RSA Dokumentation

I. Mit eigenen Worten erklären

Beim RSA-Verschlüsselungverfahren wird die Nachricht mit einem öffentlichen Schlüssel verschlüsselt. Zum Entschlüsseln verwendet man den privaten Schlüssel, welcher geheim bleibt. Da es zwei unterschiedliche Schlüssel gibt handelt es sich um ein asymmetrisches Verfahren.

Außerdem werden die Schlüssel so gewählt, dass es praktisch unmöglich ist, den privaten aus dem öffentlichen Schlüssel zu bestimmen, wodurch RSA als besonders sicher gilt. (Im Detail siehe III)

II. Inhaltlicher Aufbau

Das Programm ist zum Großteil in Funktionen strukturiert, außer
dass beim Start des CLI Programmes durch *sys.argv* die command-line arguments gelesen und dementsprechend entweder

- (a) ein neues Schlüsselpaar generiert
- (b) die Nachricht verschlüsselt
- (c) die Nachricht entschlüsselt
  wird.

Bei (a) läuft der in III. beschriebene Prozess zum generieren der Schlüssel ab, welche anschließend als *keys.txt* gespeichert werden.
Diese werden dann in entsprechend in (b) und (c) gelesen, und jeder Buchstabe im Eingabetext wird anhand des Ascii-Wertes durch die jeweilige Formel verschlüsselt.

Um den verschlüsselten Text besser darzustellen, wird er zusätzlich Base64 encoded.

Bei der GUI Version wird das Generieren des Schlüssels von JavaScript aus angefragt und dort gespeichert. Das gleiche gilt für Ver- und Entschlüsselung.

III. Mathematische Grundlage zuordnen

1. Zwei zufällige, unterschiedliche Primzahlen *p* & *q* werden generiert → *generate_keys()* (vgl. l. 55) ruft hierfür die Funktion *make_primes()* auf, welche mit *rand_bits(n)* so lange zwei zufällige Zahlen von n-Bits erstellt, bis diese zwei verschiedene Primzahlen, laut *is_prime(n)*, sind.
2. Als nächsten werden in *generate_keys()* durch die Formeln $n = pq$ und $m = (p-1)(q-1)$ berechnet
3. Durch die Funktion *random_coprime(m)* wird eine Zahl *a*, die Teilerfremd zu *m* ist, generiert
4. Das multiplikative Inverse von *a mod m* wird durch *d = pow(a, -1, m)* gezogen. Falls d = a ist, ruft sich die Funktion rekursiv selbst auf, bis d != a.
5. Verschlüsselung durch die Formel $c = x^a \mod n$ → In *encrypt(plain)* wird der Schlüssel gelesen und dann jeder Buchstabe des Eingabetextes verschlüsselt und wieder zusammengesetzt.
6. Entschlüsselung durch $p = c^b \mod n$ → In *decrypt(cipher)* wird wieder der Schlüssel gelesen, die Formel auf jeden Buchstaben des Eingabetextes angewandt.

IV. Erfahrungen

Besonders schwierig war

- herauszufinden, wie man den Text in ein numerisches Format zur Verschlüsselung transformieren kann
- effizient mit großen Zahlen zu rechnen. Daher verwendet der Key gerade nur 8 Bit. (würde z.B. den Miller-Rabin Algorithmus benötigen)

- mathematische Konzepte wie *relativ prim* programmatisch zu implementieren

github