Dokumentácia k projektu z predmetu KRY : *RSA*

Anton Firc (xfirca00)

3. mája 2020

1 Úvod

Cieľom projektu je implementácia algoritmu *RSA* ako konzolovej aplikácie v C++. Konkrétne generovania kľúčov na základe dĺžky verejného modulu, šifrovanie a dešifrovanie správ. Druhou časťou projektu bola implementácia prelomenia tohoto šifrovania pri použití slabého verejného modulu.

2 Implementácia

Aplikácia je delená na moduly. Obsahuje tri samostatné moduly pre generovanie kľúčov, šifrovanie / dešifrovanie a prelomenie šifry. Tieto moduly sú následne prepojené konzolovou aplikáciou ktorá obsluhuje užívateľské vstupy a volá príslušnú funkcionalitu modulov. Môže sa zdať, že moduly obsahujú duplicitný kód, je to však z dôvodu zachovania samostatnosti modulov. Pre prácu s veľkými číslami bola zvolená knižnica GMP[1].

2.1 Generovanie kľúčov

Generovanie kľúčov je implementované triedou `KeyGenerator`. Dĺžka generovaných kľúčov je daná užívateľom zadanou dĺžkou verejného modulu. Ako prvé prebehne určenie verejného exponentu podľa dĺžky verejného modulu. Pre dĺžky verejného modulu menšie ako 2048b je verejný exponent 3, inak 65537. Následne sú generované prvočísla p, q. Najprv sú vygenerované dve náhodné nepárne čísla. Každé číslo je následne otestované metódou Solovay-Strassen[2]. Pokiaľ číslo nebolo vyhodnotené ako prvočíslo, je inkrementované o 2 a test prebieha znovu. Po vygenerovaní oboch prvočísel je vypočítané $\Phi = (p-1) * (q-1)$ a nájdený najväčší spoločný deliteľ(gcd) verejného exponentu a Φ. Pokiaľ sa gcd nerovná 1, sú prvočísla generované znova, až kým nedôjde k splneniu tejto podmienky. Následne je spočítaný verejný modulus $N = p * q$. Posledne je pomocou rozšíreného euklidovho algoritmu[3] vypočítaný súkromný exponent $d = inv(e, \Phi)$. Súkromný resp. verejný kľúč je následne tvorený dvojicou (d, n) resp. (e, n).

2.1.1 Test prvočísla

Bola využitá metóda Solovay-Strassen. Test touto metódou je založený na princípe modulárnej ekvivalencie, alebo číselnej kongruencie podľa operácie modulo n. Obecne dochádza ku skúmaniu k čísel, kde je každé číslo pre ktoré táto ekvivalencia platí označené ako eulerov svedok. Platnosť tejto kongruencie však nezaručuje, že je testované číslo naozaj prvočíslom. Je teda vhodné vykonať toto testovanie pre viac rôznych čísel. V projekte je nastavený počet opakovaní na 50.

$$a^{(n-1)/2} \equiv \left(\frac{a}{n}\right) \pmod{n}$$

Kde ľavú stranu tejto kongruencie tvorí *Jacobiho symbol*[4]. Ten môže nadobudnúť len hodnoty z množiny $\{-1, 0, 1\}$.

[1] https://gmplib.org/
[2] https://en.wikipedia.org/wiki/Solovay%E2%80%93Strassen_primality_test
[3] https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm
[4] https://en.wikipedia.org/wiki/Jacobi_symbol

2.2 Šifrovanie a dešifrovanie

Šifrovanie a dešifrovanie je implementované triedou `CipherModule`. Šifrovanie a dešifrovanie prebieha pomocou toho istého algoritmu. Správa resp. šifra je vynásobená verejným resp. súkromným kľúčom a ako výsledok získame šifru resp. dešifrovanú správu. Metódy `cipher(message, exponent, modulus)` a `decipher(message, exponent, modulus)` prijjímajú ako argument správu resp šifru, súkromný resp. verejný exponent a verejný modulus. Výsledok je následne spočítaný ako $message^{exponent} \mod modulus$.

2.3 Prelomenie RSA

Pre prelomenie šifrovania je potrebné poznať verejný modulus. Keďže je modulus tvorený násobkom dvoch prvočísel, dokážeme ho faktorizovať, a získať prvočísla ktorých súčin tvorí, teda p, q. Po získaní p, q a verenjného exponenta už nie je problém dopočítať súkromného exponenta a dešifrovať správu. Tento postup však funguje v rozumnom čase len pre šifry so slabým verejným modulom (cca 96b).

Pre faktorizáciu verejného modulu bol zvolený Pollardov Rho algoritmus[5] ktorý je jednoduchý na implementáciu a ako sa ukázalo aj dostatočne efektívny.

```
x = 2
y = 2
d = 1

while d == 1:
    x = g(x)
    y = g(g(y))
    d = gcd(|x - y|, n)

if d == n:
    return failure
else:
    return d
```

Algoritmus hľadá netriviálny faktor p verejného modulu ($n = p * q$) pomocou generovania pseudo-náhodných postupností použitím operácie polynomiálne modulo $g(x) = (x^2 + 1) \mod n$. Ako počiatočnú hodnotu volí číslo 2 a pokračuje ako $x_1 = g(2), x_2 = g(g(2)), x_3 = g(g(g(2)))$. Táto postupnosť je príbuzná postupnosti $x_k \mod p$, p však dopredu nedokážeme odhadnúť a nemôžeme teda túto postupnosť algoritmicky vypočítať.

Množina hodnôt pre tieto postupnosti je ale konečná, takže v určitom okamihu v čase sa sekvencie začnú opakovať. Predpokladáme, že sa sekvencie spávajú ako náhodné čísla, a teda vzniká narodeninový paradox, a preto predpokladáme že počet x_k pred dosiahnutím opakovania sekvencie bude $O(\sqrt{N})$ kde N značí počet možných hodnôt.

Algoritmus obsahuje dva uzly, x a y. Kde uzol x v každom opakovaní postúpi o jeden krok dopredu a uzol y o dva kroky dopredu. Následne je hľadaný najväčší spoločný deliteľ rozdielu x,y a verejného modulu ($gcd(|x - y|, n)$). Pokiaľ ($gcd((\text{gcd}(\text{—x-y—}, \text{n}), n) \neq 1$ dochádza k cykleniu sekvencií a rozdiel $|x - y|$ je teda násobkom p. Táto situácia nastane v každom prípade, môže sa však stať, že výsledný faktor bude samotný verejný modulus. V tomto prípade skončí algoritmus neúspechom.

Po získaní faktoru p je možné ľahko dopočítať faktor q, ako $q = \frac{n}{p}$. Následne je rovnako ako pri generovaní kľúčov za použitia rozšíreného euklidovho algoritmu vypočítaný súkromný exponent d, a správa dešifrovaná.

[5] https://en.wikipedia.org/wiki/Pollard%27s_rho_algorithm