Algorithmen und Datenstrukturen

Sommersemester 2022

Prof. Dr. Philipp Jenke

HAW HAMBURG | Department Informatik

Aufgabe 4

Dieses Aufgabenblatt baut auf den Vorlesungen auf:

- Hashing,
- Kryptologie,
- Datenkompression.

Deliverables:

- PDF mit Lösungen zu den Theorie-Aufgaben
- Code für Praxisaufgabe (Code-Konventionen, Tests)

1 Theorie

1.1 Hashing

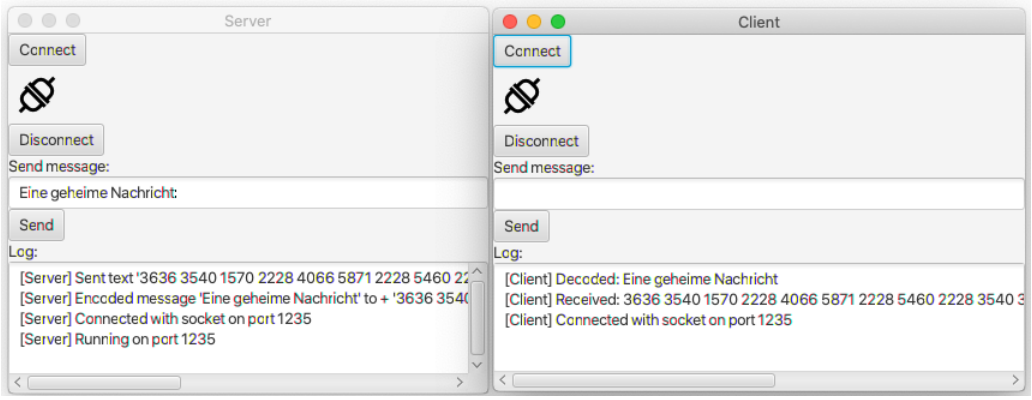
Fügen Sie die Schlüssel $1, 2, 3, 8, 9$ in eine Hashtabelle der Länge 5 ein. Verwenden Sie Quadratisches Sondieren mit der Divisionsrest-Methode. Geben Sie jeweils an, wenn und wenn ja wie Sie bei Kollisionen alternative Hashindizes bestimmt haben.

1.2 Huffman Codierung

Wir betrachten das Alphabet w, x, y, z. Die Zeichen kommen mit den folgenden Wahrscheinlichkeiten vor: $0.12, 0.21, 0.3, 0.37$. Generieren und zeichnen Sie den Baum der Huffman-Codierung und geben Sie für jedes Zeichen im Alphabet die Codierung an. Geben Sie außerdem die Zwischenschritte beim Aufbau des Baumes an.

2 Praxis

In diesem Aufgabenblatt implementieren Sie eine verschlüsselte Kommunikation zwischen einer Client- und einer Server-Anwendung.



2.1 RSA-Schlüssel

Zunächst müssen Sie den RSA-Algorithmus implementieren. Dazu benötigen Sie RSA-Schlüssel. Generieren Sie zuerst zwei Primzahlen p und q. Schreiben Sie dazu eine Methode, die in einem vorgegebenen Intervall eine zufällige Primzahl findet. Ein Primzahltest für eine Zahl x prüft einfach, ob es Teiler außer 1 und x gibt (dann ist es keine Primzahl). p und q müssen unterschiedlich sein. Für unsere Anwendung reicht es, wenn die Zahl aus dem Intervall $[50, 150]$ kommt.

Aus p und q berechnen Sie n und bestimmen dafür die Eulersche ϕ-Funktion $\phi(n)$. Für e wählen Sie eine weitere Primzahl aus dem Intervall $]\phi(n), 1.5 \cdot \phi(n)]$. Schließlich berechnen Sie d als modulare Inverse von e bezüglich $\phi(n)$. Für diese Berechnung können Sie den erweiterten Algorithmus von Euklid verwenden. Alternativ können Sie in der Klasse `BigInteger` nach passenden Hilfsroutinen suchen.

Der Schlüssel ergibt sich dann als (n, e, d) (beinhaltet öffentlichen und privaten Anteil).

2.2 Ver- und Entschlüsselung

Schreiben Sie zwei Methoden `encode()` und `decode()`, die mit einem RSA-Schlüssel eine Zahl codieren, bzw. dekodieren. Potenzen mit großen Exponenten mit `int` zu berechnen, läuft schnell aus dem Ruder. In Java gibt es in der Klasse `BigInteger` Hilfsfunktionen, die einem dieses Problem abnehmen, insbesondere `modPow()`.

2.3 Verschlüsselte Client-Server-Kommunikation

Im Vorgabeframework finden Sie eine Client-Server-Anwendung. Diese erlaubt es, einen Client und einen Server miteinander zu verbinden und jeweils Nachrichten auszutauschen. Vor dem Verschicken müssen Sie Client und Server verbinden (Klick auf *Connect*). Die Nachrichten werden im Klartext verschickt. Sie müssen daher zur Verschlüsselung in der Klasse `ClientServerApplication` (Konstruktor) vier unäre Operatoren setzen, die diesen Klartext vor dem Senden verschlüsseln und nach dem Empfangen wieder entschlüsseln. In der Vorgabe wird hier einfach die Identität verwendet (keine Verschlüsselung). Generieren Sie zwei RSA-Schlüssel, einen für den Client und einen für den Server. Beim Senden codiert der Client mit dem (öffentlichen) Schlüssel des Servers, der decodiert beim Empfangen mit seinem (privaten) Schlüssel. Wenn der Server etwas zum Client sendet, wird der Schlüssel des Clients verwendet.

Wir gehen davon aus, dass die Nachrichten nur aus Klein- und Großbuchstaben bestehen. Jeder Buchstabe wird in eine Zahl verwandelt (Index in der ASCII-Tabelle) und jede Zahl wird einzeln verschlüsselt/entschlüsselt. Die codierten Zahlen werden vor dem Übertragen zu einem String zusammengefasst, zwischen je zwei Zahlen wird ein Leerzeichen eingefügt. Beim Entschlüsseln kann die übertragene Botschaft an den Leerzeichen getrennt werden und dann wieder jede Zahl einzeln entschlüsselt und zurück in einen Buchstaben übersetzt werden.