LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Aspekte der systemnahen Programmierung bei der Spieleentwicklung

Gruppe Team 103 – Abgabe zu Aufgabe 504: RSA
Wintersemester 2022/23

Guo Linfeng Özakay Baris Julian Julian

1 Einleitung

Das Praktikum Aspekte der systemnahen Programmierung bei der Spieleentwicklung beschäftigt sich mit dem Programmieren von Prozessen auf niedriger Ebene. Dabei ist es wichtig die Schnittstellen zwischen Hardware and Software effektiv zu nutzen. Man lernt außerdem bei dem Praktikum das Umgehen vom Optimieren von Programmen sowie die Programmiersprache Assembly, AArch64. Assembler ermöglicht die Assemblersprache in Maschinensprache zu übersetzten und das Programmieren in Assembler ermöglicht eine bessere Leistung für das Programm. Diese Verbesserung spielt in vielen Bereichen der Informatik eine wichtige Rolle.

Das Praktikum beinhaltet viele Themengebiete. Ein davon ist die Kryptographie. Kryptographie ist ein essentieller Bestandteil unserer heutigen Kommunikation. Unsere Aufgabe ist es mit dem RSA-Algorithmus zu beschäftigen. Der RSA-Algorithmus wurde im Jahr 1977 zum ersten Mal veröffentlicht und ist nach seinen Erfindern Rivest, Shamir und Adleman ernannt. Die Relevantz des Algorithmus hat in der heutigen Zeit nicht nachtgelassen. Er wird in Bereichen wie zum Beispiel bei Banken, Webservern oder E-Mails, für Sicherheit und Datenschutz benutzt. Das Verfahren wird sehr oft für die Kommunikation mit mehreren Teilnehmern verwendet. Das RSA Verfahren gehört zu dem Public-key cryptography, Asymmetrisches Kryptosystem. Dies bedeutet für die Verschlüsselung der Nachricht wird der öffentlichen Schlüssel verwendet und bei der Entschlüsselung der geheimen Schlüssel.

Ein Beispiel dazu: Person A und Person B wollen mit Hilfe des RSA Verfahrens mit einander kommunizieren. Angenommen alle öffentlichen Schlüssel stehen im Internet. A will eine Nachricht an B verschicken und sucht im Internet den öffentlichen Schlüssel von B heraus. Die Nachricht von A wird anschließend verschlüsselt und an B verschickt. Die verschlüsselte Nachricht kann nur von B verschlüsselt werden, da nur er den geheimen Schlüssel kennt.

Ein Schlüssel ist ein Tupel. Der öffentliche Schlüssel ist aus dem Tupel (e, N) und der geheimen Schlüssel aus dem Tupel (d, N). Dabei ist e der Verschlüsselungsexponent und d der Entschlüsselungsexponent. Für die Wahl dieser Variablen gibt es mehrere mathematischen Voraussetzung. Dabei müssen p und q jeweils Primzahlen sein.

$$N = p \cdot q \tag{1}$$

$$1 < e < \varphi(N) \tag{2}$$

Außerdem muss e Teilerfremd zu φ(N) sein. φ(N) sei die Eulersche φ-Funktion.

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

Nachdem man die Schlüssel generiert hat, kann man die Nachricht m mit Hilfe von e und N verschlüsseln und c ist die verschlüsselte Nachricht. Für die Entschlüsselung wird d verwendet.

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

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

An Hand eines Beispiels sehen wir wie die Nachricht m = 10 verschlüsselt wird. Wir wählen für N = 1363, e = 3 und d = 859. φ(1363) = 1288 und e ist zudem Teilerfremd. Mit (4) ergibt sich die verschlüsselte Nachricht 1000 = $10^3 \mod 1288$. Mit unserem geheimen Schlüssel wollen wir anschließend entschlüsseln. $1000^{859} \mod 1288$ =

Der RSA-Algorithmus besteht aus vier Schritten: Schlüsselgenerierung, Schlüsselverteilung, Verschlüsseln und Entschlüsseln. Unsere Aufgabe ist die Generierung der drei Variablen d, e, N zu implementieren. Wir werden mit der Generierung von N anfangen, gehen anschließend darauf ein wie die Variable e gewählt wird und zum Schluss wie man mit Hilfe von erweiterten Euklidische Algorithmus aus e und φ (N) unser d erzeugt.

2 Lösungsansatz

2.1 Primzahl Generierung

Für die Wahl von N werden zwei Primezahl verwendet, p und q.

2.2 Wahl für e

Der allererste RSA- Algorithmus stammt aus dem Jahr 1977. Diese Variante wurde ursprünglich so implementiert, dass man einen zufälligen d wählt und daraus e generiert. Allerdings ist diese Methode nicht sicher und e sei zu groß gewählt. Denn bei RSA ist je kleiner der Verschlüsselungsexponent e, desto schneller ist die Verschlüsselung. Eine große Zahl kann aber auch für die Verschlüsselung sehr schnell sein und erfüllt dabei auch die Sicherheit. Für die Wahl von e müssen auf ein paar Voraussetzungen geachtet werden. Der Verschlüsselungsexponent muss größer als eins, und kleiner als φ(N) sein(1). Außerdem muss e coprime zu φ(N) sein.

Damit der RSA- Algorithmus bessere Performance leistet und effizienter ist, muss e eine kürze Bit- Länge und ein kleines Hamming Gewicht haben [1]. Das Hamming Gewicht bei einem String von Bits ist die Anzahl von Einsen. Die Bit Länge gibt die Anzahl an Bits, wenn eine Zahl in Binärschreibweise umgewandelt wird. Um diese Länge einer ganzen Zahl zu bestimmen, wird die folgende Formel verwendet:

$$bitLength(n) = \lfloor \log_2(n) + 1 \rfloor = \lceil \log_2(n+1) \rceil \tag{6}$$

Der Grund für eine Verbesserung der Implementierung bei einem Verschlüsselungsexponent e mit einer kürzen Bit- Länge und eines kleines Hamming Gewicht liegt an folgenden Eigenschaften:

1. Weniger Multiplikation: Um eine Nachricht zu entschlüsseln, muss die Nachricht e- Fach mit sich selbst multiplizieren. Daher ist es weniger Rechenaufwand, falls e klein ist und verkürzt somit auch die Berechnungszeit.(3)

2. Weniger Modulo Reduktion: Nach der Berechnung von m^e wird die Operation Modulo N ausgeführt. Wenn e eine kürze Bit Länge hat, so muss weniger Modulo Reduktionen durchgeführt werden. Eine Potenz Modulo Berechnung hat eine Zeitkomplexität von O(log y). Dabei sei y das Exponent.

3. Kleineres Hamming Gewicht: ein kleineres Hamming Gewicht bedeutet, dass die Zahl in der Bit Schreibweise mehr Null- Bits hat. Dies spart ebenfalls Rechnen Aufwand bei der Potenzrechnung und der Modulo Operation.

Die meisten gewählten Zahlen für e sind 65537 und 3. 2^{16} + 1 = 65537 ist eine der meistgewählten Zahl für e. Diese Zahl ist groß genug und kann für die Sicherheit der Verschlüsselung sorgen. Zudem hat diese Zahl ein kleines Hammning Gewicht, in Binärschreibweis 0b10000000000000001. Die kleinste Zahl für e ist die drei. Drei hat ein Hamming Gewicht von zwei, 0b11, und ist somit für die Verschlüsselung sehr schnell. Allerdings ist die Verwendung diese Zahl nicht sicher für das RSA- Verfahren.

Bei unsere Implementierung haben wir zunächst geschaut ob unsere φ(N) größer ist als 65537 oder ob 65537 kleiner gleich φ(N) ist. Falls kleiner gleich wird e = 3 gesetzt. Im anderen Fall ist e = 65537.

2.2.1 Camicael Funktion

Für die Berechnung von φ(N) wird die Eulersche φ- Funktion verwendet. Die φ Funktion gibt die Obereschranke für e an. Es gibt neben der Eulersche φ- Funktion noch eine weitere Funktion, um die obere Schranke für e bestimmt, die Carmicael Funktion, auch λ- Funktion genannt. Diese Funktion berechnet den kleinsten positiven Quotienten der φ- Funktion aus und optimiert somit die obere Grenze. So kann e möglichst klein gewählt werden. Je kleiner e, desto schneller ist die Verschlüsselung. Die Formel für die Carmicael- Funktion sieht wie folgt aus:

$$g^m = 1 \mod n \tag{7}$$

Dabei ist n der Eingabewert der Funktion. g sind jede Teilerfremde Zahl zu n, welchen in der Funktion berechnet wird. m ist der Rückgabewert. Zur Implementierung der Carmicael Funktion haben wir zunächst die gcd()- Funktion als Hilfsmethode implementiert, welche den größten gemeinsamen Teiler zwischen zwei Zahlen bestimmt. Falls das Ergebnis gleich eins ergibt, heißt die zwei Zahlen sind teilerfremd und kann so als g verwendet werden. Die Implementierung des Algorithmus ist aus wenigen Komplexität. Es beinhaltet eine for- Loop, für die Überprüfung von allen teilerfremden

Zahlen die kleiner als n ist und eine while- Loop, die Überprüft bei welchem Exponenten die Eigenschaft von (7) erfüllt wird. Falls eine teilerfremde Zahl gefunden wird, wird nach dem Bestimmen des Exponenten den Rückgabewert k aktualisiert. Dabei wird das kleinste gemeinsame Vielfache zwischen k und des Exponenten berechnet.

2.3 Der erweiterte Euklidische Algorithmus

Bevor wir mit dem erweiterten Euklidischen Algorithmus beginnen, sollten wir zunächst verstehen, was der Euklidische Algorithmus bewirkt. Er ist der effizienteste Weg, um den größten gemeinsamen Teiler GCD zwischen zwei Zahlen zu bestimmen. In unserer Aufgabe müssen wir prüfen, ob die obere Schranke phi(N) und unser Verschlüsselungsexponent e teilerfremd sind. Die grundlegende Idee des Euklidischen Algorithmus ist, dass der GCD von zwei ganzen Zahlen derselbe ist wie der GCD der kleineren Zahl und dem Rest der größeren Zahl, die durch die kleinere Zahl dividiert wird. Das Verfahren ist wie folgt:

1. Die Funktion bekommt zwei Eingaben, a und b. Daraus berechnet man sich r, welcher den Rest von der Division zwischen a und b bildet.
2. Falls r gleich null ist, ist der größte gemeinsame Teiler von a und b gleich b und wir geben ihn als Ausgabe aus.
3. Falls r nicht gleich null ist, ersetzen wir a durch b und b durch r und beginnen erneut beim Schritt 1.

Wie der Name schon verrät, ist der erweiterte Euklidische Algorithmus eine Erweiterung des ursprünglichen Euklidischen Algorithmus. Beide Verfahren berechnen den GCD zwischen zwei ganzen Zahlen. Der erweiterte Algorithmus gibt uns außerdem noch den Koeffizienten der Bezout-Identität an. Es sind zwei ganze Zahlen x und y und stehen wie folgt im Zusammenhang:

$$ax + by = gcd(a, b) \tag{8}$$

Diese Koeffizienten werden verwendet, um die modulare Inverse einer Zahl a $\mod$ b zu finden, die $a^{-1} \mod$ b ist. Die Implementierung vom EEA funktioniert wie folgt. Wir initialisieren zuerst x = 1 und y = 0. Diese sind unsere Koeffizienten der Bezout-Identität. Wir führen den euklidischen Algorithmus aus. Bei jedem Schritt werden außerdem die zwei Koeffizienten x und y mit folgender Formel aktualisiert.

$$x_{new} = y, y_{new} = x - (floor(r/s)) \cdot y \tag{9}$$

Die Variablen r und s sind Reste und Divisor des euklidischen Algorithmus. Die Funktion floor() gibt die größte Zahl, die kleiner als der Eingabewert ist, zurück. Wenn der euklidische Algorithmus terminiert, erhälten man das Ergebnis von GCD und die zwei Koeffizienten der Bezout-Identität, x und y. Für die Modulo Inverse von a $\mod$ b kann man mit Hilfe von x $\mod$ b berechnen, falls gcd(a,b) = 1 ist. Im anderen Fall gibt es keine Modulo Inverse.

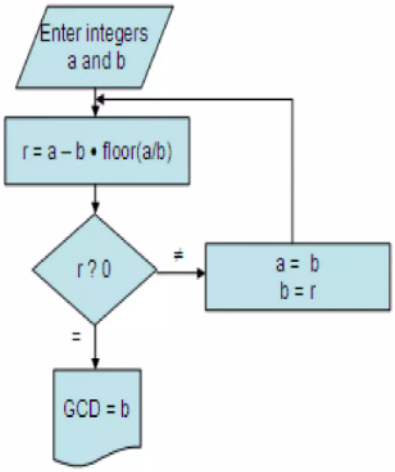

Abbildung 1: Funktionsweise von EEA

In unsere Aufgabe sieht die Gleichung folgendermaßen aus:

$$ed + \varphi(N)y = gcd(e, \varphi(N)) = 1 \tag{10}$$

Mit Hilfe des EEAs und den bekannten öffentlichen Schlüssel (e, N) wollen wir nun den privaten Schlüssel (d, N) berechnen. Mit einer Umformung:

$$ed = 1 mod(\varphi(N)) \tag{11}$$

Lässt sich auch d aus e und φ (N) bestimmen. Wir können den EEA nicht nur für die Berechnung von GCD zwischen e und φ(N) verwenden, sondern auch die Modulo Inverse vom Publishen Schlüssel. Wobei das Ergebnis den privaten Schlüssel d ergbit.

2.4 Sicherheit von RSA

Bereits bei dem EEA-Teil haben wir die Formel ed = 1 mod φ(n) vorgestellt. Dabei sei (e, N) uns bekannten öffentlichen Schlüssel. Um die Verschlüsselte Nachricht knacken zu können, muss man wesentlich nur den privaten Schlüssel d herausfinden.

$$d = e^{-1} \mod \varphi(n) \tag{12}$$

Dabei sei φ(N) wie folgt definiert:

$$\varphi(n) = \varphi(p)\varphi(q) = (p-1) \cdot (q-1), n = p \cdot q \tag{13}$$

Die Variable e sei uns bereits bekannt. Der schwierige Teil liegt wesentlich daran φ(n) herauszufinden. Mit anderen Worten muss man die zwei Primfaktoren p und q für n finden. Die Primfaktorzerlegung von einer ausreichend große Zahl zu finden, sei fast unmöglich. Im praktischen würde so ein Algorithmus nicht in polynomiale Zeit laufen.

Mit dem „Schroeppel factoring algorithm“ wird die faktorisier Zeit bei unterschiedlichen n- Länge dargestellt. Bereits bei einer Länge von 75 kann es zu mehrere Monate führen. Selbst der heutzutage meistverwendete Primfaktorzerlegungsalgorithmus, Trial division, hat eine Laufzeit von O(n^2). Somit ist die Sicherheit vom RSA hauptsächlich auf der Schwierigkeit von der Faktorisieren basiert.

Länge n	Anzahl Operationen	Zeit
50	1.4×10^{10}	3.9 Stunden
75	9×10^{12}	104 Tage
100	2.3×10^{15}	74 Jahre
200	1.2×10^{23}	3.8×10^{9} Jahre
300	1.5×10^{29}	4.9×10^{15} Jahre

Tabelle 1: Laufzeit Schroeppel factoring

Ein weiterer Punkt wovon die Sicherheit von RSA auch abhängig ist, ist die Wahl von e. In Sektion 2.3 wurde erwähnt, dass die kleinste Zahl für e die Drei ist. Auch wenn der Verschlüsselungsexponent bei e = 3 sehr schnell verläuft, ist die Sicherheit nicht vielversprechend. Im Fall, dass m^e kleiner als N ist und auch wenn die Verschlüsselung der Nachricht sehr schnell verläfut, bedeutet es mit großen Wahrscheinlichkeit, dass der Verschlüsselungsexponent die drei ist. Solcher Angriff wird auch als Timing Attack genannt. Der Angreiffer benutzt die Zeit für die Verschlüsselung der Nachricht, um Informationen über den Verschlüsselungsprozess zugewinnen. Das Verfahren was wir implemenmtiert haben wird auch als Textbook RSA genannt. Es ist RSA ohne Padding. Das Verfahren verwendet viele mathematische Operationen und kann zu manchen Schwächen führen. Um diese zu vermeiden wird das Padding für RSA hinzugefügt. Ein sehr häufigverwendetes Padding Verfahren für RSA ist das öptimal asymmetric encryption padding".

3 Korrektheit/Genauigkeit

4 Performanzanalyse

5 Zusammenfassung und Ausblick

In dieser Projektarbeit hatten wir die Aufgabe mit dem Veerschlüsselungsverfahren RSA, welches in der heutigen Zeit immer noch oft verwendet wird, zu beschäftigen. Dabei sei unsere genaue Aufgabe die Schlüsselgenerierung von RSA in Assembly zu Implemetieren. Neben den ganzen Implementierungen sollen wir außerdem noch ein Rahmenprogramm in C schreiben. Das Rahmenprogramm soll unterschiedlichen Eingabenoptimierungen annehmen, sowie eine Hilfestellung für die Funktionsweise des Programmes. Auch eine Performanzanalyse haben wir durchgeführen müssen. Mit Hilfe

davon konnte man festestellen feststellen wie die Laufzeit mit Hilfe vom Assembly Code optimiert werden.

Für die Implementierung der Schlüsselgenerierung haben wir zunächst das Problem in drei Teile aufgeteilt. Wir haben mit der Variable N angefangen. Dabei sei die schwierigkeit zwei zufällige Primezahl zu generieren. Anschließend kommt die Wahl von e in Frage. Zum