2. projekt KRY 2017/2018

Jordán Jarolím, xjarol03

Cílem 2. projektu KRY bylo v první části generovat parametry pro asymetrický šifrovací algoritmus RSA a ve druhé části se pokusit prolomit šifrování dané zprávy faktorizací prvočísel. Na velkých prvočíslech je založen právě celý algoritmus RSA, kdy předpokladem pro úspěšné šifrování a dešifrování tímto algoritmem je právě vygenerování 2 vstupních velkých prvočísel. Délka generovaných prvočísel nesmí být větší než *1/2* požadované délky klíče a současně musí být větší než *2^(k-1/2)*(https://crypto.stackexchange.com/questions/20474/maximum-length-in-bits-of-the-product-n-pq).

Generování RSA parametrů, šifrování a dešifrování

Při implementaci tohoto úkolu byl použit algoritmus SolovayStrassen (https://en.wikipedia.org/wiki/Solovay%E2%80%93Strassen_primality_test). SolovayStrassen funguje na principu modulární ekvivalence, neboli číselné kongruence podle modulo n. Obecně je tedy zkoumáno *k* čísel, kde pokud tato kongruence některé z kandidátů na prvočíslo nebude platit, daný kandidát je takzvaný Eulerův svědek. Nic ovšem nezaručuje, že pokud tato kongruence platí, tak zkoumané číslo musí být skutečně prvočíslem. Je tedy vhodné otestovat více různých čísel přičemž pravděpodobnost s každým otestovaným číslem roste. Při implementaci tohoto projektu byla nastavena hranice na 50 různých čísel pro ověření, že je kandidát skutečně prvočíslem.

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

Na pravé straně kongruence se nachází takzvaný Jacobiho symbol (https://en.wikipedia.org/wiki/Jacobi_symbol), jehož vstupní hodnotou dvě čísla a výstupem je prvek množiny {-1,0,1}. Při implementaci je nejprve vypočítána hodnota levé strany výrazu, poté hodnota pravé strany výrazu, na oba výsledky je poté aplikována operace *modulo N* a hodnoty jsou porovnány.

Po získání 2 pravděpodobných velkých prvočísel je možné generovat další parametry potřebné pro RSA. Dohromady jsou jimi:

- prvočísla *p, q*
- veřejný modulus *n = p*q*
- *phi = (p-1)*(q-1)*
- veřejný exponent *e: e > 1, e < phi, gcd(e, phi) = 1*
- soukromý exponent *d = inv(e, phi(n))*

Pro generování veřejného i soukromého klíče byl použit Euklidův algoritmus (https://en.wikipedia.org/wiki/Euclidean_algorithm), potažmo jeho rozšířená verze (https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm). Základní Euklidův algoritmus slouží především pro nalezení největšího společného dělitele, který je uvažován právě při generování veřejného klíče. Avšak pro výpočet soukromého klíče *d* je nutné najít takzvaný **multiplicative inverse**. Je jím jeden z koeficientů Bezoutovy identity *(x, y),* které se dají získat právě rozšířeným euklidovým algoritmem. Cílem je získat takové *d,* že*:*

$$ed \equiv 1 \pmod{\varphi(n)}$$

Základním tvarem rozšířeného euklidova algoritmu je však:

$$ax + by = \gcd(a, b)$$

Pokud dosadíme *a = e, b = phi* a předpokládáme, že *gcd(e, phi) = 1*, získáváme po dosazení

$$ex + \varphi(n)y = 1$$

Pokud na tento výraz aplikujeme *modulo phi*, získáme:

$$ex \equiv 1 \pmod{\varphi(n)}$$

Z výše uvedeného je evidentní, že *x = d*. Stačí tedy aplikovat rozšířený euklidův algoritmus, získat koeficienty Bezoutovy identity, přičemž právě koeficient *x* nabývá hodnoty hledaného *d* (https://crypto.stackexchange.com/questions/5889/calculating-rsa-private-exponent-when-given-public-exponent-and-the-modulus-fact).

Po získání požadovaných parametrů je možné tyto využit pro šifrování a dešifrování dle principů RSA:

- šifrování: cyphered = m^e mod N
- dešifrování: message = c^d mod N

RSA Cracking

Pro prolomení RSA šifry je možno aplikovat několik přístupů. Základním je tzv. *bruteforce*, který se snaží testovat všechny možná čísla proti vstupnímu N. V implementaci tohoto projektu byla tato část optimalizována tak, že je nejprve otestována dělitelnost 2 a v případě, že je vstupní číslo nedělitelné, poté jsou testovány pouze čísla lichá.

Z pokročilejších algoritmů byl nejprve vyzkoušen algoritmus *Fermat Factorization Method*. Ten se ale bohužel ukázal jako velmi pomalý (alespoň jeho implementovaná verze). Dalším algoritmem je *Pollard's Rho.* Ten byl také implementován do finální podoby a jeho algoritmus je následující:

```
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
```

Je založen na konečném počtu pokusů generátoru pseudonáhodných čísel g(x) (v implementaci je to *x^2 + a mod N*) a využívá **narozeninový paradox**. Namísto generování 1 čísla generuje čísla 2 a tím velmi zvyšuje šanci na správné uhodnutí. Narozeninový paradox je založen na principu toho, že pokud generujeme náhodná čísla z intervalu 1-1000, šance že dostaneme 42 je přesně 0,001. Pokud však vygenerujeme čísla 2, a řekneme, že chceme aby *x-y = 42* pak šance správného výsledku vzroste na cca 0,002. https://www.cs.colorado.edu/~srirams/courses/csci2824-spr14/pollardsRho.html.