

Aufgabe 33

Diese Frage lässt sich als Variante vom Geburtstags-’Paradox’ auffassen, und ihre Lösung wird hier wie im Skript hergeleitet : Sei n hier die Anzahl der möglichen Passwortkombinationen (diese Menge ist endlich und aufzählbar), k die Anzahl der User ($\{1, \dots, k\}$), und die Passwörter zufällig generiert (insbesondere also auch gleichmässig verteilt, sodass gilt $\mathcal{P}(b_i = r) = \frac{1}{n}$ für $i = 1, \dots, k$ und $r = 1, \dots, n$, wobei b_i die abgezählte Nummer des Passworts von User i und r die i te Passwortmöglichkeit bezeichnen). Die Wahrscheinlichkeit, dass zwei User i und j mit $i \neq j$ das selbe abgezählte Passwort r haben ist $\mathcal{P}(b_i = r, b_j = r) = \mathcal{P}(b_i = r) \cdot \mathcal{P}(b_j = r) = \frac{1}{n^2}$ und damit die Wahrscheinlichkeit dass beide ein beliebiges gleiches Passwort r haben $\mathcal{P}(b_i = b_j) = \sum_{r=1}^n \mathcal{P}(b_i = r, b_j = r) = \sum_{r=1}^n \frac{1}{n^2} = \frac{1}{n}$

Die Wahrscheinlichkeit, dass k Leute verschiedene Passwörter haben ist

$$B_k = \bigcap_{i=1}^{k-1} A_i$$

wobei A_i das Ereignis ist, dass das Passwort b_{i+1} von User $i+1$ verschieden von allen ’vorherigen’ Passwörtern b_1, \dots, b_i ist, also $A_i = (b_{i+1} \neq b_1) \wedge (b_{i+1} \neq b_2) \wedge \dots \wedge (b_{i+1} \neq b_i)$. Da $B_k = A_{k-1} \cap B_{k-1}$ gilt $\mathcal{P}(B_k) = \mathcal{P}(B_{k-1}) \cdot \mathcal{P}(A_{k-1} | B_{k-1})$ wenn anfänglich $\mathcal{P}(B_1) = 1$ gesetzt wird. Wenn die b_1, \dots, b_{k-1} alle verschieden sind, ist die bedingte Wahrscheinlichkeit, dass $b_k \neq b_i$ für $1 \leq i \leq k-1$ gleich $\frac{(n-k+1)}{n}$, denn von den n möglichen Passwörtern sind $n-k+1$ noch nicht ’belegt’. Aus den Vorüberlegungen oben folgt durch Iterieren $\mathcal{P}(B_k) = 1 \cdot (1 - \frac{1}{n}) \cdot (1 - \frac{2}{n}) \cdot \dots \cdot (1 - \frac{k-1}{n})$. Da $1+x \leq e^x$ gilt, gilt auch $\mathcal{P}(B_k) \leq (e^{-\frac{1}{n}}) \cdot (e^{-\frac{2}{n}}) \cdot \dots \cdot (e^{-\frac{k-1}{n}})$ und damit $\mathcal{P}(B_k) = e^{\sum_{i=1}^{k-1} -\frac{i}{n}} = e^{-\frac{k \cdot (k-1)}{2n}}$. Da B_k die Wahrscheinlichkeit, dass alle Passwörter paarweise verschieden sind angibt, ist $1 - B_k$ die Wahrscheinlichkeit angegeben, dass das Gegenteil der Fall ist – also mindestens 2 Passwörter gleich sind; diese wird letztenendes gesucht.

Die Wahrscheinlichkeit, dass zwei aus k generierte Passwörtern gleich sind ist grösser $\frac{1}{100}$ gdw. $B_k \leq \frac{99}{100}$, also $-\frac{k \cdot (k-1)}{2n} \leq \ln \frac{99}{100}$, nach k aufgelöst also $k \geq \frac{1}{2} \cdot \left(1 + \sqrt{1 + 8n \cdot \ln \left(\frac{100}{99} \right)} \right)$ (wegen

$$\begin{aligned}
-\frac{k \cdot (k-1)}{2n} &\leq \ln\left(\frac{99}{100}\right) \\
k \cdot (k-1) &\geq 2n \cdot -1 \cdot \ln\left(\frac{99}{100}\right) \\
k^2 - k + \frac{1}{2} &\geq \frac{1}{4} + 2n \cdot \ln\left(\left(\frac{99}{100}\right)^{-1}\right) \\
\left(k - \frac{1}{2}\right)^2 &\geq \frac{1}{4} + 2n \cdot \ln\left(\frac{100}{99}\right) \\
k - \frac{1}{2} &\geq \sqrt{\frac{1}{4} + 2n \cdot \ln\left(\frac{100}{99}\right)} \\
k &\geq \frac{1}{2} + \sqrt{\frac{1}{4} \cdot \left(1 + 8n \cdot \ln\left(\frac{100}{99}\right)\right)} \\
k &\geq \frac{1}{2} \cdot \left(1 + \sqrt{1 + 8n \cdot \ln\left(\frac{100}{99}\right)}\right)
\end{aligned}$$

(a) $|\text{GROSSBUCHSTABEN}| = 26$

Insgesamt sind also $n = 26^5 = 11881376$ verschiedene Passwörter möglich.
Eingesetzt in obige Ungleichung ergibt sich :

$$\begin{aligned}
k &\geq \frac{1}{2} \cdot \left(1 + \sqrt{1 + 8 \cdot 26^5 \cdot \ln\left(\frac{100}{99}\right)}\right) \\
k &\geq 489,19611 \dots
\end{aligned}$$

Somit ist die Wahrscheinlichkeit, dass zwei Nutzer das selbe Passwort erhalten grösser $1/100$, wenn mehr als 490 User Passwörter erhalten.

(b) Insgesamt sind $n = 94^5 = 7339040224$ verschiedene Passwörter möglich.
Eingesetzt in obige Ungleichung ergibt sich :

$$\begin{aligned}
k &\geq \frac{1}{2} \cdot \left(1 + \sqrt{1 + 8 \cdot 94^5 \cdot \ln\left(\frac{100}{99}\right)}\right) \\
k &\geq 12146,2663 \dots
\end{aligned}$$

Somit ist die Wahrscheinlichkeit, dass zwei Nutzer das selbe Passwort erhalten grösser $1/100$, wenn mehr als 12147 User Passwörter erhalten.

Die Berechnungen wurden per Google durchgeführt.

Aufgabe 34

(a)

$$\begin{aligned}
 x_p = y^d \pmod p &= 5817 \\
 x_q = y^d \pmod q &= 6998 \\
 (d, X_p, X_q) &= \text{extended_euclid}(p, q) = (1, 4500, -4499) \\
 x &= 35191907
 \end{aligned}$$

Somit entspricht das entschlüsselte x 35191907.

Dieses Ergebnis wurde mittels Perl-Skript ermittelt :

```
./uebung9.pl 34
```

(b)

$$\begin{aligned}
 x_{pt} = y^d \pmod n &\Rightarrow 0 \leq x_{pt} < n \\
 x_p = y^d \pmod p &\stackrel{(1)}{\Rightarrow} x_p = x_{pt} \pmod p \\
 x_q = y^d \pmod q &\stackrel{(1)}{\Rightarrow} x_q = x_{pt} \pmod q \\
 &x \equiv x_{pt} \pmod p \wedge x \equiv x_{pt} \pmod q \\
 &\Rightarrow x = x_{pt} - j \cdot p \wedge x = x_{pt} - k \cdot q \\
 &\Rightarrow x + j \cdot p = x_{pt} \wedge x + k \cdot q = x_{pt} \\
 &\stackrel{(2)}{\Rightarrow} k = j = 0 \wedge x = x_{pt}
 \end{aligned}$$

$$(1) y^d = x_{pt} + n \cdot i = x_{pt} + (pq) \cdot i$$

$$(2) x + j \cdot p = x_{pt} \wedge x + k \cdot q = x_{pt} \Rightarrow x + j \cdot p = x + k \cdot q \rightarrow j \cdot p = k \cdot q$$

$$\Rightarrow j = k = 0 \vee j = r \cdot q \wedge k = r \cdot p (r \in \mathbb{N})$$

Nimmt man an $j = r \cdot q \wedge k = r \cdot p \Rightarrow x_{pt} = x + r \cdot q \cdot p = x + r \cdot n \geq n$ was im Widerspruch zu $0 \leq x_{pt} < n$ steht (analog für k und p). Also $j=k=0$.

(c) Wieviel Rechenzeit hierbei gewonnen wird, hängt stark von den Implementations-eigenschaften von `modexp` (und in gewissen Grenzen `extended_euclid`) ab; ist ein zweifaches `mod_exp` mit kleineren p, q schneller als ein einfaches mit n , ist ein Rechenzeitvorteil zu erzielen, zumindest wenn man den notwendigen `extended_euclid` Berechnungsschritt aussen vor lässt (was man im Allgemeinen tun kann, da dieser nur von p und q abhängt und somit je Schlüsselpaar nur ein mal ausgeführt werden muss).

Allerdings muss hierfür tatsächlich zur Laufzeit der Unterschied der Ausführungszeiten bei `mod_exp` relevant sein (dies ist in unserer Perl-Implementation z.B. für solch kleine Zahlen wie hier im Beispiel nicht der Fall; Selbst unter Zuhilfenahme der externen GMP-BigInt-Bibliothek ist (auch mit grossen p, q, d) kein Gewinn zu verzeichnen – im Gegenteil.)

Dies kann natürlich auch am recht hohen Sprachniveau von Perl oder ungeeignet gewählten Zahlen liegen; es folgt also eine theoretische Betrachtung dieser Frage :

mod_exp benötigt $O(l^3)$ Bitoperationen, wenn die Eingabewerte a, b , und n jeweils Zahlen mit l Bits sind. Da in unserem Fall n die (Bitweise betrachtet) längste Zahl ist, nehmen wir o.B.d.A l als die Bitlänge von n an. In der alten RSA-Berechnungsmethode ist wird mod_exp einmal mit n aufgerufen. In der neuen RSA-Berechnungsmethode wird mod_exp 2 mal aufgerufen, jeweils mit einem p und q , welche (etwa) eine Bitlänge von $\frac{1}{2} \cdot l$ haben.

Vergleicht man nun die Laufzeit der Algorithmen in O-Notation, so ergeben sich $O_{old}(l^3)$ und $O_{new}\left(2 \cdot \left(\frac{1}{2} \cdot l\right)^3\right) = O_{new}\left(2 \cdot \frac{1}{8} \cdot l^3\right) = O_{new}\left(\frac{l^3}{4}\right)$. Hiernach ergibt sich eine theoretisch optimale Rechenzeiterparnis von 75% bzgl. Bitoperationen.

Aufgabe 35

Die beiden Faktoren von $n = 49601$ sind $p = 193$ und $q = 257$.

Dieses Ergebnis wurde mittels Perl-Skripts ermittelt :

```
./uebung9.pl 35 49601 515 14507
```

(welches das im Skript auf Seite 84f beschriebene Verfahren implementiert)

Anhang 33-35

```
#!/usr/local/bin/perl
use strict; $|=1; no strict 'refs'; use POSIX; use Math::BigInt;
use Benchmark;

sub extended_euclid {
    $_[1] or return $_[0], 1, 0;
    my ($d, $x, $y) = extended_euclid ($_[1], $_[0] % $_[1]);
    $d, $y, $x - $y * ($_[0] / $_[1]);
}

sub modexp {
    my ($a, $b, $n) = @_;
    my $d = new Math::BigInt (1);
    my @b = split //, sprintf "%0b", $b;

    for my $i (0..@b-1) {
        $d = $d*$d % $n;
        $d = $d*$a % $n if $b[$i]
    }
    $d
}

sub prob34 {
    my ($p, $q, $y, $d) = map { new Math::BigInt ($_) } 8999, 9001, 2**24, 80964007;
```

```

my (undef, $Xp, $Xq) = extended_euclid ($p, $q);

sub oldRSA {
  my $x = modexp ($y, $d, $p*$q), "\n";
}

sub newRSA {
  my ($xp, $xq) = map { modexp $y, $d, $_ } $p, $q;
  my $x = ($xp*$Xq*$q + $xq*$Xp*$p) % ($p*$q);
}

timethis (100, 'oldRSA');
timethis (100, 'newRSA');
}

sub prob35 {
  sub findp {
    sub euclid {
      $_[1] or return $_[0];
      euclid ($_[1], $_[0] % $_[1])
    }

    my ($n, $e, $d) = @_; my $a;
    my $s = scalar map { $a |= $_; $a ? () : $_ }
      reverse split //, sprintf '%b', $d*$e-1;
    my $r = ($e*$d-1)/(2**$s);
    while (1) {
      $a = int rand $n;
      my $q = euclid $a, $n;
      for (1..$s) {
        $q > 1 && $q < $n and return $q;
        $q = euclid modexp ($a, 2**($s-$_)*$r, $n)-1, $n
      }
    }
  }

  print 'p = ', $_[1] = findp (@_), ', q = ', $_[0]/$_[1], "\n";
}

&{'prob'.$ARGV[0]} (splice @ARGV,1)

```