Aufgabenblatt Kryptografie mit öffentl. Schlüsseln

Lösen Sie die nachfolgenden Aufgaben und bereiten Sie diese bis zum nächsten Lehrveranstaltungstermin vor.

LB-KÖS 00. (nicht abzugeben)

Schreiben Sie ein Programm unter Zuhilfenahme der *GMP*, das zwei als Argumente übergebene Zahlen multipliziert und das Ergebnis ausgibt. Überprüfen Sie Ihr Programm mit den Beispielzahlenpaaren `2 3` und `x x`, wobei `x` die größtmögliche Zahl ist, die in einem vorzeichenbehafteten 32-Bit-Integer gespeichert werden kann.
Hinweis: Bauen Sie auf dem zur Verfügung gestellten Template für GMP *auf.*

LB-KÖS 01.

Die Verschlüsselung c einer Nachricht m mit dem öffentlichen Schlüssel (e, N) ist in RSA wie folgt definiert: $c \equiv m^e \mod N$. Die Entschlüsselung mit dem privaten Schlüssel (d, N) ist analog wie folgt definiert: $m \equiv c^d \mod N$.
Schreiben Sie ein Programm unter Zuhilfenahme der *GMP*, das eine Ver- oder Entschlüsselung mit RSA implementiert. Das Programm soll drei Parameter entgegennehmen: die zu ver- bzw. entschlüsselnde Nachricht m **als Zahl**, einen Exponenten (e bzw. d) und den Modulus (N). Verwenden Sie für Ihre Implementierung die Funktion `mpz_powm`.
Hinweis: Verwenden Sie die Methode `mpz_class::get_mpz_t`, um die Instanzen der `mpz_class`-Klasse an die Funktion `mpz_powm` zu übergeben. Details zu den GMP*-Funktionen entnehmen Sie der Dokumentation unter `https://gmplib.org/manual/Function-Index.html#Function-Index`.*

LB-KÖS 02.

Ein RSA-Schlüsselpaar, bestehend aus dem öffentlichen Schlüssel (e, N) und dem privaten Schlüssel (d, N) kann wie folgt erzeugt werden:

1. Wählen Sie zwei voneinander verschiedene Primzahlen p und q, d.h. $p, q \in \mathbb{P}$, wobei $p \neq q$.
2. Berechnen Sie $N = pq$.
3. Berechnen Sie $\varphi(N) = (p-1)(q-1)$.
4. Wählen Sie für den öffentlichen Schlüssel eine ganze Zahl e zwischen 1 und $\varphi(N)$ (jeweils ohne die beiden Grenzen), die zu $\varphi(N)$ teilerfremd ist, d.h. $\mathrm{ggT}(e, \varphi(N)) = 1$.
5. Berechnen Sie für den privaten Schlüssel d als die Inverse von e modulo $\varphi(N)$, d.h. $d \equiv e^{-1} \mod \varphi(N)$.

Schreiben Sie ein Programm, das ein RSA-Schlüsselpaar wie oben angegeben erzeugt und den öffentlichen sowie den privaten Schlüssel auf der Konsole ausgibt. Verifizieren Sie die Schlüssel, indem Sie diese verwenden, um eine Nachricht mit Ihrem Programm aus LB-KÖS 01. zu ver- und wieder zu entschlüsseln. Die Schlüsselgröße (Bitlänge von N, die durch die Längen von p und q beeinflusst wird) soll mindestens 2.048 Bit betragen.
Hinweis: Verwenden Sie die Funktion `gmp_randinit_default`*, um bei Programmstart einen Zufallszahlengenerator zu initialisieren. Erzeugen Sie mit diesem und der Funktion* `mpz_urandomb` *eine Zufallszahl und wenden Sie* `mpz_nextprime` *an, um eine Primzahl zu erhalten.*
Für die Wahl von e iterieren Sie durch den angegebenen Wertebereich, bis Sie einen Wert gefunden haben, der das spezifizierte Kriterium erfüllt. Für die Berechnung des größten gemeinsamen Teilers verwenden Sie die Funktion `mpz_gcd`*. Nutzen Sie für arithmetische und Vergleichsoperationen wo immer möglich die entsprechenden überladenen Operatoren, z.B.* `+`*,* `<`*,* `==` *etc.*

LB-KÖS 03.

Modifizieren Sie Ihr Programm aus LB-KÖS 02. derart, dass die Berechnung der Inversen (für d) **nicht** über die Funktionen der *GMP*, sondern selbst implementiert wird. Nutzen Sie dabei die Tatsache aus, dass die Inverse im Fall von RSA aus dem Ergebnis des erweiterten Euklidischen Algorithmus abgelesen werden kann. Implementieren Sie dazu den erweiterten Euklidischen Algorithmus.
Hinweis: Eine iterative Variante zur Berechnung des erweiterten Euklidischen Algorithmus finden Sie unter anderem unter `http://www.oxfordmathcenter.com/drupal7/node/58`*. Alternativ kann auch eine rekursive Implementierung verwendet werden. Falls Sie den Algorithmus aus einer Quelle übernehmen, geben Sie die entsprechende Referenz im Quellcode an.*