RSA

Bc. Vít Barták (xbarta47)

I. Algoritmus RSA

Algoritmus *RSA* je první asymetrický šifrovací algoritmus, který pro svou funkci využívá faktu, že faktorizace obřích čísel na prvočísla je velmi časově náročná. Algoritmus se dá použít jak pro utajení přenášené zprávy, tak pro elektronický podpis. Princip fungování je následující:

1) Zvolí se dvě náhodná velká prvočísla p a q.
2) n=pq (n je veřejný modulus)
3) $\varphi(n) = (p-1)(q-1)$
4) Zvolí se $e < \varphi(n)$, které je s $\varphi(n)$ nesoudělné.
5) Nalezne se číslo d (privátní exponent): $d * e \equiv 1 (\mathrm{mod}\, \varphi(n))$, kde $\equiv$ značí kongruenci.
6) Je-li e prvočíslo, tak $d = \frac{1+r\varphi(n)}{e}$, kde $r = (e-1)\varphi(n)^{e-2}$

Veřejným klíčem je pak (n,e) a soukromým (n,d). Šifrování (zprávy M) je pak symetrická operace a provádí se následovně: $c \equiv m^e \bmod n$ a dešifrování (kryptogramu C) $m \equiv c^d \bmod n$. Na slabé klíče se dá provést útok pomocí rozložení na prvočísla, avšak v praxi se používají klíče větší než 1024 bitů, pro které je algoritmus považován za bezpečný. [1]

II. Implementace

Přiložený C++ program kry.cpp implementuje šifrování, dešifrování a útok na algoritmus RSA. Neimplementuje však generování klíčů. Pro práci s velkými celými čísly je využita knihovna GMP. Program nejprve čte argumenty z příkazové řádky a následně je parsuje – dle toho jaké přepínače a argumenty se nacházejí na přík. řádce je volána příslušná RSA funkce.

Šifrování a dešifrování (přepínače -e -d) jsou implementovány ve funkci *rsa_encrypt_decrypt(...),* která očekává exponent, modulus a zprávu (plaintext nebo kryptogram). Díky symetrii RSA je to jedna funkce, která podle použití dostane jiný vstup. Po inicializaci pomocných mpz_t tříd (multiprecision integers) pomoci funkce *mpz_powm()* je vypočteno $c|m \equiv m^e|c^d \bmod n$ (podle vstupu , | ‘značí nebo) a výsledek je v hexadecimální soustavě vypsán na STDOUT.

Útok na RSA provádím pomocí dvou algoritmů: naivního dělení a Pollard-Rho faktorizace. Pokud je při spuštění zadán přepínač -b a libovolné číslo v libovolné soustavě (tím by měl být veřejný modulus), pak je zavolána metoda *rsa_break()*.

Po inicializaci potřebných celých čísel nejprve zkouším, jestli není číslo na vstupu 1, pokud ano hned ho vypíšu na výstup a končím. Poté zkouším, jestli nelze dělit číslem 2, pokud ano, vím, že jedno z prvočísel, které ho dělí je 2. Poté pro všechna čísla (kromě sudých) od 3 do milionu zkouším číslo N postupně dělit, pokud naleznu číslo, které ho dělí beze zbytku, vypíši toto číslo a program ukončím.

Pro dělitele větší než milion je z důvodů výkonnosti lepší využít sofistikovanější metodu, proto pro větší čísla N využívám upravený Pollard Rho algoritmus[2]. V případě, že máme opravdu náhodné číslo X dosahuje tento algoritmus úspěchu v polovině času v $O\left(\sqrt{\mathrm{p}}\right) \leq O(n^{1/4})$ iteracích.

Nejprve je pomocí zdroje náhodnosti /dev/urandom inicializován *gmp_randstate_t*. Poté probíhá výpočet následujícího algoritmu (zjednodušený zápis):

pollard_rho:

$$X = random(0, N+1)$$
$$C = random(0, N+1)$$
$$Y = X$$
$$D = 1$$

while(D == 1):

$$f(x, N) = (x^2 \ mod \ N + C) \ mod \ N$$
$$X = f(X, N)$$
$$Y = f\big(f(Y, N)\big)$$
$$D = \gcd(|X - Y|, N)$$
$$if(D == N) goto \ pollard\_rho;$$
$$else \ return \ D;$$

Funkci gcd jsem implementoval jako *e_gcd(X,Y)*, kde největší společný dělitel dvou čísel počítám pomocí Eukleidova algoritmu[3]. Vzhledem k náhodnosti v algoritmu dostávám rozdílně dlouhé časy při faktorizaci. Pokud metoda selže (i v případě, že je N složené), je potřeba změnit polynom, z toho důvodu randomizujeme číslo C.

III. Závěr

Algoritmus RSA byl úspěšně naimplementován za pomocí knihovny GMP (mimo generování klíčů). Statická analýza kódu ani sanitizéry nenalézají při překladu žádné chyby, úniky paměti apod. V průměru trvá faktorizovat 96 bitové číslo okolo 60 sekund, což bylo zjištěno opakovaným testováním (vzhledem k náhodnosti algoritmu je někdy číslo faktorizováno ihned, jindy to trvá okolo devadesáti sekund – testováno na serveru Merlin).

IV. Zdroje

[1] https://cs.wikipedia.org/wiki/RSA

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

[3] https://cs.wikipedia.org/wiki/Eukleid%C5%AFv_algoritmus