RUHR
UNIVERSITÄT
BOCHUM

RUB

Lehrstuhl für Kryptologie und IT-Sicherheit
Prof. Dr. Alexander May
Alexander Meurer, Ilya Ozerov

Hausübungen zur Vorlesung

Kryptanalyse

WS 2011/2012

Blatt 1 / 19. Oktober 2010 / Abgabe bis spätestens 26. Oktober 2011, 10 Uhr in dem Kasten auf NA 02

AUFGABE 1 (5 Punkte)**:**
Alice feiert eine weitere Party und schickt eine Einladung m an Bob und Berta (Birte ist im Urlaub). Dabei verschlüsselt Alice m mit den öffentlichen RSA-Schlüsseln (N, e_1) und (N, e_2) von Bob und Berta, wobei e_1 und e_2 teilerfremd sind.
Wieder ist Eve nicht zur Party eingeladen. Helfen Sie Eve auch diesmal und zeigen Sie, dass man aus den Chiffretexten die Einladung m effizient berechnen kann.

Folgende Aufgabe liefert den Beweis zu Satz 25 zur *simultanen Polynomauswertung*.

AUFGABE 2 (10 Punkte)**:**
Sei $f(x)$ ein Polynom vom Grad. $n-1$ wobei n eine Zweierpotenz ist. Im Folgenden bezeichne mod das modulo auf dem Ring der Polynome in x. Wir setzen voraus, dass man $g(x)$ mod $h(x)$ für beliebige Polynome des Grades höchstens $n-1$ in Zeit $O(n \log n)$ berechnen kann. Gegeben seien $f(x)$ und n Stellen $x_0, \ldots, x_{n-1}$. Zu berechnen ist $f(x_i)$ für $i = 0, \ldots, n-1$. Für $0 \leq i \leq j \leq n-1$ definieren wir $p_{ij}(x) = \prod_{m=i}^{j}(x - x_m)$ und $q_{ij}(x) = f(x) \bmod p_{ij}(x)$.

(a) Zeigen Sie, dass $f(x) \bmod (x - z) = f(z)$ für alle z.

(b) Zeigen Sie, dass $q_{kk}(x) = f(x_k)$ und $q_{0,n-1}(x) = f(x)$.

(c) Zeigen Sie, dass für $i \leq k \leq j$ gilt $q_{ik}(x) = q_{ij}(x) \bmod p_{ik}(x)$ und $q_{kj}(x) = q_{ij}(x) \bmod p_{kj}(x)$.

(d) Konstruieren Sie einen Algorithmus, der in Zeit $O(n \log^2 n)$ die Werte $f(x_0), \ldots, f(x_{n-1})$ berechnet. Beweisen Sie *Korrektheit* und Laufzeit.

Hinweis: Realisieren Sie eine Divide-and-Conquer Strategie. Benutzen Sie (a) - (c).

Im Laufe des Übungsbetriebs werden regelmäßig praktische Aufgaben zur Implementierung von diversen Algorithmen und Angriffen gestellt. Dazu wird die kostenlos zur Verfügung stehende Mathematiksoftware *sage* verwendet (www.sagemath.org). Die Software läuft unter Unix-basierten Betriebssystemen oder unter Windows mit Hilfe von cygwin oder VirtualBox bzw. VMWare Player. Alternativ steht unter www.sagenb.org eine online-Version bereit.

Sage an sich ist in python implementiert und verwendet die python Syntax. D.h. beispielsweise sind Code-Blöcke nicht wie in C durch geschweifte Klammern gekapselt, sondern durch ein gewisses Einrückungsniveau. Eine Funktionsdefinition sieht dann folgendermaßen aus:

```
def foo(bar):
   print bar;
   return;
```

Sie finden die zu den Implementierungsaufgaben gehörigen Daten (z.B. Public Keys oder abgefangene Chiffretexte) auf der Webseite zur Vorlesung[1].

AUFGABE 3 (5 Punkte)**:**
Alice hat eine Einladung zu ihrer Geburtstagsparty an 17 Freunde verschickt. Diese besitzen die paarweise teilerfremden RSA-Moduln $N_1, N_2, \ldots, N_{17}$ und verwenden alle den öffentlichen Exponenten $e = 17$. Die Einladung wurde symmetrisch mit einem 80 Bit Schlüssel k verschlüsselt, welcher in einer (für alle Gäste identischen) Nachricht m asymmetrisch mit RSA verschlüsselt wurde. Die Nachricht m (Padding + Schlüssel) ist ein gültiger Klartext für alle Moduln, d.h. es gilt $m < \min\{N_1, \ldots, N_{17}\}$. Eve ist nicht zur Party eingeladen, konnte aber die Chiffretexte $c_1, \ldots, c_{17}$ mitschneiden.

Implementieren Sie einen Broadcast-Angriff (siehe Präsenzübung Blatt 1, Aufgabe 2) in *sage*. Benutzen Sie hierzu die Dateien `N.sobj` und `c.sobj`. Sie können die Liste der Moduln beispielsweise durch den Befehl `N=load('N')` einlesen, mit `N[i]` können Sie sich dann den $(i-1)$-ten Modul ausgeben lassen. Alternativ können Sie die Daten auch per Copy & Paste aus der Datei `crt.txt` einlesen. Wie lautet der Schlüssel k (die letzten 80 Bit von m)? Geben Sie den Quelltext Ihres Programms mit ab.

Hinweis: Sie können in *sage* mit der Funktion `crt` den verallgemeinerten Chinesischen Restsatz berechnen[2]. Zur Berechnung von $\sqrt[n]{a}$ über den ganzen Zahlen können Sie z.B. mit `(x^n-a).roots()` alle Nullstellen der Funktion $f(x) = x^n - a$ bestimmen und die ganzzahlige Lösung herausfiltern. Zur Bestimmung der letzten 80 Bit von m können Sie mit `hex` die Nachricht hexadezimal darstellen.

[1] `http://cits.rub.de/lehre/ws11/kryptanal1ws11.html`
[2] `http://www.sagemath.org/doc/reference/sage/rings/arith.html`

AUFGABE 4 (5 Punkte)**:**
Seien p, q, r paarweise verschiedene Primzahlen gleicher Bitlänge. Sei $N = p \cdot q$ ein RSA-Modul und $a = p \cdot r + x_0$ ein RSA-Modul mit kleinem additiven Fehler $x_0 \in \mathbb{Z}$ wobei $|x_0| < X_0$ für eine obere Schranke X_0.

(a) Beschreiben Sie einen Brute-Force Angriff, welcher zur Eingabe (N, a) in Zeit $\mathcal{O}(X_0 \cdot \log^2 N)$ die Faktorisierung von N berechnet. Zeigen Sie Korrektheit und beweisen Sie die Laufzeit.

(b) Implementieren Sie den Angriff in *sage*. Benutzen Sie hierfür die Daten aus der Datei `ggt.txt` und gehen Sie davon aus, dass $X_0 = 2048$. Wie groß ist x_0? Wie lautet die Faktorisierung von N? Geben Sie den Quelltext Ihres Programms mit ab.

Hinweis: In *sage* können Sie den ggT von x, y mittels `gcd(x,y)` berechnen. Die Laufzeit von `gcd` kennen Sie aus der Vorlesung.