RUHR UNIVERSITÄT BOCHUM

RUB

Lehrstuhl für Kryptologie und IT-Sicherheit
Prof. Dr. Alexander May
Mathias Herrmann, Enrico Thomae

Hausübungen zur Vorlesung

Kryptanalyse

WS 2010/2011

Blatt 1 / 13. Oktober 2010 / Abgabe bis spätestens 20. Oktober 2010, 10 Uhr in dem Kasten auf NA 02

AUFGABE 1 (5 Punkte)**:**
Seien $a, b, k, n, p \in \mathbb{N}$, p prim.
Zeigen Sie die folgenden Eigenschaften der Eulerschen φ-Funktion:

(a) $\varphi(p^k) = p^k(1 - \frac{1}{p})$

(b) $\varphi(ab) = \varphi(a)\varphi(b)$, falls $\gcd(a, b) = 1$.

(c) $\varphi(n) = n \prod_{p|n}(1 - \frac{1}{p})$, falls $n = \prod_{p|n} p^{k_p}$ die Primfaktorzerlegung von n ist.

AUFGABE 2 (5 Punkte)**:**
Sei G eine zyklische Gruppe. Zeigen Sie, dass es $\varphi(\mathrm{ord}(G))$ viele Generatoren in G gibt.

AUFGABE 3 (5 Punkte)**:**
Zeigen Sie den verallgemeinerten Chinesischen Restsatz:
Seien $m_1, m_2, \ldots, m_n$ teilerfremde natürliche Zahlen. Es existiert genau eine Lösung $x \bmod m_1 m_2 \ldots m_n$ des Gleichungssystems

$$\left| \begin{array}{rcl} x & = & a_1 \bmod m_1 \\ x & = & a_2 \bmod m_2 \\ & \vdots & \\ x & = & a_n \bmod m_n \end{array} \right| .$$

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;
```

AUFGABE 4 (5 Punkte)**:**

1. Implementieren Sie in *sage* eine Methode `mygcd` die den größten gemeinsamen Teiler von zwei Parametern a und b mit Hilfe des euklidischen Algorithmus berechnet.

2. Schreiben Sie eine Methode `mytest`. Diese Methode erhält zwei Parameter `bitsize` und `iter`. Zunächst sollen innerhalb der Methode zwei zufällige `bitsize`-bit große Zahlen generiert werden (z.B. mit der Funktion `randint()`). Dann wird `iter`-mal der gcd dieser beiden Zahlen mit der Funktion `mygcd` berechnet.

3. Verändern Sie die Methode `mytest` so dass sie als Rückgabewert die Zeit für die `iter`-malige Wiederholung der gcd-Berechnung liefert[1]. Dazu benötigen Sie das zusätzliche Modul *time*. Dieses kann hinzugefügt werden durch den Befehl `import time`. Danach liefert die Funktion `time.time()` die (abhängig vom Betriebssystem) die vergangenen Sekunden seit einem fixen Datum.

4. Erzeugen Sie eine Liste von Datenpunkten der Form [Bitgröße,Zeit], indem Sie die Methode `mytest` wiederholt mit steigenden Bitgrößen von 10 bis 1500 und einer Wiederholungszahl von 500 aufrufen(Eine Schrittweite von 10 ist ausreichend. Wenn die Berechnung zu lange dauern sollte, können Sie die Wiederholungszahl oder die maximale Bitgröße etwas verringern.). Betrachten Sie mit der Funktion `list_plot` einen Plot der Datenpunkte. Entspricht dieses Bild dem was Sie von der Laufzeit des euklidischen Algorithmus erwarten?

[1] Dies soll einfach nur die gemessene Zeit in einen vernünftigen Bereich bringen