25. OBERSCHULE „AM POHLANDPLATZ“ DRESDEN

Dokumentation zum „RSA-Tool“

eine Informatikabschlussarbeit
der 25. Oberschule „Am Pohlandplatz“
Dresden

von Alec Schitzkat und Karl R. Jahn

Dresden, den 02.04.2022

Das RSA-Tool ist eine kleine Windowsanwendung, welche zur Visualisierung der RSA-Verschlüsselung dient, und hierbei gleichzeitig heutigen Sicherheitsstandards entsprechen soll. Umgesetzt wurde dies in C++ (ISO C++ 20, MSV) und C# (Windows Forms, .NET Framework), welche über eine Kommandozeilenschnittstelle einseitig kommunizieren können. C# übernimmt hierbei die Darstellung um den Benutzer das Prinzip verständlich herüberzubringen. Dies ist die Dokumentation zu beiden Anwendungen und den Projektablauf/Schaffungsprozess.

1 INHALTSVERZEICHNIS

2 IDEE

Im Zeitalter der Digitalisierung wird die Sicherheit dieser digitalen Daten zunehmend wichtiger. Deshalb ist es sinnvoll, sich mit ebendieser zu beschäftigen und so kam uns die Idee zum einen anderen dabei behilflich zu sein und zum anderen selber natürlich dazuzulernen, indem wir dies mit dem „RSA-Tool“ ermöglichen.

Das RSA-Tool soll in der Lage sein, das RSA-Verfahren durchzuführen (und dies nach heutigen Standard) und es gleichzeitig zu visualisieren, sodass der Benutzer in der Lage ist dieses augenscheinlich trockene Verschlüsslungsverfahren nachvollziehen und verstehen zu können.

3 MATHEMATISCHE HINTERGRÜNDE

3.1 PRIMZAHLEN

Primzahlen sind die Bausteine der natürlichen Zahlen. Über die Verteilung und Eigenschaften dieser Rätselt die Menschheit seit langer Zeit. Die wichtigste Eigenschaft ist die, welche sie definiert, die Teilbarkeit und zwar sind sie nur durch sich und sich selbst (im Bereich der natürlichen Zahlen) teilbar.

3.1.1 Einwegfunktionen

Ein großes Problem, das bis jetzt nicht gelöst wurde ist die effiziente Faktorisierung von großer Zahlen. Dieses Problem lässt sich für ein wichtiges Prinzip der Informatik verwenden, die Einwegfunktionen. [QI03]

> *In der Informatik ist eine Einwegfunktion eine mathematische Funktion, die komplexitätstheoretisch „leicht" berechenbar, aber „schwer" umzukehren ist. In einem erweiterten Sinn werden auch Funktionen so bezeichnet, zu denen bisher keine in angemessener Zeit praktisch ausführbare Umkehrung bekannt ist.*
>
> *Ein anschauliches Beispiel wäre ein klassisches Papier-Telefonbuch einer größeren Stadt: Kennt man den Namen, dann findet man sehr schnell die dazugehörige Telefonnummer. Kennt man jedoch nur die Telefonnummer, so ist es sehr aufwändig, den zugehörigen Namen zu finden.*
>
> *Einwegfunktionen bilden die Grundlage asymmetrischer Kryptosysteme.*

Quelle: Wikipedia (https://de.wikipedia.org/wiki/Einwegfunktion); Stand 02.04.2022

3.1.2 Primzahlen finden / Primzahlentest [1]

Trotz dessen ist es heutzutage möglich in einer Angemessen große Primzahlen von über 600 Dezimalstellen zu finden. Hierzu gibt es viele Möglichkeiten, wie z.B. die Probedivision, das Sieb des Eratosthenes, oder die verbesserte Variante dessen, das Sieb von Atkin, jedoch kommen diese Aufgrund Ihres Speicheraufwandes nicht infrage, um Primzahlen dieser Größe zu generieren.

Ich stützte mich bei der Generierung von Primzahlen auf probabilistische Primzahltests, die zwar auch „falsche" Primzahlen finden können, aber vereint, die Wahrscheinlichkeit dafür verschwindend gering ist. So nutze ich eine Kombination von Tests, die auf den kleiner fermatscher Satz, den Satz nach Miller und einer stark beschränkten Probedivision beruht.

So lassen sich durch Probedivisionen die Vielfachen der ersten Primzahlen heraussieben. Die Übriggebliebenen kann man dann mit den kleinen Fermat (siehe unten) filtern und zu Letzt, Gewissheit durch den Miller-Rabin-Test (übernimmt die Bibliothek, folglich hier nicht mehr dazu) schaffen.

wenn	d.h., wenn
$p \in \mathbb{P}$	$p \in \mathbb{N}$
$a \in \mathbb{N}; 1 < a < p$	$a \in \mathbb{N}; 1 < a < p$
dann gilt:	$a^{p-1} \bmod p \neq 1$
$a^{p-1} \equiv 1 \bmod p$	dann gilt:
	$p \notin \mathbb{P}$

kleiner Satz von Fermat und Folgerung zum Primzahlentest

[1] siehe [QI04] & [QI05]

3.2 DAS RSA-VERFAHREN

Das RSA-Verfahren, welches nach dessen Entwicklern Rivest, Shamir und Adlema benannt wurde basiert auf Zahlentheoretischen Problemen (u.a. das in Abschnitt 3.1 erwähnte) und ist ein asymmetrisches Verfahren, d.h. es gibt einen privaten und einen öffentlichen Schlüssel (oder in den Fall sogar zwei). [QI01]

3.2.1 Schlüsselerzeugung [2]

Ablauf:

1. Wähle zwei unterschiedliche riesige (min. 600 Dezimalstellen, also 2048 Bytes) Primzahlen p und q.

   $p \in \mathbb{P}$
   $q \in \mathbb{P};\ p \neq q$

2. Berechne das RSA-Modul aus p und q.

   $N \coloneqq p * q;\ N \in \mathbb{N}$

3. Berechne die Eulersche φ-Funktion aus N.

   Eulersche φ-Funktion: $\varphi(m) = |\{ x \in \mathbb{N} \mid 1 \leq x \leq m \ \wedge\ \mathrm{ggT}(x, m) = 1 \}|$
   $\varphi(N) = (p-1)(q-1)$
   da $\varphi(p) = p - 1$, $\varphi(q) = q - 1$ und $N = p * q$

4. Wähle eine zu $\varphi(\mathrm{N})$ teilerfremde Zahl e.

   $e \in \mathbb{N};\ 1 < e < \varphi(N) \ \wedge\ \mathrm{ggT}\big(e, \varphi(N)\big) = 1$

5. Berechne d, welches das multiplikative Inverse zu e bezüglich $\mathrm{mod}\ N$ darstellt. Dies geht über den erweiterten euklidischen Algorithmus.

   $d \in \mathbb{N};\ d * e \bmod N = 1$

6. Fertig! Die öffentlichen Schlüssel sind nun (e, N) und das private Schlüsselpaar besteht aus (d, N). $\varphi(N)$, p und q werden nun nicht mehr gebraucht.

Der C++ - Quellcode herfür befindet sich in der Datei TODO.cpp, sowie ein kleiner Auszug dessen ist im Anhang 2 „Code“, Abschnitt 3 zu finden.

3.2.2 Verschlüsselung

Zur Verschlüsslung der Nachricht wird das öffentliche Schlüsselpaar (also $(e,\ N)$) benötigt. Dieses muss sich derjenige, der die Nachricht versenden möchte zuerst vom Empfänger anfordern. Nun muss der zu verschlüsselnde Text, bzw. die Nachricht als Zahl Codiert werden, dies erfolgt zumeist über den ASCII-Code. Da die Nachricht aber einer Häufigkeitsanalyse zu Opfer fallen kann, werden nun die Codierten Zeichen zu Blöcken zusammengefasst, meist werden hierfür zwei bis vier Zeichen zu einen Block zusammengefasst. Zu beachten ist hierbei, dass die Primzahl auch wirklich groß genug ist, da die Zahl kleiner als das RSA-Modul, N, sein muss.

Nach dieser kleinen Vorarbeit kann nun mit der Verschlüsselung begonnen werden. Hierfür wird jeder Zahlenblock verschlüsselt, indem man diesen Block mit den Exponenten e Potenziert und das N-te Modul daraus zieht. Dieses Verfahren lässt sich durch die binäre Exponentiation beschleunigen.

[2] siehe [QI01], [QI02] & [VI01] – [VI04]

$m \in \mathbb{N};\ \ m \coloneqq \text{Nachrichtenblockkodierung}$
$c = c^e \bmod N$

Der C++ - Quellcode herfür befindet sich in der Datei TODO.cpp, sowie ein kleiner Auszug dessen ist im Anhang 2 „Code“, Abschnitt 2 zu finden.

3.2.3 Entschlüsselung

Zur Entschlüsselung braucht der Empfänger logischer Weise seinen privaten Schlüssel. Nun wird das Verschlüsselungsverfahren umgedreht wiederholt. D. h. zuerst wird die Nachricht, bzw. der Nachrichtenkodierungsblock als Basis zum Exponenten d potenziert und das N-te Modul hieraus gezogen. Danach muss der der Block nur noch „auseinandergestückelt“ und decodiert werden und das war‘s.

$c \in \mathbb{N};\ \ c \coloneqq \text{verschlüsselter Nachrichtenblock}$
$m = c^d \bmod N$

Der C++ - Quellcode herfür befindet sich in der Datei TODO.cpp, sowie ein kleiner Auszug dessen ist im Anhang 2 „Code“, Abschnitt 3 zu finden.

4 VERSCHLÜSSELUNGSPROGRAMM (C++, CMD-LINE)

Das „Verschlüsselungsprogramm“ ist das eigentliche Programm, denn es generiert die Schlüssel, schreibt und liest die Schlüsseldateien, und ver-, bzw. enschlüsselt, wobei es sich hierbei auch um die Dateien kümmert.

4.1 BIBLIOTHEKEN

Ein großer Nachteil von C++ ist, dass es (bis jetzt, ISO C++ 20) nicht in der Lage ist Standartmäßig Integer, die größer als 64, bzw. 128 Bit sind, geeignet darzustellen und zu handeln. So kamen wir nicht darum herum externe Bibliotheken einzubinden, oder aber selber die Möglichkeit BigInts nutzen zu können zu schaffen, was jedoch den Projektrahmen völlig gesprengt hätte – und zumal ineffizient gewesen wäre –, weshalb wir uns für ersteres Entschieden.

4.1.1 Boost

Also banden wir Boost (boost.org), ein mächtiges C++-library, ein, was wir uns auch bei der Primzahlengenerierung zunutze machten.

Insgesamt aus Boost eingebundene Header:

Header	Nutzen
`<boost/multiprecision/cpp_int.hpp>`	• größere Zahlen verwenden • mit diesen arbeiten/rechnen
`<boost/multiprecision/miller_rabin.hpp>`	• den Miller-Rabin-Test als Teil des Primzahlenfindungsalgorithmus verwenden

4.1.2 C++ Standard

Header	Nutzen
`<iostream>`	• Konsolen Ein/Ausgabe
`<fstream>`	• Dateiströme (Dateien lesen und schreiben)
`<thread>`	• Mutithreading

4.2 PRIMZAHLENERZEUGUNG

Große Primzahlen schnell zu erzeugen ist eine nicht so einfache Sache. Der Algorithmus, welchen wir hierzu verwenden ist dementsprechend vierteilig gestaltet, um eine annehmbare Performanz und gleichzeitig notwendige Sicherheit, dass es sich auch wirklich um eine Primzahl handelt zu gewährleisten.

4.2.1 Algorithmus

1. Zuerst wird eine „zufällige“ [3], ungerade Pseudoprimzahl n und eine zweite Zufallszahl a erzeugt. Für a gilt $1 < a < n$, was über $a \; mod \; (n-2)+1$ zu erreichen ist, es wird für den späteren Primzahlentest benötigt. Daraufhin wird n solang mit zwei Inkrementiert, bis es sich um eine Primzahl handelt.

[3] Zufall zu über Computer zu erzeugen ist unmöglich, meist handelt es sich hierbei um ein Modul einer Multiplikation eines „zufälligen“ Seeds, der im Standartfall entweder die Zeit, rauschen, oder ein im Prozessor gemessener Wiederstand ist. In diesen Fall nutzen wir das häufigste Verwendete, die Zeit, der Generator jedoch erzeugt einen zufälligen Bitstrom.

Jedoch haben wir dies etwas abgewandelt und überschreiben e, wenn nach 50 Iterationen sich immer noch um keine Primzahl handelt mit einer neuen Pseudoprimzahl (die wiederum einen neuen Seed hat).

2.

5 UI UND VISUALISIERUNG (C#, WINDOWSFORMS)

6 INTERFACE DER BEIDEN ANWENDUNG

Die Schnittstelle beider Anwendung, findet, wie bereits schon in den beiden vorherigen Abschnitten angedeutet über die „Kommandozeile“ und Dateien statt. Dies hat den Vorteil, dass das C++-Tool auch für andere Programme verwendet werden kann und so auch die Benutzung dessen von anderen Programmen für realistischere Anwendungsfälle in Betracht gezogen werden kann. Über den Programmnamen und eines der Argumenten „--h“, „--help", oder auch „/h“ oder „/help“, erhält der Benutzer des C++-Tools im CMD die in Bild TODO zu sehende Übersicht valider Argumente und der Benutzung des C++-Tools.

7 SCHAFFUNGSPROZESS

8 CODEAUSSCHNITTE

9 QUELLEN

[QI01] https://de.wikipedia.org/wiki/RSA-Kryptosystem
Stand 02.04.2022; Zugriffsort: Deutschland

[QI02] https://tools.justus-d.de/rsa/
Stand 02.04.2022; Zugriffsort: Deutschland

[QI03] https://de.wikipedia.org/wiki/Einwegfunktion
Stand 02.04.2022; Zugriffsort: Deutschland

[QI04] https://de.wikipedia.org/wiki/Primzahltest
Stand 02.04.2022; Zugriffsort: Deutschland

[QI05] https://de.wikipedia.org/wiki/Kleiner_fermatscher_Satz
Stand 02.04.2022; Zugriffsort: Deutschland

[VI01] https://www.youtube.com/watch?v=j2NBya6ADSY
Stand 02.04.2022; Zugriffsort: Deutschland

[VI02] https://studyflix.de/informatik/rsa-verschlusselung-1608
Stand 02.04.2022; Zugriffsort: Deutschland

[VI03] https://www.youtube.com/watch?v=gFd_SZZF63I
Stand 02.04.2022; Zugriffsort: Deutschland

[VI04] https://www.youtube.com/channel/UC_FGVqET9-GHgKZ7G0ejTSA
Stand 02.04.2022; Zugriffsort: Deutschland