<u>Frage</u>: Die multiplikative Gruppe $\mathbb{Z}_n^*$, $n$ Primzahl, ist bekanntermaßen zyklisch, also

$$\mathbb{Z}_n^* = \{a^1, a^2, \dots, a^{n-1}\} \quad \text{für ein } a \in \underset{\{1,2,\dots,n-1\}}{\overset{}{\mathbb{Z}_n^*}}.$$

Wieviele Elemente von $\mathbb{Z}_n^*$ sind Quadratzahlen (sog. quadratische Reste), wie viele sind keine Quadratzahlen (sog. quadratische Nichtreste)?

<u>Antwort</u>: Diejenigen $a^i$ mit $i$ gerade sind die Quadratzahlen. Also gibt es genau $\frac{n-1}{2}$ quadr. Reste und $\frac{n-1}{2}$ quadr. Nichtreste.

<u>Beispiel</u>: $\mathbb{Z}_7^* = \{3^1, 3^2, 3^3, 3^4, 3^5, 3^6\}$

$$\begin{array}{cccccc} 3^1 & 3^2 & 3^3 & 3^4 & 3^5 & 3^6 \\ \parallel & \parallel & \parallel & \parallel & \parallel & \parallel \\ 3 & 2 & 6 & 4 & 5 & 1 \end{array}$$

Quadrate sind: $\begin{cases} 2 = 3^2 = 4^2 \\ 4 = 2^2 = 5^2 \\ 1 = 6^2 = 1^2 \end{cases}$ $\quad QR_7 = \{2, 4, 1\}$

Nicht-Quadrate sind: $3, 6, 5$ $\quad QNR_7 = \{3, 6, 5\}$

<u>Aufgabe</u>: Bei allen Krypto-Protokollen, die auf dem Diskreten Logarithmus beruhen, benötigt man eine Primzahl $n$ und eine Primitivwurzel $a$ mod $n$. Bekannt ist das <u>Primitivwurzelkriterium</u>: es muss gelten $a^{(n-1)/q} \not\equiv 1 \pmod{n}$ für alle Primteiler $q$ von $n-1$.

Wir nehmen an, die Primzahleigenschaft ist effizient überprüfbar (→ später). Aber wie kann man $n$ so bestimmen, dass man das Kriterium anwenden kann (Problem: Faktorisierung von $n-1$ sollte bekannt sein).

<u>Antwort</u>: Man wählt zunächst eine große Primzahl $q$ und analysiert dann $2 \cdot q + 1 =: n$, ob dies auch eine Primzahl ist. Wenn dies gelingt, dann kennt man die Primfaktorisierung von $n-1$, nämlich: $2 \cdot q$. Man muss also für das Kriterium überprüfen: $a^{\frac{n-1}{2}} = a^q \neq 1$ und $a^{\frac{n-1}{q}} = a^2 \neq 1$.

**Anmerkungen:** In diesem Fall, also

$n$ Primzahl, $n = 2q+1$, $q$ Primzahl

nennt man $n$ eine <u>sichere Primzahl</u> und

$q$ eine <u>Sophie-Germain-Primzahl</u>.

Da solche $n, q$ - Kombinationen vielleicht nicht so häufig vorkommen, könnte man auch stattdessen versuchen $n$ Primzahl, $n = 2^t \cdot q + 1$ (für eine kleine Zahl $t$), $q$ Primzahl, zu finden.

Die Häufigkeit der Primzahlen innerhalb der natürlichen Zahlen wird durch den <u>Primzahlsatz</u> festgestellt:

$$\boxed{\text{Anzahl Primzahlen} \in \{1, \dots, n\} \approx \frac{n}{\ln(n)}}$$

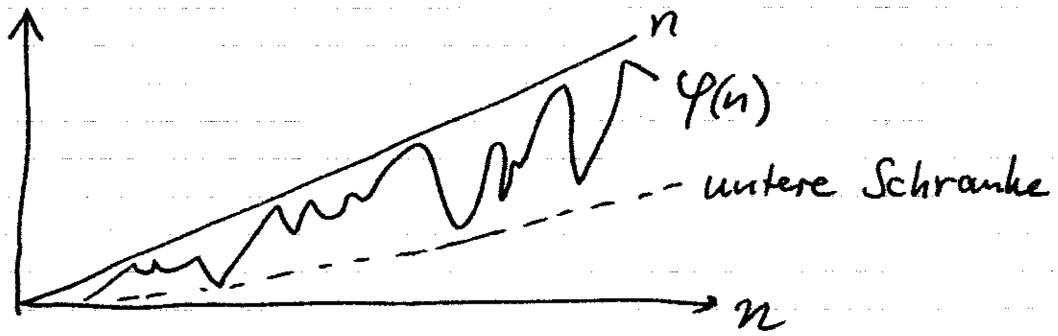
Wenn man also eine ca. 500 Bit-Primzahl benötigt, so rechnet man: $\dfrac{2^{500}}{\ln(2^{500})} \approx 10^{148}$

Somit: ca. jede $\dfrac{2^{500}}{10^{148}} \approx 346$. Zahl ist eine Primzahl, bzw. jede 173. <u>ungerade</u> Zahl. Methode:

$$\boxed{1}\underbrace{\boxed{\phantom{0}}\boxed{\phantom{0}}\boxed{\cdots}\boxed{\phantom{0}}\boxed{\phantom{0}}}_{498 \text{ Zufallsbits}}\boxed{1}$$

Teste, ob diese Zahl eine Primzahl ist.

Anmerkung: Die Gruppe $\mathbb{Z}_n^*$ hat $\varphi(n)$ viele Elemente. Aus kryptographischen Gründen sollten $n$ und $\varphi(n)$ nicht zu weit auseinanderklaffen.




Die größte Kluft zwischen $n$ und $\varphi(n)$ entsteht, wenn $n$ folgende Form hat:

$$n = \underbrace{2\cdot 3}_{6}, \quad \underbrace{2\cdot 3\cdot 5}_{30}, \quad \underbrace{2\cdot 3\cdot 5\cdot 7}_{210}, \quad \underbrace{2\cdot 3\cdot 5\cdot 7\cdot 11}_{2310}, \ldots$$

$$\varphi(n) = \underbrace{1\cdot 2}_{2}, \quad \underbrace{1\cdot 2\cdot 4}_{8}, \quad \underbrace{1\cdot 2\cdot 4\cdot 6}_{48}, \quad \underbrace{1\cdot 2\cdot 4\cdot 6\cdot 10}_{480}, \ldots$$

Es gilt der folgende Satz:

$$\boxed{\text{Für große } n \text{ gilt } \varphi(n) \geq \frac{n}{6\cdot \ln(\ln(n))}}$$

Beispiel: Für $n$ Primzahl ist die Anzahl der Primitivwurzeln von $\mathbb{Z}_n^*$ genau $\varphi(n-1) \geq \dfrac{n-1}{6\cdot \ln(\ln(n-1))}$

Wenn $n$ eine 500-Bitzahl ist, so ist dies eine $\approx 495$ Bitzahl.

## Rechenzeit der verschiedenen mathematischen Operationen, die zum Ver-/Ent-Schlüsseln nötig sind.

**Addition / Subtraktion** (ggf. modulo n)

nach „Schulmethode“: $O(m)$ $m$ ... Anzahl Bits

**Multiplikation / Division** (ganzzahlig mit Berechnung des Rests: $123 : 20 = 6$ Rest $3$)

$6$: 123 div 20

$3$: 123 mod 20

nach Schulmethode: $O(m^2)$

**Mult:**

$$\overset{\longleftarrow m \longrightarrow}{10110} * \overset{\longleftarrow m \longrightarrow}{11001}$$

$$\left.\begin{array}{r}
10110\phantom{000} \\
+\ 10110\phantom{0000} \\
\hline
1000010\phantom{000} \\
+\ 10110\phantom{0} \\
\hline
1000100110
\end{array}\right\}\ \text{Aufwand } O(m^2)$$

## Division:

$$
\begin{array}{l}
\overleftrightarrow{\quad n \quad} \qquad \overleftrightarrow{m} \\
1111011 : 101 = 11000 \text{ Rest } 11 \\
-\,101 \\
\phantom{-\,}0101 \\
-\,\phantom{0}101 \\
\phantom{-\,0}0011
\end{array}
$$

Grob: $O(n^2)$ Aufwand

Genauer: $O(m \cdot (n-m))$ , $n \geq m$

## Exponentiation:

$$
a^b = \begin{cases} 1, & \text{falls } b = 0 \\ a, & \text{falls } b = 1 \\ (a^{b/2})^2, & \text{falls } b > 1, \text{ gerade} \\ a \cdot (a^{\frac{b-1}{2}})^2, & \text{falls } b > 1, \text{ ungerade} \end{cases}
$$

Zusätzlich:

Berechnung modulo $n$ durchführen.

(Jedes Zwischenergebnis kann mod $n$ reduziert werden.)

Rekursive Prozedur:

```
proc modexp (a, b, n)
  // berechnet a^b mod n
  if b = 0 then return 1
  if b = 1 then return a
  if even(b) then return sqr(modexp(a, b/2, n)) mod n
  else (a * sqr(modexp(a, (b-1)/2, n))) mod n
```

Analyse: $a, b, n$ seien $m$-Bitzahlen

Anzahl rekursive Aufrufe ist $\leq \log_2 b \approx m$

Pro Aufruf 1 oder 2 modulare Multiplikationen und mod-Operation

Insgesamt: Aufwand $O(m^3)$.

## Iterative Version:

Die Binärdarstellung von $b$ sei $(x_{m-1} \cdots x_1 x_0)_2$

d.h.: $$b = \sum_{i=0}^{m-1} x_i \cdot 2^i$$

Algorithmus:

$d := 1;$

**for** $i := m-1$ **downto** $0$ **do**

$\quad d := (d * d) \bmod n;$

$\quad$ **if** $x_i = 1$ **then** $d := (d * a) \bmod n$

**Beispiel:**

$$b = 22 = (1\,0\,1\,1\,0)_2$$

$$a^{22} = \underbrace{(\underbrace{\underbrace{(\underbrace{(\underbrace{(1^2\cdot a)^2}_{a^2})^2}_{a^4}\cdot a}_{a^5})^2}_{a^{10}}\cdot a}_{a^{11}})^2}_{a^{22}}$$

Berechnung des größten gemeinsamen Teilers mit dem Euklid-Algorithmus:

Rekursive Formulierung:

```
proc Euklid (a,b)
if b=0 then return a
  else return Euklid(b, a mod b)
```

Beispiel: $a = 72$, $b = 21$

$72 : 21 = 3$ Rest $9$

$21 : 9 = 2$ Rest $3$

$9 : 3 = 3$ Rest $0$

$\Uparrow$

$ggT(72,21)$

Analyse von Euklid:

Sei $a = n_1$, $b = n_2$. Nun berechnet Euklid:

$$n_1 = q_1 \cdot n_2 + n_3 \qquad (n_3 < n_2)$$
$$n_2 = q_2 \cdot n_3 + n_4 \qquad (n_4 < n_3)$$
$$\vdots$$
$$n_{t-2} = q_{t-2} \cdot n_{t-1} + n_t \qquad (n_t < n_{t-1})$$
$$n_{t-1} = q_{t-1} \cdot n_t + 0$$

Das Euklid-Ergebnis ist $n_t$.

Wir zeigen: Erstens $n_t$ ist ein Teiler von $n_1, n_2$

Von der letzten Zeile zur ~~letzten~~ ersten hocharbeiten:

$$n_t \mid n_{t-1}, \quad n_t \mid n_{t-2}, \ldots, n_t \mid n_2, \quad n_t \mid n_1$$

Zweitens zeigen: Wenn $d$ beliebiger gemeinsamer Teiler von $n_1, n_2$ ist, dann ist $d$ auch ein Teiler von $n_t$.

Von oben nach unten arbeiten:

erste Zeile: $n_3 = n_1 - q_1 \cdot n_2 \Rightarrow d \mid n_3$

dito: zweite Zeile, usw.

vorletzte Zeile $\Rightarrow d \mid n_t$.

Also ist $n_t$ größter gemeinsamer Teiler von $n_1, n_2$

Laufzeitbetrachtung:

Betrachte die Folge der Argumente von

Euklid: $(a_0, b_0), (a_1, b_1), (a_2, b_2), \ldots, (a_{k-1}, b_{k-1}), (\underset{\underset{ggT}{\shortparallel}}{a_k}, 0)$

Dann gilt: $b_i > 2 \cdot b_{i+2}$

Nachrechnen: $b_i = a_{i+1} = q \cdot b_{i+1} + b_{i+2}$, $q \in \mathbb{N}$

Es folgt: $b_i \geq b_{i+1} + b_{i+2} > b_{i+2} + b_{i+2} = 2 b_{i+2}$

Somit ist die Anzahl der rekursiven Euklid-Aufrufe höchstens $O(\log b_0) = O(m)$

Grobe Abschätzung der Laufzeit:

$$\underbrace{O(m)}_{\text{rekursive Aufrufe}} \cdot \underbrace{O(m^2)}_{\text{Aufwand für Division}} = O(m^3)$$

Genauere Abschätzung der Laufzeit: Die Eingabezahlen haben die Anzahl ~~[illegible]~~ von Bits $m_0 \geq m_1$.

Danach $\geq m_2 \geq m_3 \geq \ldots \geq m_k$

Die Divisionen bei Euklid haben die Laufzeiten: $(m_0 - m_1)m_1$, $(m_1 - m_2)m_2, \dots, (m_{k-1} - m_k)m_k$

Aufsummieren:

$$(m_0 - m_1)m_1 + (m_1 - m_2)m_2 + \dots + (m_{k-1} - m_k)m_k$$

$$= m_0 m_1 - m_1^2 + m_1 m_2 - m_2^2 + \dots + m_{k-1} m_k - m_k^2$$

$$\leq m_0^2 - m_1^2 + m_1^2 - m_2^2 + \dots + m_{k-1}^2 - m_k^2$$

$$= m_0^2 - m_k^2 \leq m_0^2$$

Somit ist die genauere Laufzeit-Abschätzung: $O(m^2)$