Klausurersatzleistung RSA-Algorithmus

Beschreibung der Datenstruktur sowie Erläuterung der Vor- & Nachteile

Von Anton Herzog und Christopher Glania.

In unserem Projekt rsa\_mts.EXE verwenden wir zur Speicherung der Bytes der textfile.txt, sowie der daraus konvertierten Integers, hauptsächlich Arrays, da C# jede File in ein Bytearray umwandeln kann. Die Länge des Bytearrays bleibt im gesamten Programmablauf unveränderlich, da die Information je Byte verändert bzw. verschlüsselt wird.

Um die textfile.txt Datei einzulesen, verwenden wir die Klasse File (using System.IO), dieser wird der Dateipfad übergeben. Natürlich kann man das auch sehr schnell universeller gestalten, uns geht es jedoch nur um die Ver- bzw. Entschlüsselung. Dies gilt auch für die Auswahl der Primzahlen, welche aktuell konstant verwaltet werden. Für eventuelle Änderungen sind jedoch bereits Methoden eingebaut welche den Absturz des Programms verhindern oder zu falschen Ergebnissen führen würden. Beispielsweise die Methode Ggt, welche überprüft ob e und PhiN teilerfremd sind. Da es vorkommen kann, dass beim errechnen von Zwischenwerten, die Kapazitäten eines Integers überstiegen werden rechnen wir die einzelnen Integerwerte beim direkten Ver- oder Entschlüsseln in BigIntegerwerte um. In einigen Aufgabenstellungen sind Integers gefordert, deshalb wandeln wir unsere BigIntegerwerte nach den Rechenoperationen wieder in Integers zurück und geben diese sowie die entsprechenden Bytes in der Konsole aus.

Zusätzlich werden Verschlüsselter Text und Entschlüsselter Text in separaten Textdateien ausgegeben. Zu finden sind diese im Ordner „...rsa\_mts\bin\x86\Debug“.

Bei der Ausgabe des verschlüsselten Textes wird zunächst ein Leerer String Initialisiert, in welchen dann die einzelnen Integerwerte + Leerzeichen geschrieben werden. Nach jedem 7. Integerwert wird ein Return eingefügt. Dieser String wird dann an die File.WriteAllText(string Dateipfad) Methode übergeben.

Beim Einlesen des verschlüsselten Textes oder bei uns „encrypted.txt“, speichern wir den gesamten Text zunächst in einem String mittels der Mtehode File.ReadAllText(string Dateipfad) ab. Danach ersetzen wir alle Enterzeichen mit einem ““. Leerzeichen werden im Anschluss beim Methodenaufruf von „string.Split(string text)“ automatisch entfernt und wir erhalten ein Stringarray zurück. Da dieses jedoch noch die unbestimmte anzahl von ““ enthält und wir somit das Stringarray noch nicht in ein Integerarray konvertieren können, entfernen wir diese mittels einer List<int>, denn dabei muss anfangs keine Größe feststehen. Alles was sich nun in ein Integer parsen lässt wird in die Liste aufgenommen, welche Im Anschluss in ein Integerarray gewandelt wird. Nun können wir wieder ganz normal damit weiterarbeiten. Die Vor und Nachteile der jeweiligen Datenstrukturen sind auf der Folgeseite aufgelistet.

BigInteger Vorteile:

* Repräsentation beliebig großer ganzer Zahlen
* viele nützliche Methoden bereits implementiert
* Zahlbereich praktisch unbegrenzt groß, da BigInteger auf Arbeitsspeicher arbeitet

BigInteger Nachteil:

* Vergleichs- und Rechenoperationen aufwändig

Array Vorteile:

* Schneller direkter Zugriff auf Elemente
* Einfache Handhabung und Deklaration
* Arrays sind generisch

Array Nachteile:

* Größe muss bei Initialisierung festgelegt werden
* wenig vorimplementierte Methoden
* hoher Aufwand für späteres hinzufügen von Elementen

List<> Vorteile:

* ist praktisch beliebig erweiterbar
* bietet reichlich vorimplementierte Methoden
* flexibler als Arrays
* arbeiten oft performanter als Arrays

List<> Nachteile:

* sind in der Praxis meist nicht Relevant

Filereader / Filewriter Vorteile:

* einfach zu initialisieren und zu benutzen
* bietet bereits vorimplementierte Methoden

Filereader / Filewriter Nachteile:

* Nicht bekannt