4.4 Generovanie prvočísiel

Tento oddiel pojednáva o algoritmoch na generáciu prvočísiel pre šifrovacie účely. Sú tu prezentované štyri algoritmy: Algoritmus 4.44 pre generáciu pravdepodobností prvočísiel (viď Definícia 4.5), algoritmus 4.53 pre generáciu silných prvočísiel(viď definícia 4.52), Algoritmus 4.56 pre generáciu pravdepodobných prvočísiel p a q vhodných pre používanie v algoritme DSA(Digital Signature Algorithm), a Algoritmus 4.62 pre generáciu dokázateľných prvočísiel (viď definícia 4.34).

4.43 Poznámka (generácia prvočísiel vs. test prvočíselnosti) Generácia prvočísiel sa líši od testu prvočíselnosti ako je popísané v §4.2 a §4.3, ale môže a typicky aj obsahuje nedostatok.
Prvý pripustí konštrukciu kandidátov pevného tvaru čo môže viesť k účinnejšiemu testovaniu než je možné pre náhodných kandidátov.

4.4.1 Náhodné hľadanie pravdepodobných prvočísiel

Podľa poučky o prvočíslach (fakt 2.95), pomer (kladných) celých čísiel $\leq x$, ktoré sú prvočísla je približne $1/\ln x$. Pretože polovica zo všetkých celých čísel $\leq x$ sú párne, pomer nepárnych celých čísel $\leq x$, ktoré sú prvočísla je približne $2/\ln x$. Napríklad, pomer zo všetkých nepárny celých čísel $\leq 2^{512}$, ktoré sú prvočísla je približne $2/(512 * \ln(2)) \approx 1/177$. Toto núka myšlienku, že rozumná stratégia pre vybratie náhodného k-bitového (pravdepodobného) prvočísla je opakovane zobrať náhodné k-bitové nepárne celé čísla n až kým sa nenájde jedno definované ako "prvočíslo" algoritmom MILLER-RABIN(n,t) (Algoritmus 4.24) pre príslušnú hodnotu bezpečnostného parametra t (diskutovaný nižšie).

Ak náhodné k-bitové nepárne celé číslo n je deliteľné malým prvočíslom, je menej výpočtovo náročné stanovenie kandidáta n pomocou skúšky delenia než pri použití Miller-Rabin testu. Pretože pravdepodobnosť, že náhodné celé číslo n má malého prvočíselného deliteľa je relatívne veľká, skôr než použijeme Miller-Rabin test, kandidát n by mal byť testovaný pre malé delitele pod akousi vopred danou medzou B. To sa môže urobiť delením čísla n všetkými prvočíslami menšími ako B, alebo výpočtom najväčšieho spoločného deliteľa čísla n a (prepočítaných) produktov niekoľkých prvočísiel $\leq$ B. Rozsah kandidáta nepárneho celého čísla n nepredpísaného pomocou tejto skúšky delenia je $\Pi_{3\leq p\leq B^{(1-1/p)}}$ ktorý, podľa Mertensovej poučky, je približne 1.12/lnB (tu p sa pohybuje cez prvočíselné hodnoty). Napríklad, ak B = 256, potom iba 20% kandidátov nepárneho celého čísla n prejde fázou skúšky delenia, t.j., 80% je vyradené skôr než je náročnejší Miller-Rabin test vykonaný.

4.44 Algoritmus *Náhodné hľadanie prvočísla použitím Miller-Rabin testu*

RANDOM-SEARCH (k, t)
VSTUP: celé číslo k, a bezpečnostný parameter t (cf. poznámka 4.49).
VÝSTUP: náhodné k-bitové pravdepodobné prvočíslo.

1. Generovať nepárne k-bitové celé číslo n náhodne.
2. Použiť skúšobné delenie na určenie či n je deliteľné nejakým nepárnym prvočíslom ≤ B (viď poznámka 4.45 pre návod na výber B). Ak to platí choď na krok 1.
3. Ak výstupom MILLER-RABIN(n,t) (Algoritmus 4.24) je "prvočíslo" skonči a vráť(n). Inak, choď na krok 1

4.45 Poznámka (*najvhodnejšia skúška delenia medze B*). Nech E označuje čas pre plné k-bitové modulové umocňovanie, a nech D označuje čas požadovaný pre výpis jedného malého prvočísla ako deliteľa k-bitového celého čísla. (Hodnoty E a D závisia na konkrétnej implementácii aritmetiky veľkých celých čísel). Potom skúška delenia medze B, ktorá minimalizuje očakávanú dobu behu Algoritmu 4.44 pre vytvorenie k-bitového prvočísla je zhruba B = E / D. Presnejší odhad optimálnej veľkosti výberu B môže byť získaný experimentálne. Nepárne prvočísla až do B môžu byť prepočítané a uložené v tabuľke. Ak pamäť je vzácna, hodnota B, ktorá je menšia než optimálna hodnota môže byť použitá. Pretože Miller-Rabin test neprevádza matematický dôkaz že číslo je skutočne prvočíslo, číslo n vrátené algoritmom 4.44 je pravdepodobne prvočíslo (definícia 4.5). Je preto dôležité, mať odhad pravdepodobnosti že n je naozaj zložené.

4.46 Definícia Pravdepodobnosť , že RANDOM-SEARCH (k, t) (algoritmus 4.44) vracia zložené číslo je označené $p_{k,t}$.

4.47 Poznámka (*poznámky na odhad* $p_{k,t}$). Je lákavé vyvodzovať priamo z faktu 4.25, že $p_{k,t} \le (1/4)^t$. Takéto myslenie je chybou (hoci typicky záver bude v praxi správny) pretože to neberie do úvahy rozdelenie prvočísiel. (Napríklad, ak všetci kandidáti n boli vybratí z množiny S zložených čísel, pravdepodobnosť omylu je 1.) Nasledujúca diskusia je vypracovaná na tomto bode. Nech X znázorňuje udalosť že n je zložené, a nech Y_t značí udalosť že MILLER-RABIN (n, t) vyhlási n za prvočíslo.

Potom fakt 4.25 konštatuje, že $P(Y_t|X) \le (1/4)^t$. Čo je však dôležité k odhadu $p_{k,t}$ je veľkosť $P(Y_t|X)$. Predpokladáme že kandidáti n sú vykreslení jednotne a náhodne z množiny S nepárnych čísel, tiež že p je pravdepodobnosť, že n je prvočíslo (toto závisí na kandidátovi množiny S). Predpokladajme tiež že 0 < p<1. Potom Bayesova teoréma (fakt 2.10):

$$P(X \mid Y_t) = \frac{P(X).P(Y_t \mid X)}{P(Y_t)} \le \frac{P(Y_t \mid X)}{P(Y_t)} \le \frac{1}{p}.\left(\frac{1}{4}\right)^t \quad ,$$

Pretože $P(Y_t) > p$. Teda pravdepodobnosť $P(X \mid Y_t)$ môže byť značne väčšia než $(1/4)^t$ ak p je malé. Avšak, pravdepodobnosť omylu Miller-Rabin testu je obyčajne menšia než $(1/4)^t$ (viď. poznámka 4.26). Použitím lepších odhadov pre $P(Y_t|X)$ a odhady na čísla k-bitových prvočísiel, to bolo ukázané že $p_{k,t}$ je, v skutočnosti, menší než $(1/4)^t$ pre všetky dostatočne veľké k. Konkrétnejší výsledok je nasledovný: ak kandidáti n sú vybratí náhodne z množiny nepárnych čísel v intervale [3, x], potom

$P(X \mid Y_t) \leq (1/4)^t$ pre všetky x ≥ 10^{60}. Ďalší upresnenie $P(X \mid Y_t)$ dovoľuje nasledujúce explicitná horná hranica na $p_{k,t}$ pre rôzne hodnoty k a t.[5]

4.48 Definícia (niektoré ohraničenia $p_{k,t}$ v algoritme s Miller - Rabinovým testom)

(i) $p_{k,1} < k^2 4^{2-\sqrt{k}}$ for $k > 2$.

(ii) $p_{k,t} < k^{3/2} 2^t t^{1/2} 4^{2-\sqrt{tk}}$ for $(t = 2, k > 88)$ or $(3 < t < k/9, k > 21)$.

(iii) $p_{k,t} < \frac{7}{20} k 2^{-5t} - \frac{1}{7} k^{15/4} 2^{-k/2-2t} + 12k 2^{-k/4-3t}$ for $k/9 \leq t \leq k/4$, $k \geq 21$.

(iiii) $p_{k,t} < \frac{1}{7} k^{15/4} 2^{k/2-2t}$ for $t > k/4$, $k > 21$.

Ak napríklad $k = 512$ a $t = 6$, potom podľa vety 8 (II) dostávame $p_{512,6} \leq (1/2)^{88}$. Teda pravdepodobnosť , že výsledkom Náhodného Hľadania (512,6) je 512 – bitové celé číslo je menej ako $(1/2)^{88}$. Použitím prepracovanejších postupov môžeme dosiahnuť zlepšenie (spresnenie) ohraničení $p_{k,t}$, určených použitím vety 8 . Tieto spresnené ohraničenia je možné určiť pomocou vzťahov , ktoré platia pre dané postupy . Tabuľka 3 udáva niektoré spresnené ohraničenia $p_{k,t}$ pre vybrané hodnoty parametrov k a t . Pre ilustráciu, pravdepodobnosť , že výsledkom Náhodného Hľadania (500,6) je zložené číslo, je < $(1/2)^{92}$. Zaujímavé je , že hodnoty $p_{k,t}$, ktoré je možné určiť z tabuľky 3 sú podstatne menšie než $(1/4)^t = (1/2)^{2t}$.

	t									
k	1	2	3	4	5	6	7	8	9	10
100	5	14	20	25	29	33	36	39	41	44
150	8	20	28	34	39	43	47	51	54	57
200	11	25	34	41	47	52	57	61	65	69
250	14	29	39	47	54	60	65	70	75	79
300	19	33	44	53	60	67	73	78	83	88
350	28	38	48	58	66	73	80	86	91	97
400	37	46	55	63	72	80	87	93	99	105
450	46	54	62	70	78	85	93	100	106	112
500	56	63	70	78	85	92	99	106	113	119
550	65	72	79	86	93	100	107	113	119	126
600	75	82	80	95	102	108	115	121	127	133

Tabuľka 3 : *Ohraničenia $p_{k,t}$ pre vybrané hodnoty* k *a* t . *Hodnota* j *odpovedajúca hodnotám* k *a* t *spĺňa podmienku* $p_{k,t} \leq \left(\frac{1}{2}\right)^j$.

[5] odhady $p_{k,t}$ prezentované v zostávajucej časti tejto podsekcie sú odvodené pre prípad algoritmu využívajúceho Miller- Rabinov test , kedy na skúšku nie je využívané delenie malým prvočíslom pre vylúčenie niektorých **n** . Pretože skúška delením nikdy nevylúči prvočíslo, môže tento postup poskytnúť lepšiu príležitosť vylúčiť kombinácie . Takto môže byť pravdepodobnosť chyby $p_{k,t}$ dokonca menšia , než poskytujú uvádzané odhady .

4.49 Poznámka (korekcia pravdepodobnosti chyby)

V praxi je pri generovaní prvočísel algoritmom využívajúcim Miller – Rabinov test postačujúca pravdepodobnosť chyby $(1/2)^{80}$. Tabuľka 4 ilustruje najnižšie hodnoty t , pre ktoré je pri použití vety 8 , a pre vybrané hodnoty k , možné dosiahnuť platnosť podmienky $p_{k,t} \leq (1/2)^{80}$. Napríklad pri generovaní 1000 – bitového prvočísla je postačujúce realizovať $t = 3$ opakovania Miller – Rabinovho testu . Algoritmus využívajúci Miller – Rabinov test vylučuje väčšinu kandidátov n buď skúškou delením (v 2. kroku) , alebo prevedením jedinej iterácie Miller – Rabinovho testu (v 3. kroku) . Práve preto , jediný účinok výberu väčšieho bezpečnostného parametra t v priebehu výkonu algoritmu , bude zvýšenie požadovaného v poslednej fáze , kedy je zvolené predpokladané prvočíslo .

k	t
100	27
150	18
200	15
250	12
300	9
350	8
400	7
450	6

k	t
500	6
550	5
600	5
650	4
700	4
750	4
800	4
850	3

k	t
900	3
950	3
1000	3
1050	3
1100	3
1150	3
1200	3
1250	3

k	t
1300	2
1350	2
1400	2
1450	2
1500	2
1550	2
1600	2
1650	2

k	t
1700	2
1750	2
1800	2
1850	2
1900	2
1950	2
2000	2
2050	2

Tabuľka 4 : *Najmenšie* t *určené podľa vety 8 , ktoré pri vzorke* k *spĺňa podmienku* $p_{k,t} \leq (1/2)^{80}$.

4.50 Pripomienka (Miller – Rabinov test so základom $a = 2$)

Miller – Rabinov test vyžaduje umocňovanie základu a . Táto operácia môže byť realizovaná opakovaným aplikovaním algoritmu Umocňovania a Násobenia . Ak a=2, násobenie základom a je jednoduchý postup .
Jedna optimalizácia algoritmu s Miller – Rabinovým testom potom spočíva v stanovení základu $a = 2$ v 3. kroku prvého priebehu Miller – Rabinovho testu . Keďže väčšina zložených čísel by pri Miller – Rabinovom teste so základom a = 2 zlyhala , skráti sa tým čas priebehu algoritmu , ktorý na generovanie prvočísel využíva Miller – Rabinov test .

4.51 Poznámka (Prírastkové vyhľadávanie)

(i) Alternatívnym postupom pre náhodné generovanie kandidátov n v 1. kroku algoritmu s Miller – Rabinovým testom , je zvoliť najprv náhodné k – bitové nepárne číslo n_0 a potom prednostne testovať s čísel $n\text{-}n_0$, n_0+2 , n_0+4 , . . . , $n_0+2(s-1)$. Ak sa zistí , že všetky tieto čísla sú číslami zloženými , potom bol daný algoritmus neúspešný . Ak $s = c \ln 2^k$, kde c je konštanta , potom pravdepodobnosť $q_{k,t,s}$ určenia zloženého čísla , pomocou vyššie uvedeného postupu , je nižšia než $\delta k^3 2^{-\sqrt{k}}$, pre nejaké konštantné δ . Tabuľka 5 poskytuje otvorené ohraničenia pre túto pravdepodobnosť chyby pre $k = 500$ a $t \leq 10$. Za teoretických predpokladov bolo dokázané , že pravdepodobnosť zlyhania tohto algoritmu je menšia než $2e^{-2c}$, pre dostatočne veľké k .

(ii) Výhodou prírastkového vyhľadávania prvočísel je menší potrebný počet náhodných bitov . Navyše skúška (resp. test) delením malými prvočíslami v 2. kroku algoritmu s Miller – Rabinovým testom môže byť účinne zdokonalená . Najprv sú vypočítavané hodnoty $R[p] = n_o \bmod p$ pre všetky nepárne čísla $p \le B$. V ďalšom sú hodnoty $R[p]$ zväčšované o 2 , pričom v tabuľke sú R aktualizované ako $R[p] \leftarrow (R[p] + 2) \bmod p$. Kandidáti n prejdú skúškou delením len vtedy , ak žiadne $R[p]$ nie je rovné 0 .

(iii) Ak B je veľké , alternatívnym postupom pre realizáciu skúšky delením je inicializovať tabuľku $S[i] \leftarrow 0$ *pre* $0 \le i \le (s-1)$, pričom vstup $S[i]$ prislúcha kandidátovi n_o+2i . Pre každé nepárne číslo $p \le B$ je vypočítavané $n_o \bmod p$. Ak j je najmenší index , pre ktorý platí $(n_o+2j) \equiv 0 \pmod p$, potom $S[j]$ a každé p^{th} sú nastavené na 1 . Kandidát n_o+2i potom prejde skúškou delením len vtedy , ak $S[i] = 0$.

	t									
c	1	2	3	4	5	6	7	8	9	10
1	17	37	51	63	72	81	89	96	103	110
5	13	32	46	58	68	77	85	92	99	105
10	11	30	44	56	66	75	83	90	97	103

Tabuľka 5 : *Ohraničenia pravdepodobnosti chyby prírastkového vyhľadávania pre k=500 a vybrané hodnoty* c *a* t . *Vstup* j *odpovedajúci* c *a* t *spĺňa podmienku* $q_{500,t,s} \le (1/2)^j$, *kde* $s = c \ln 2^{500}$.

4.2 Silné prvočísla

RSA kryptosystém (§8.2) používa modul vo forme $n=pq$, kde p a q sú odlišné prvočísla. Prvočísla p a q musia byť dostatočne veľké aby faktorizácia ich súčinu sa dala vypočítať. Navyše, by mali byť náhodné prvočísla, v zmysle že by mali byť vybraté ako funkcie náhodného vstupu cez proces definovania bloku kandidátov s dostatočnou veľkosťou, ktoré by vylučovali akekoľvek napadnutie. V praxi, vyplývajúce prvočísla musia tiež mať predbežne zistenú dĺžku kvôli špecifikácii systému. Objavenie RSA kryptosystému viedlo k úvahám o niekoľkých ďalších obmedzeniach na výber p a q, ktoré sú nutné na zabezpečenie RSA systémov pred kryptoanalytickým útokom, a pojem silných prvočísel (Definícia 4.52) bol zadefinovaný. Tieto útoky sú opísané v potrebnej dĺžke v kapitole 8.8(iii); ako najznámejšie, to je teraz myslené že silné prvočísla poskytujú malú ochranu akú ponúkali náhodné prvočísla, od náhodne vybratých prvočísel o veľkosti typicky používaných v RSA module dnes budú vyhovovať obmedzeniam s vyššou pravdepodobnosťou. Na druhej strane, oni nie sú menej bezpečné a vyžadujú iba minimum času navyše pre výpočet; teda je malé reálne navýšenie ceny pri ich používaní.

4.52 Definícia Prvočíslo p sa nazýva *silné prvočíslo,* ak celé čísla r, s a t existujú tak že sú splnené nasledujúce tri podmienky:

(i) p-1 mal veľký prvočíselný faktor, označovaný r;
(ii) p+1 mal veľký prvočíselný faktor, označovaný s; a
(iii) r-1 mal veľký prvočíselný faktor, označovaný t.

V definícií 4.52 presná kvalifikácia „veľký“ závisí na špecifikácií útoku proti akému by mal byť chránený; pre viac detailov viď. odsek 8.8(iii).

4.53 ALGORITMUS Gordonov Algoritmus pre generovanie silného prvočísla

---

Stručný obsah: silné prvočíslo p je generované.

1. Generuje dve dlhé prvočísla s a t zhruba odpovedajúci počtu bitov (pozri odsek 4.54).
2. Vyber cele čislo i_0. Nájdi prvé prvočíslo v sequencii $2it+1$,pre $i = i_0 , i_0 + 1, i_0+2$ ... (pozri odsek 4.54). Nahraď toto prvočíslo $r=2it+1$.
3. Vypočítaj $p_0=2(s^{r-2} \bmod r) s-1$.
4. Zvoľ si celé číslo j_0. Nájdi prvé prvočíslo v sekvencii p_0+2jrs, pre $j=j_0, j_0+1, j_0+2$ ... (pozri odsek 4.54). Nahraď toto prvočíslo $p=p_0+2jrs$.
5. Vráť(p).

Zdôvodnenie. Aby sme videli že prvočíslo p vypočítané Gordonovym algoritmom je naozaj silné prvočíslo, všimnime si najprv(predpoklad $r \neq s$) že $s^{r-1} \equiv 1 \pmod r$; to vyplýva z Fermatovho teoremu (fakt 2.127). Odtiaľ $p_0 \equiv 1 \pmod r$ a $p_0 \equiv -1 \pmod s$. Nakoniec (porovnaj Definíciu 4.52),

(i) $p-1=p_0+2jrs-1=0 \pmod r$,a teda p-1 má hlavný koeficient r;
(ii) $p+1=p_0+2jrs+1=0 \pmod s$, a teda $p+1$ má hlavný koeficient s; a
(iii) $r-1=2it=0 \pmod t$, a teda r-1 ma hlavný koeficient t.

4.54 Poznámka *(implementovanie Gordonovho algoritmu)*

(i) Prvočísla s a t požadované v 1 kroku môžu byť pravdepodobne generovane Algoritmom 4.44. Miller Rabinov test (Algoritmus 4.24) môže byť použitý na testovanie každého kandidáta na prvočíslo v krokoch 2 a 4, po vyradení kandidátov deliteľných malým prvočíslom menším než hraničné B. Pozri poznámku 4.45 pre dozor na zvolene B. Zatiaľ čo Miller Rabinov Test je pravdepodobnostný test prvočísiel, výstup tejto implementácie Gordonovho algoritmu je pravdepodobne prvočíslo.
(ii) Pozorným zvolením veľkosti prvočísiel s, t a parametrov i_0, j_0, môže kontrolovať presnú bitovú dĺžku výsledného prvočísla p. Poznačte si že bitová dĺžka r a s bude asi polovica z p, zatiaľ čo bitová dĺžka t trochu menšia ako r.

4.55 Definícia (čas výpočtu Gordonovho Algoritmu) Ak Miller Rabinov test je prvočíselný test použitý v krokoch 1,2 a 4, očakávaný čas trvania Gordonovho algoritmu na nájdenie silného prvočísla je iba asi 19% viac ako očakávaný čas Algoritmu 4.44 trvajúci na nájdenie náhodného prvočísla.

4.4.3 NIST metóda pre generovanie DSA prvočísiel

Niektoré schémy verejných kľúčov vyžadujú prvočísla uspokojujúce rôzne špeciálne požiadavky. Napríklad , NIST Digital Signature Algorithm(DSA paragraf 11.5.1) vyžaduje dva prvočísla p a q uspokojujúce nasledovné tri požiadavky:

(i) $2^{159}<q<2^{160}$; t.j. q je 160-bitové prvočíslo;

(ii) $2^{L-1}<p<2^{L}$ pre predpísané L, kde $L=512+64l$ kde $0\leq l\leq 8$; a

(iii) q delí p-1.

Táto sekcia predstavuje algoritmus pre generovanie takýchto prvočísiel p a q. V nasledujúcom, H značí SHA-1 hashovaciu funkciu(Algoritmus 9.53) ktorý mapuje bitové reťazce bitovej dĺžky$<2^{64}$ na 160-bitovych hash kódoch. Kde vyžadujú, celé číslo x v rozsahu $0\leq x<2^{g}$ ktorého binárna reprezentácia je $x=x_{g-1}2^{g-1}+x_{g-2}2^{g-2}+...+x_2 2^2+x_1 2+x_0$ mala by byť prekonvertovaná do g-bitovej postupnosti($x_{g-1}x_{g-2}...x_2x_1x_0$), a viac všestrannejší.

4.56 Algoritmus NIST metódy pre generovanie DSA prvočísiel

VSTUP: celočíselné l, $0\leq l\leq 8$.

VÝSTUP: 160-bitové prvočíslo q a L-bitové prvočíslo p, kde $L=512+64l$ a $q/(p-1)$.

1. Vypočítať $L=512+64l$. Použitím dlhého delenia od $(L-1)$ do 160, nájsť n, b tak že $L-1=160n+64l$, kde $0\leq b\leq 160$.
2. Opakovať nasledujúce:
   2.1 Vybrať náhodné jadro s (nemusí byť tajné) o dĺžke $g\geq 160$.
   2.2 Vypočítať $U=H(s)\oplus H((s+1) \bmod 2^{g})$.
   2.3 Zoradiť q podľa nastavenia do 1, najviac a najmenej významových bitov z U (Dbať na to, že q je 160-bitové nepárne celé číslo.)
   2.4 Testovať q na prvočíselnosť použitím MILLER-RABIN (q,t) pre $t\geq 18$ (viď odsek 4.57)
   Ak je q nájdené bude to (pravdepodobne) prvočíslo.
3. Nastav $i\leftarrow 0$, $j\leftarrow 2$.
4. Dokiaľ $i<4096$ vykonávať nasledujúce:
   4.1 Pre k od 0 do n vykonávať nasledujúce: nastaviť $V_k\leftarrow H((s+j+k) \bmod 2^{g})$.
   4.2 Pre celočíselné W , definované nižšie, nech $X=W+2^{L-1}$. (X je L-bitové celé číslo.)
   $W=V_0+V_1 2^{160}+V_2 2^{320}+...+V_{n-1}2^{160(n-1)}+(V_n \bmod 2^{b})2^{160n}$.
   4.3 Vypočítať $c=X \bmod 2q$ a nastaviť $p=X-(c-1)$ (Uvedomiť si, že $p\equiv 1 \pmod{2q}$)
   4.4 Ak $p\geq 2^{L-1}$ potom vykonávať následovné:
   Testovať p na prvočíselnosť použitím MILLER-RABIN(p,t) pre $t\geq 5$ (viď odsek 4.57).
   Ak p je (pravdepodobne) prvočíslo, potom vráť (q,p).
   4.5 Nastav $i\leftarrow I+1$, $j\leftarrow j+n+1$.
   4.6 Choď na krok 2.

4.57 Poznámka (k výberu testu prvočíselnosti v algoritme 4.56)

i. Dokument FIPS 186, v ktorom bol algoritmus 4.56 pôvodne opísaný, špecifikuje len robustný prvočíselný test, použitý v krokoch 2.4 a 4.4, napr. Test prvočíselnosti, kde pravdepodobnosť toho, že zložené číslo bude deklarované ako prvočíslo je maximálne $\left(\frac{1}{2}\right)^{80}$.

Ak je spravený heuristický predpoklad, že *q* je náhodne zvolené 160 bitové číslo podľa tabuľky 4.4, potom Miller-Rabinov(q,18) test je robustný test prvočíselnosti čísla *q*. Ak predpokladáme, že *p* je náhodne zvolené číslo dĺžky L-bitov, potom podľa tabuľky 4.4, potom Miller-Rabinov(p,5) test je robustný test prvočíselnosti čísla *p*. Miller-Rabinov test určí dané číslo ako prvočíslo s určitou pravdepodobnosťou.

ii. Kvôli zvýšeniu výkonu, čísla *p* a *q* (pravdepodobné prvočísla) by mali byť podrobené skúšobnému deleniu všetkými nepárnymi prvočíslami menšími ,ako nejaké číslo B, pred tým, než sa začne Miller-Rabinov test. Pozri poznámku 4.45, na spôsob určenia čísla B

4.58 Poznámka („slabé“ prvočísla nemôžu byť skonštruované úmyselne)
Algoritmus 4.56 má takú vlastnosť, že náhodné číslo s nie je vstupom samotného algoritmu na generovanie prvočísla, ale skôr vstupom na nepredikovateľný a nekontrolovateľný proces náhodného výberu, ktorého výstup je použitý ako náhodné číslo. Toto zabráni ovplyvneniu výstupného prvočísla vstupným číslom. Ak číslo s a počítadlo i sú zverejnené, potom ktokoľvek môže skontrolovať, že p a q boli vygenerované schválenou metódou. Táto vlastnosť chráni centrálnu autoritu, ktorá generuje p a q ako celosystémové parametre na použitie v DSA pred zámerným vytvorením „slabých“ prvočísel p a q, ktoré by mohli byť zneužité na odhalenie privátnych kľúčov iných entít.

4.4.4 Techniky tvorenia dokázateľných prvočísel

Maurerov algoritmus (algoritmus 4.62) generuje náhodné dokázateľné prvočísla, ktoré sú takmer rovnomerne rozložené okolo množiny prvočísel danej veľkosti. Predpokladaný čas na generovanie prvočísla je len o málo väčší ako ten, ktorý je potrebný na generovanie prevdepodobného prvočísla rovnakej dĺžky použitím algoritmu 4.44 s bezpečnostným parametrom t = 1.
Hlavná myšlienka algoritmu 4.62 je fakt 4.59, čo je malá modifikácia Pocklingtonovej teorémy (fakt 4.40) a fakt 4.41.

4.59 Definícia Nech $n \geq 3$ je nepárne číslo a nech splňuje podmienku $n = 1 + 2Rq$, kde q je nepárne prvočíslo. Ďalej predpokladajme, že $q > R$.

1. Ak existuje také číslo a, ktoré splňuje podmienku $a^{n-1} \equiv 1 (\mathrm{mod}\ n)$ a $\gcd(a^{2R} - 1, n) = 1$, potom n je prvočíslo.

2. Ak n je prvočíslo, potom pravdepodobnosť náhodného zvolenia základu a, $1 \leq a \leq n - 1$, splňuje podmienku $a^{n-1} \equiv 1 (\mathrm{mod}\ n)$ a $\gcd(a^{2R} - 1, n) = 1$ je $(1 - 1/q)$

Algoritmus 4.62 rekurzívne generuje nepárne prvočíslo q a potom vyberá náhodné čísla R, R<q, až pokiaľ n = 2Rq + 1 môže byť dokázateľné prvočíslo použitím 4.59 (1.) pre nejaký základ a. Podľa 4.59(2.) je veľkosť takého základu 1-1/q pre prvočíslo n. Na druhej strane, ak n je zložené číslo, potom pre väčšnu základov a nebude splňovať podmienku $a^{n-1} \equiv 1 \pmod n$.

4.60 **Poznámka** (popis konštánt c a m v algoritme 4.62)

1. Optimálna hodnota konštanty c definujúca skúšobné delenie číslom $B=ck^2$ v kroku 2 závisí na implementácii „long-integer“ aritmetiky, a je najlepšie určená experimentálne.

2. Konštanta m=20 zaisťuje, že I je najmenej 20 bitov dlhé a pretointerval, z ktorého je R vybrané, <I+1, 2I>, je dostatočne veľký (pre hodnoty k s praktickým významom), zvyčajne obsahuje aspoň jednu hodnotu R, pre ktorú n = 2Rq+1 je prvočíslo.

4.61 **Poznámka** (relatívna veľkosť r čísla q vzhľadom na n v algoritme 4.62)
Relatívna veľkosť r čisla q vzhľadom na n je definovaná ako r = log q/log n. Na zaistenie skutočnosti, že generované prvočíslo n je naozaj vybrané náhodne s nevyhnutne rovnomerným rozložením oklo množiny všetkých k – bitových prvočísel, veľkosť faktora prvočíselnosti q čísla n – 1, musí byť vybraný vzhľadom na pravdepodobnosť rozloženia najväčšieho faktora prvočíselnosti náhodne zvoleného k – bitového čísla. Vzhľadom na to, že q > R, relatívna veľkosť r čísla q je obmedzená intervalom <0.5, 1>. Môže to byť odvodené z 3.7 – kumulované rozloženie pravdepodobnosti relatívnej veľkosti r najväčšieho faktora prvočíselnosti veľkého náhodného čísla, pre dané r = 0,5 je (1+log r) pre $0,5 \le r \le 1$. V kroku 4 algoritmu 4.62, je relatívna veľkosť r generovaná vzhľadom na túto distribúciu výberom náhodného čísla $s \in <0,1>$ a nastavením $r = 2^{s-1}$ Ak k < 2m, potom r je vybrané ako najmenšia prípustná hodnota, menovite ½, na zaistenie toho, že interval, z z ktorého je vyberané R je dostatočne veľký.

4.62 Maurerov algoritmus na generovanie dokázateľných prvočísel

DOKÁZATEĽNÉ_PRVOČÍSLO(k)
VSTUP: kladné celé číslo k,
VÝSTUP: k-bitové prvočíslo n.
1. (Ak k je malé číslo, potom testuj pomocou skúšobného delenia náhodné celé čísla, tabuľka malých prvočísel má byť prepočítaná pre tento účel.)
Ak k<=20, potom vykonávaj opakovane nasledujúce:
1.1 Vyber náhodné k-bitové nepárne číslo n.
1.2 Použi skúšobné delenie pre všetky prvočísla menšie ako $\sqrt{n}$ na určenie toho, či je n prvočíslo.
1.3 Ak n je prvočíslo, vráť(n).
2. Nastav $c \leftarrow 0.1$ a $m \leftarrow 20$ (pozri vetu 4.60).
3. (Obmedzenie skúšobného delenia) Nastav $B \leftarrow c.k^2$ (pozri vetu 4.60).

4. (Vygeneruj r, veľkosť q porovnaj s n – pozri vetu 4.61) Ak k>m, opakovane vykonávaj nasledujúce : vyber náhodné číslo s z intervalu (0,1), nastav $r \leftarrow 2^{s-1}$ pokiaľ (k - rk)>m, inak(napr. $k \leq 2m$) nastav $r \leftarrow 0.5$.
5. Vypočítaj $q \leftarrow$ DOKÁZATEĽNÉ_PRVOČÍSLO([r.k] + 1).
6. Nastav $[I \leftarrow 2^{k-1}/(2q)]$.
7. Úspech $\leftarrow 0$.
8. Pokiaľ (úspech=0) vykonávaj nasledujúce :
   8.1 (Výber kandidáta n) Vyber náhodné číslo R z intervalu[I+1,2I] a nastav $n \leftarrow 2Rq+1$.
   8.2 Použi skúšobné delenie na určenie toho, či n je deliteľné nejakým prvočíslom<B.
   Ak to neplatí vykonaj nasledujúce:
   Vyber náhodné číslo a z intervalu[2, n-2]
   Vypočítaj $b \leftarrow a^{n-1} \bmod n$.
   Ak b=1, vykonaj nasledujúce :
   Vypočítaj $b \leftarrow a^{2R} \bmod n$ a $d \leftarrow \gcd(b-1, n)$.
   Ak d=1 potom úspech $\leftarrow 1$.
9. Vráť(n).

4.63 Poznámka (Vylepšenie algoritmu 4.62)

(i) Zrýchlenie sa môže dosiahnuť použitím vety 4.42 namiesto vety 4.59(i) pre vyskúšanie prvočísla $n = 2Rq+1$ v kroku 8.2 Maurerovho algoritmu – veta 4.42 požaduje iba to, aby q bolo väčšie ako $\sqrt[3]{n}$.

(ii) Ak kandidát n prejde skúšobným delením(v kroku 8.2), potom Miller-Rabinov test(algoritmus 4.24) s jediným koreňom a = 2 má byť prevedený na n; iba ak n prejde týmto testom, bude zabezpečená jeho prvočíselnosť(zvyšok kroku 8.2). To vedie k rýchlejšej realizácii s riadnou efektivitou Miller-Rabinovho testu s jediným koreňom a = 2.

(iii) Krok 4 vyžaduje požitie reálneho číselnej aritmetiky keď počíta 2^{k-1}. Týmto výpočom sa dá vyhnúť, jeden sa prepočíta a uloží sa zoznam takýchto hodnôt pre výber náhodných čísel $s \in [0,1]$.

4.64 Poznámka (pravdepodobné a dokázateľné prvočísla) Dokázateľné prvočísla sú výhodnejšie ako pravdepodobné prvočísla. V algoritme 4.44 na generovanie dokázateľných prvočísel s t=1 je nepatrne rýchlejšii ako Maurov algoritmus. Okrem toho, ten druhý vyžaduje viac .... pamäte potrebnej k rekurzívnemu(opätovnému) použitiu. Dokázateľné prvočísla sa preferujú pred pravdepodobnými prvočíslami v tom zmysle, že tie predchadzajúce majú pradepodobnosť nulovej chyby. V niektorých kryptografických aplikáciach, hoci je tu vždy nenulovej chybovej pravdepodobnosti niektorých katastrofických porúch(zlyhaní), takých ako protikladný dohad(tušenie) tajný kľúč alebo hardwarové zlyhanie. Pretože chybová pravdepodobnosť pravdepodobných prvočísel môže byť účinné znížená do akceptovaných nízkych úrovní, nezdá sa byť žiaden dôvod na oprávnenie použitia dokazateľných prvočísel pred pravdepodobnými prvočíslami.

Príklad **GORDON algoritmus**

1. Vygenerujeme dva náhodné prvočísla s a t so zhruba rovnakou bitovou dĺžkou. Môžeme použiť napríklad Miller-Rabin algoritmus.
   $s = 12049$
   $t = 8123$
2. Zvolíme celé číslo i_0. Nájdeme prvé prvočíslo v postupnosti $2it + 1$, pre $i = i_0, i_0 + 1, \ldots$
   $i_0 = 1$ zvolíme.
   $r_1 = 2.1.8123 + 1 = 16247$ Nie je prvočíslo.
   $r_2 = 2.2.8123 + 1 = 32493$ Nie je prvočíslo.
   $r_3 = 2.3.8123 + 1 = 48739$ Nie je prvočíslo.
   $r_4 = 2.4.8123 + 1 = 64985$ Nie je prvočíslo.
   $r_5 = 2.5.8123 + 1 = 81231$ Nie je prvočíslo.
   $r_6 = 2.6.8123 + 1 = 97477$ Nie je prvočíslo.
   $r_7 = 2.7.8123 + 1 = 113723$ Je prvočíslo. Potom $r = r_7$.
3. $p_0 = 2.\left(12049^{113721} \bmod 113723\right)12049 - 1 = 1978325309$
4. Zvolíme celé číslo j_0. Nájdeme prvé prvočíslo v postupnosti $p_0 + 2jrs$, pre $j = j_0, j_0 + 1, \ldots$
   $j_0 = 1$ zvolíme.
   $p_1 = 1978325309 + 2.1.(113723.12049) = 2253752163$ Nie je prvočíslo.
   $p_2 = 1978325309 + 2.2.(1370248427) = 25264249017$ Nie je prvočíslo.
   $p_3 = 1978325309 + 2.3.(1370248427) = 28004745871$ Nie je prvočíslo.
   $p_4 = 1978325309 + 2.4.(1370248427) = 30745242725$ Nie je prvočíslo.
   $p_5 = 1978325309 + 2.5.(1370248427) = 33485739579$ Nie je prvočíslo.
   $p_6 = 1978325309 + 2.6.(1370248427) = 36226236433$ Nie je prvočíslo.
   $p_7 = 1978325309 + 2.7.(1370248427) = 38966733287$ Nie je prvočíslo.
   $p_8 = 1978325309 + 2.8.(1370248427) = 41707230141$ Nie je prvočíslo.
   $p_9 = 1978325309 + 2.9.(1370248427) = 44447726995$ Nie je prvočíslo.
   $p_{10} = 1978325309 + 2.10.(1370248427) = 47188223849$ Nie je prvočíslo.
   $p_{11} = 1978325309 + 2.11.(1370248427) = 49928720703$ Nie je prvočíslo.
   $p_{12} = 1978325309 + 2.12.(1370248427) = 52669217557$ Nie je prvočíslo.
   $p_{13} = 1978325309 + 2.13.(1370248427) = 55409714411$ Je prvočíslo.

   Preto $p = p_{13} = 55409714411$.

Príklad.1 **RANDOM-SEARCH(21,3)**

1. $n=(10111110111110101000 01)_B = (1564321)_D$
2. $B = c.k^2 = 0{,}1.21^2 = 44{,}1 \cong 45$
   Ľahko zistíme, že 11 delí 1564321. Vraciame sa preto na krok 1.

1′. $n=(10111110111110101001 11)_B = (1564327)_D$
2′. Podobne aj tu 19 delí 1564327.

1′′. $n=(10111110111110101001 11)_B = (1564331)_D$
2′′. Žiadne z daných čísel nie je deliteľom 1564331.
3′′. MILLER-RABIN test:

1. $n-1 = 2^s.r$: $1564330 = 2^1.782156$
2. pre i=1 až 3:

2.1.Náhodné a= 17.

2.2. $y = a^r \ mod \ n = 17^{782156} \ mod 1564331 = 596018$

2.3. $y \neq 1 \wedge n-1$

j=1;
1>0;
$y \neq 1 \Rightarrow$ „zložené“

i=2:

2.1.Náhodné a=2473.

2.2. $y = a^r \ mod \ n = 2473^{782156} \ mod 1564331 = 1222171$

2.3. $y \neq 1 \wedge n-1$

$y \neq 1 \Rightarrow$ „zložené“

i=3:

2.1.Náhodné a=473823

2.2. $y = a^r \ mod \ n = 473823^{782156} \ mod 1564331 = 173304$

2.3. $y \neq 1 \wedge n-1$

$y \neq 1 \Rightarrow$ „zložené“

Záver Millera-Rabina: Dané číslo je zložené.

1′′′. $n=(10111110111110101100 01)_B = (1564337)_D$
2′′′. Žiadne z daných čísel nie je deliteľom 1564337.
3′′′. MILLER-RABIN test:

1. $n-1 = 2^s.r$: $1564336 = 2^4.97771$
2. pre i=1 až 3:

2.1. Náhodné a= 23.

2.2. $y = a^r \ mod \ n = 23^{97771} \ mod 1564337 = 565857$

2.3. $y \neq 1 \wedge n-1$

j=1;
1<3:

$y = y^2 \ mod \ n = 565857^2 = 954278$

j=2

2<3:

$y = y^2 \ mod \ n = 954278^2 = 567811$

j=3

3=3

$y = y^2 \ mod \ n = 567811^2 = 1040358$

$y = n - 1 \Rightarrow$ „zložené“

2.1´. Náhodné a=3.

2.2´. $y = a^r \ mod \ n = 3^{97771} \ mod \ 1564337 = 406869$

2.2´. $y \neq 1 \wedge n - 1$

j=1;

1<3:

$y = y^2 \ mod \ n = 406869^2 = 1113147$

j=2

2<3:

$y = y^2 \ mod \ n = 1113147^2 = 549279$

j=3

3=3

$y = y^2 \ mod \ n = 549279^2 = 1564336$

$y = n - 1 \Rightarrow$ „prvočíslo“

Záver Miller-Rabin: Prvočíslo

Záver Random-Search: Prvočíslo.

Priklad . 2 Maurerov algoritmus na generovanie dokázateľných prvočísel

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
#include "Function.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"

int dlzka_k, k, k1, m, succes, B, succes1;
double c, r;
unsigned long n, q, I, R, a, b, d;
int increment=1;

TForm1 *Form1;
//---------------------------------------------------------------------------
```

```
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 /*nacitanie*/
 /*****************************************************/
 dlzka_k=Edit1->GetTextLen();
 dlzka_k++;
 char *pom= new char[dlzka_k];
 Edit1->GetTextBuf(pom,dlzka_k);
 k=atoi(pom);
 /*****************************************************/
 /*vizualizacna uprva*/
 Button1->Cursor=crHourGlass;
 /*************************/

 if(k<=20)
   {
   n=podprogram1(k);
   /********vypis*************/
   char *pom1= new char[50];
   itoa(n,pom1,10);
   Edit2->Text=pom1;
   /*************************/
   return;
   }
 m=20;
 c=0.1;
 B=(int)(c*k*k);
 r=0.5;
 k1=(int)(r*(double)k)+1;
 q=podprogram1(k1);
 double k_d;
 k_d=(double)(k);
 k_d--;
 I=(double)(((unsigned long)(pow(2,k_d))/(2*q)));
 succes=0;
 while(succes==0)
   {
   /*********************************************/
   /*do
     {
      R=random(2*I);
    }while(R<(I+1));                          */
   /*****************************************************/
   /*******Modifikacie-pseudonahodne vybery**************/
   /**1**/
```

```
   if(increment==1) R=I+1;
   else
     { R++;
      if(R>=2*I)
       {
        char *pom2= new char[50];
        pom2="Somarina";
        Edit2->Text=pom2;
        Button1->Cursor=crDefault;
        return;
       }
     }
   /**2**/
   /*do
     {
      if(increment==1) R=random(2*I);
      else
        {
         R=R+1;
         if(R >=2*I){R=random(2*I); increment=1;}
        }
     }while(R <=(I+2));*/

   n=(2*R*q)+1;
   succes1=test(n,B);
   if(succes1==1)
     {
      do
        {
         a=random(n-2);
        }while(a<=2);
      b=(unsigned long)modul((double)a,(double)(n-1),(double)n);
      if(b==1)
        {
         b=(unsigned long)modul((double)a,(double)(2*R),(double)n);
         d=GCD(b-1,n);
         if(d==1) succes=1;
        }
     }
  increment++;
  }
 /***vypis*********************************************/
 char *pom2= new char[50];
 itoa(n,pom2,10);
 Edit2->Text=pom2;
 Button1->Cursor=crDefault;

}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
 Close();
}
//---------------------------------------------------------------------------


#include <stdlib.h>
#include <math.h>

int RNG(int);
int test(int, int);
double modul(double, double, double);
double GCD(double, double);

//---------------------------------------------------------
unsigned long podprogram1(int k)
{
 int succes=0, b;
 unsigned long n;
 while(succes==0)
  {
   n=RNG(k);
   b=sqrt(n);
   succes=test(n, b);
  }
 return(n);
}
//---------------------------------------------------------
//---------------------------------------------------------
 int RNG(int k)
 {
  double delitel;
  int n, max, min;
  randomize();
  do
    {
     max=(int)(pow(2,(double)k)-1);
     min=(int)(pow(2,(double)(k-1)));
     n=random(max);
    }while(n<min);
  return n;
 }
//---------------------------------------------------------
//---------------------------------------------------------
 int test(int n, int b)
 {
  int i=2, succes=0, odmoc_n=b;
  while(i<=odmoc_n)
```

```
   {
    int pom, zvys;
    pom=n/i;
    zvys=(n-(pom*i));
    if(zvys==0)
      {
       succes=0;
       return succes;
      }
    i++;
   }
  succes=1;
  return succes;
 }
//-------------------------------------------------------
//-------------------------------------------------------
 double modul(double x, double y, double z)
 {
  /*double x1=x;
  for(int i=1; i<y; i++)
    {
     x1=fmod((x1*x),z);
    }
  return x1;*/
  /***************INAC*****************************/
  unsigned long zv[32], i=0, index=0;
  unsigned long y0,y1;
  for(i=0;i<32;i++) zv[i]=0;
  y0=(unsigned long)(y);
  y1=(unsigned long)(y);
  while(y1!=0)
    {
     y1=y0/2;
     zv[index]=fmod(y0,2);
     index++;
     y0=y1;
    }
  unsigned long *v;
  long double pom, sucin;
  v=(unsigned long*)calloc(index,sizeof(unsigned long));
  v[0]=x;
  for(i=1; i<index; i++)
    {
     pom=(long double)(v[i-1])*(long double)(v[i-1]);
     v[i]=(unsigned long)fmodl(pom,(long double)(z));
    }
  sucin=1;
  for(i=0; i<index; i++)
    {
     if(zv[i]==1)
```

```
      {
       pom=sucin*(long double)v[i];
       sucin=fmodl(pom,(long double)(z));
      }
    }
  free(v);
  return (double)sucin;
 /*********************************************/
 }
//-------------------------------------------------------
//-------------------------------------------------------
double GCD(double x, double y)
 {
  double pom, r;
  if(x<y)
    {
     pom=x;
     x=y;
     y=pom;
    }
  while(y!=0)
    {
     r=fmod(x,y);
     x=y;
     y=r;
    }
  return x;
 }
//-------------------------------------------------------
```