VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DOKUMENTACE K PROJEKTU DO PŘEDMĚTU KRY

Projekt 2

Bc. Lukáš Pelánek, xpelan03@stud.fit.vutbr.cz
28. dubna 2018

Úvod

Cílem tohoto projektu bylo naimplementovat konzolovou aplikaci v C++, která dokáže generovat klíče RSA na základě délky veřejného modulu. Šifrovat a dešifrovat zprávu pomocí zadaného veřejného/soukromého exponentu, veřejného modulu a zprávy/šifry k šifrování/dešifrování. A v posledním případě prolomit šifru na základě slabého veřejného modulu. Pro práci s velkými čísly byla ze zadání určena knihovna *GMP*[1].

Generování klíčů

Klíče jsou generovány na základě požadované délky veřejného modulu. Délka veřejného modulu musí být minimálně 6 bitů. Nejprve dojde k vybrání veřejného exponentu *e*. Pokud je požadovaná délka veřejného modulu větší než 2048, tak veřejný exponent bude 65537. V opačném případě 3. Tím docílíme rychlejšího generování klíčů.

Dalším krokem je zvolit náhodně 2 prvočísla *p*, *q* o délce poloviny požadované délky veřejného modulu a spočítat hodnotu *phi*. Ta je spočítána jako *phi* = (*p* - 1) * (*q* - 1). Pokud největším společným dělitelem *phi* a veřejného exponentu *e* je číslo 1, pak prvočísla přijmeme, v opačném případě generujeme nová prvočísla dokud největší společný dělitel není 1. Následně spočítáme veřejný modulus *n* jako *n* = *p* * *q*.

Poslední částí algoritmu je spočítání soukromého exponentu *d* pomocí operace nalezení inverzního prvku[2] *inv*(*e*, *phi*). Tato metoda očekává, že největší společný dělitel vstupních parametrů je číslo 1 a využívá rozšířený Euklidův algoritmus. Složitost této metody je O(Log *n*). Soukromý, resp. veřejný klič je poté dvojice (*d*, *n*), resp. (*e*, *n*).

Šifrování a dešifrování

Šifrování a dešifrování se provádí pomocí stejného algoritmu. Mezi očekávanými vstupními parametry je veřejný nebo soukromý exponent *e*, veřejný modulus *n* a zpráva k zašifrování nebo šifra k dešifrování *m*. Výsledek je poté spočítán jako *m^e % n*. Pro tuto operaci jsem využil funkci knihovny *GMP mpz_powm.*

Prolomení RSA

Poslední funkcionalitou aplikace je prolomení RSA na základě slabého veřejného modulu. Veřejný modulus je nejprve zkontrolován metodou zkusmého dělení. Pokud dělitel není nalezen, tak je zavolána Pollardova faktorizační metoda[3].

---

[1] https://gmplib.org/
[2] https://www.geeksforgeeks.org/multiplicative-inverse-under-modulo-m
[3] https://www.geeksforgeeks.org/pollards-rho-algorithm-prime-factorization

Zkusmé dělení iterativně hledá společného dělitele pro prvních 1000000 čísel pomocí funkce modulo. Je to nejjednodušší a nejpomalejší faktorizační metoda s exponenconiální časovou složitostí.

Pokud zkusmé dělení dělitele nenašlo, tak je zavolána Pollardova faktorizační metoda. Tato metoda funguje rychle pro velká složená čísla, kde jedno z prvočísel je výrazně menší. Ačkoliv toto v mém případě neplatí, protože obě prvočísla generuji jako polovinu délky veřejného modulu, tak i přesto je její rychlost přijatelná a v případě klíčů generovaných jiným nástrojem může být ještě efektivnější. Dělitel je možno najít s pravděpodobností okolo 0.5 v O(*sqrt*(*d*) <= O(*n*^(1/4)) iteracích.

Výsledkem je dělitel veřejného modulu a tedy jeden z klíčů, který byl použit při generování. Druhý klíč, který byl při generování použit lze následně snadno dopočítat.

Test prvočísel

Při generování klíčů RSA využívám pro testování prvočísel algoritmus Miller–Rabin[4]. Algoritmus přijímá jako vstupní parametry číslo k otestování a počet iterací algoritmu. Algoritmus následně s určitou pravděpodobností určí, zda číslo je prvočíslo. Platí, že čím více iterací algoritmu, tím větší pravděpodobnost, že odhad bude správný. Dostačující počet iterací je kolem 20.

Závěr

Naimplementoval jsem aplikaci v C++, která odpovídá rozsahu zadání. Testování aplikace jsem prováděl na serveru merlin. Aplikace dokázala velmi rychle vygenerovat i klíče pro veřejný modulus o délce 4096 bitů. Šifrování i dešifrování zpráv fungovalo velmi dobře. Doba faktorizace je velmi proměnlivá, protože algoritmus využívá generátoru náhodných čísel. Provedl jsem několik testů a faktorizace 96 bitového čísla v nejlepším případě zabrala 23 vteřin a v nejhorším případě 2 minuty a 31 vteřin.

[4] https://rosettacode.org/wiki/Miller%E2%80%93Rabin_primality_test