1 Úvod

Cieľom projektu je štúdium asymetrickej šifry RSA a následná implementácia RSA, tj. generovanie kľúčov, šifrovanie, dešifrovanie a prelomenie šifry s dôrazom na efektivitu výsledného riešenia. RSA je asymetrický šifrovací algoritmus s verejným kľúčom, ktorý je založený na probléme faktorizácie veľkých čísiel. Projekt je implementovaný v jazyku C a používa funkcie z knižnice GMP[1], ktorá poskytuje aritmetiku s ľubovoľnou presnosťou nad celými, racionálnymi číslami a číslami s pohyblivou rádovou čiarkou.

2 Generovanie kľúčov

Implementovaný program obsahuje generovanie súkromných a verejných kľúčov RSA. Spúšťa sa prepínačom `-g` a parametrom B pre tento prepínač je požadovaná bitová veľkosť verejného modulu. Vo funkcii `compute_p_q_n` sa na začiatku za pomoci funkcie `generate_prime` vygenerujú dva prvočísla p a q o bitovej veľkosti $B/2$, u nepárnej hodnoty B o veľkosti $(B/2)+1$ a $B/2$. Pre získanie týchto prvočísiel sa najskôr vygenerujú dva náhodne čísla pomocou funkcie `mpz_urandomb` so seedom nastaveným podľa interných hodín zariadenia. Prvé dva najvýznamnejšie bity sa nastavia na 1 pre zaistenie požadovanej dĺžky verejného modulu. Najmenej významný bit sa nastaví tiež na 1 pre vylúčenie párnych čísiel. Následne prebieha vo funkcii `is_prime_numer` overovanie, či čísla sú prvočíslami. Dôležitá je aj kontrola, či prvočíslo p nie je rovnaké ako prvočíslo q - ak áno, proces generovania sa opakuje.

Knižnica GMP používa Fermatov test pred Miller Rabinov testom na zvýšenie efektívnosti samotného overovania prvočíselnosti a toto chovanie bolo implementované aj v tomto projekte. Nad dvomi nahodnými číslami sa spúšťa Fermatov test (implementovaný vo funkcii `fermat_test`), ktorý vie rozhodnúť, či číslo je pravdepodobne prvočíslo, alebo či sa jedná o zložené číslo. Ak sa pravdepodobne jedná o prvočíslo, spúšťa sa finálne overenie pomocou Miller Rabinovho testu (funkcia `miller_rabin_test`), ktorý rozhodne či číslo je skutočne prvočíslo. Počet iterácií Miller Rabin je 64, čo je aj hodnota použitá v knižnici libgcrypto. Tento počet iterácii je dostačne veľký na to, aby sa výrazne znížila pravdepodobnosť chyby na približne $(1/4)^{64}$.

Fermatov test prvočíselnosti vychádza z malej Fermatovej vety[2], teda že pre každé prvočíslo platí $a^{p-1} \equiv 1 \mod p$. Ak rovnosť pre testované číslo p neplatí, tak určite nie je prvočíslom, ak platí, tak dané číslo je pravdepodobne prvočíslo. Miller Rabinov test[3] je podobný Fermatovmu testu. Používa sa vlastnosť, ktorá pre prvočísla platí bez výnimiek, ale zložené čísla ju spĺňať nemusia. Rovnako ako v prípade Fermatovho testu súvisí táto vlastnosť s umocňovaním modulo testované číslo.

Po získaní prvočísel p a q dochádza k výpočtu hodnoty $\phi(n)$ podľa vzťahu $\phi(n) = (p-1) \cdot (q-1)$. Hodnota $\phi(n)$ sa používa na výpočet verejného exponentu e. Výpočet oboch hodnôt je implementovaný vo funkcii `compute_e_phi`. Verejný exponent e je vygenerovaný z rozsahu $[1, \phi(n)]$ a musí platiť, že $gcd(e, \phi(n)) = 1$. Ak tento vzťah neplatí, vygeneruje sa nový verejný exponent e, a test nesúditeľnosti e s $\phi(n)$ sa zopakuje. Na výpočet najväčšieho spoločného deliteľa je v projekte použitý rozšírený Euklidov algoritmus, ktorý je implementovaný vo funkcii `extended_euclidean`.

Poslednou počítanou hodnotou je súkromný exponent d, ktorý sa vypočíta ako $d = e^{-1} \mod \phi(n)$, čo je multiplikatívna inverzia e v module $\phi(n)$, ktorú je možné vypočítať rozšíreného Euklidovho algoritmu. Nasleduje kontrola, či hodnoty e a d sa nerovnajú, čo by znamenalo, že oba kľúče sú identické - ak kontrola zlyhá, opakuje sa celý doterajší proces generovania kľúčov. Kontrolovaná nerovnosť taktiež určuje, že hodnoty B musia byť väčšie ako 6, keďže pre $B <= 6$ nie je možné vygenerovať platné kľúče a zároveň splniť všetky nutné podmienky. Výstupom programu sú hodnoty p, q, e, n, d oddelené medzerou a výstup je zakončený znakom nového riadka.

3 Šifrovanie a dešifrovanie správ

Šifrovanie sa spúšťa prepínačom `-e` a vyžaduje tri dodatočné parametry E (verejný exponent v hexadecimálnom formáte), N (verejný modul v hexadecimálnom formáte) a M (správa na zašifrovanie vo forme čísla). Pred začatím procesu šifrovania program skontroluje podmienku, či M je väčšie ako 0 a zároveň menšie alebo rovné N. Ak táto podmienka nie je dodržaná, program vypíše chybu o nemožnosti zašifrovania správy M. Ak podmienka platí, dochádza k procesu šifrovania správy M. Zašifrovaná správa C sa získa výpočtom podľa vzťahu $C = M^e \mod N$ a je následne vypísaná na výstup programu v hexadecimálnom formáte.

[1] `https://gmplib.org/`
[2] `https://link.springer.com/referenceworkentry/10.1007%2F978-1-4419-5906-5_449`
[3] `http://www.sciencedirect.com/science/article/pii/0022314X80900840`

Dešifrovanie sa spúšťa prepínačom `-d` a vyžaduje tri dodatočné parametry D (súkromný exponent v hexadecimálnom formáte), N (verejný modul v hexadecimálnom formáte) a C (zašifrovaná správa v hexadecimálnom formáte). Ak N je 0, program končí chybou. Po tomto overení dochádza k procesu dešifrovania správy C. Pôvodná správa M sa získa výpočtom podľa vzťahu $M = C^d \mod N$ a je následne vypísaná na výstup programu v hexadecimálnom formáte.

4 Prelomenie RSA - faktorizačná metóda Pollard ρ Brent

Lámanie RSA je implementované vo funkcii `break_rsa`. Spúšťa sa prepínačom `-b` a vyžaduje tri dodatočné parametry E (verejný exponent v hexadecimálnom formáte), N (verejný modul v hexadecimálnom formáte) a C (zašifrovaná správa v hexadecimálnom formáte). Ak sa jedná o triviálny prípad párneho verejného modulu, je jedným z prvočísiel číslo 2. Pre netriviálne prípady je potrebné použiť faktorizáciu. Na faktorizáciu verejného modulu bola vybratá metóda Pollard ρ [4] a to hlavne kvôli nízkej priestorovej zložitosti a vyššej rýchlosti oproti Fermatovej a Pollard $\rho-1$ metóde. Jedná sa o pravdepodobnostnú metódu Monte Carlo využívajúcu narodeninového paradoxu. Patrí do exponenciálnej triedy zložitosti. Rýchlosť metódy sa značne odvíja od vygenerovaných hodnôt. Vo funkcii `pollard_rho_brent` je implementovaná metóda Pollard ρ s Brentovou modifikáciou [5]. Pseudokód implementovaného algoritmu je nasledujúci:

Algoritmus 1: Metóda Pollard ρ Brent

Vstup: Číslo n
Výstup: Deliteľ g

```
Vygeneruj náhodne hodnoty y, c , m z intervalu [1, n − 1].
Polynóm f(x) je definovaný vzťahom f(x) = x^2 + c mod n.
g ← 1, r ← 1, q ← 1
while g = 1 do
    x ← y
    for i ← 1 to r do
        y ← f(y)
    end
    k ← 0
    while k < r and g = 1 do
        ys ← y
        for i ← 1 to min(m, r − k) do
            y ← f(y)
            q ← q · |x − y| mod n
        end
        g ← gcd(q, n)
        k ← k + m
    end
    r ← 2r
end
if g = n then
    while g = 1 do
        ys ← f(ys)
        g ← gcd(|x − ys|, n)
    end
end
return g
```

Algoritmus využíva polynóm f pre generovanie presudonáhodnej postupnosti $x_{n+1} = f(x_n)$. Základná verzia Pollard ρ postupne generuje hodnoty $x = f(x)$ a $y = f(f(y))$ dokým x sa nerovná y. Cyklus bol nájdený ak hodnota x sa rovná hodnote y a výpočet následne končí. V každej iterácii sa tiež počíta hodnota d podľa vzťahu $d = \gcd(|x - y|, n)$. Ak hodnota d je väčšia ako 1, algoritmus našiel deliteľa čísla n. U tohto algoritmu sa môže stať, že nenájde deliteľa zloženého čísla. Riešením tohto problému je zmena polynómu f a opakovanie behu algoritmu.

Brentova varianta vychádza zo základnej verzie algoritmu Pollard ρ, kde pôvodný spôsob detekcie cyklu (Floydova detekcia cyklu) je nahradený Brentovou detekciou cyklu. Oproti základnej verzie je Brentova metova rýchlejšia z dôvodu využitia znalosti, že ak $\gcd(x, n) > 1$, potom aj $\gcd(ax, n) > 1$ pre celé kladné číslo a, a

[4] `https://link.springer.com/article/10.1007%2FBF01933667`
[5] `http://wwwmaths.anu.edu.au/~brent/pd/rpb051i.pdf`

preto každej iterácii nie je potrebné počítať $\gcd(|x-y|, n)$. Algoritmu stačí vypočítať súčin niekoľko po sebe idúcich $|x-y|$ a až potom hľadať najväčšieho spoločného deliteľa pre vypočítaný súčin a číslo n.

Výstupom faktorizácie verejného modulu N je jedno z prvočísel. Druhé prvočíslo je získané vydelením verejného modulu nájdeným prvočíslom. Hodnota $\phi(n)$ sa vypočíta ako $\phi(n) = (p-1) \cdot (q-1)$. Ďalej je potrebné vypočítať súkromný exponent d podľa vzťahu $d = e^{-1} \mod \phi(n)$. Pôvodná správa M sa následne získa podľa vzťahu $M = C^d \mod N$. Prvočísla p a q sú spoločne so správou M vypísané na výstup programu v hexadecimálnom formáte.

4.1 Meranie časovej náročnosti lámania RSA

Na svojom notebooku s procesorom Intel Haswell 4720HQ som pre rôzne bitové veľkosti verejného modulu vykonal 20 meraní, kde som skúmal čas, za aký môj program prelomí RSA. Výsledky meraní sú znázornené v tabuľke 4.1.

Veľkosť verejného modula (b)	Priemer (s)	Medián (s)	Minimum (s)	Maximum (s)
80	0,1131	0,0826	0,0784	0,1615
90	1,3994	1,1589	0,5705	4.5256
100	9,8536	9,6132	2,3819	20,1156
110	52,6879	47,0838	18,4291	103,7809

Tabuľka 1: Výsledné časy potrebné na prelomenie RSA pre dané bitové veľkosti

Ako vidíme v tabuľke 4.1, pre účely tohto projektu a s ohľadom na testované bitové veľkosti okolo 96 bitov je algoritmus *Pollard ρ Brent* dostatočne rýchly, avšak pri reálne používaných bitových veľkostiach verejného modulu (2048 / 4096) by lámanie trvalo veľmi dlho a teda je vhodné na rýchlejšie lámanie RSA použiť efektívnejšie algoritmy - napr. *General number field sieve*, resp. *Shorov algoritmus*, ktorý existuje pre kvantové PC.