Schlüssel und einen privaten, geheimen, Schlüssel gibt. Möchte Alice Bob eine verschlüsselte Nachricht schicken, so fragt Sie Bob nach seinem öffentlichen Schlüssel und erstellt damit den Geheimtext. Aufgrund der mathematischen Konstruktion des Kryptosystems ist es sehr schwer, den Geheimtext ohne den privaten Schlüssel zu entschlüsseln.
Ein öffentlicher Schlüssel besteht aus zwei natürlichen Zahlen, dem Tupel (e, N), der dazugehörige private Schlüssel werde mit (d, N) bezeichnet.
Für die Zahlen N, d und e müssen bestimmte mathematische Bedingungen erfüllt sein: Man wählt zufällig zwei Primzahlen p und q und berechnet das Produkt

$$N = pq. \tag{1}$$

Zusätzlich wählt man eine natürliche Zahl mit

$$1 < e < \varphi(N) = (p-1)(q-1) \text{ und } \gcd(e, \varphi(N)) = 1. \tag{2}$$

Dabei bezeichnet $\varphi(N)$ die *Eulersche φ-Funktion* und $\gcd(a, b)$ den größten gemeinsamen Teiler zweier Zahlen a und b. Anschließend berechnet man ein d mit $1 < d < \varphi(N)$ aus e, sodass gilt:

$$d \cdot e \equiv 1 \mod \varphi(N) \tag{3}$$

Da $\gcd(e, \varphi(N)) = 1$, existiert d und kann durch den erweiterten Euklidischen Algorithmus berechnet werden.
Eine Nachricht m kann dann durch die folgende Operation verschlüsselt werden:

$$c = m^e \mod N \tag{4}$$

Der berechnete Wert c ist dann der Geheimtext. Die Entschlüsselung funktioniert analog dazu mit dem privaten Schlüssel d:

$$m = c^d \mod N \tag{5}$$

Ihre Aufgabe ist es, Tripel (e, d, N) zu berechnen, die allen geforderten Kriterien entsprechen. Im Weiteren dürfen Sie davon ausgehen, dass $N < 2^{64} - 1$ gilt.

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.

Wichtig: Sie dürfen im Assemblercode nur solche arithmetischen Operationen verwenden, die grundlegende Berechnungen durchführen (im Zweifel: die vier Grundrechenarten), nicht jedoch Instruktionen, die komplexere Berechnungen durchführen (z.B. Wurzel, Logarithmus, Exponentiation).

2.3.1 Theoretischer Teil

- Verifizieren Sie, dass $(e, d, N) = (3, 147, 253)$ alle oben geforderten Eigenschaften erfüllt. Generieren Sie dann von Hand ein weiteres Tripel, indem Sie die Zahlen auf Papier konstruieren.
- Denken Sie sich ein sinnvolles Verfahren zur zufälligen Generierung von Primzahlen aus. Beschränken Sie die generierten Primzahlen so, dass $N < 2^{64} - 1$ gilt.
- Schlagen Sie in geeigneter Sekundärliteratur nach, wie der erweiterte Euklidische Algorithmus funktioniert.
- Worauf beruht die Sicherheit von RSA? Warum ist diese bei den von Ihnen generierten Zahlen nicht gegeben?
- Untersuchen Sie Ihre fertige Implementierung auf Performanz. Vergleichen Sie Ihre Lösung mit einer alternativen Implementierung in C.

2.3.2 Praktischer Teil

- Implementieren Sie im Rahmenprogramm I/O-Operationen in C, mit welchen Sie die generierten Werte (e, d, N) ausgeben können.
- Implementieren Sie in der Datei mit Ihrem Assemblercode die Funktion:

  ```
  void get_rsa_params(unsigned long* e, unsigned long* d, unsigned long
                                        * N)
  ```

  Die Funktion berechnet zufällig die RSA-Parameter (e, d, N) und schreibt sie in die jeweils vorgesehenen übergebenen Speicherbereiche.

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.
- `-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 Ihre Implementierung auf der Referenzplattform des Praktikums (`HimMUC`) kompiliert und funktionsfähig ist. Überprüfen Sie ebenfalls, ob Ihre Ausarbeitung auf der `lxhalle` fehlerfrei kompiliert.
- Die Implementierung soll mit GCC/GNU as kompilieren. Verwenden Sie keinen Inline-Assembler. SIMD-Implementierungen sind unter Verwendung der NEON-Erweiterung zu schreiben. Andere ISA-Erweiterungen 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.

- 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; 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.