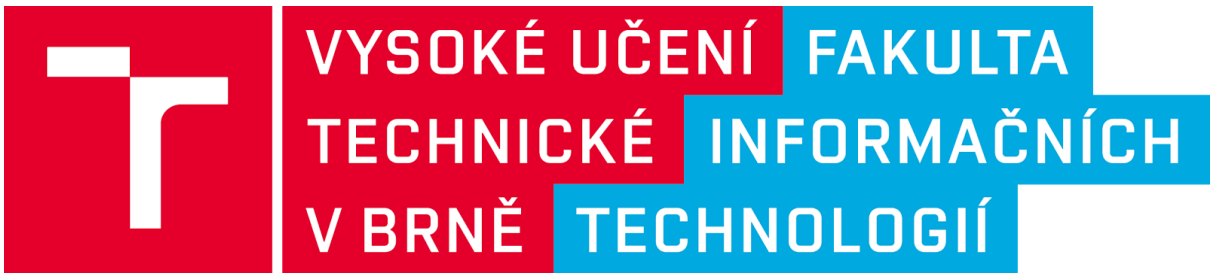

Fakulta informačních technologií
Ústav informačních systémů
Kryptografie

Implementace a prolomení RSA

Projekt č. 2

Tomáš Bruckner, xbruck02

12. dubna 2016

Úvod

Tato práce slouží jako dokumentace implementace mého řešení projektu č. 2 do předmětu Kryptografie. K řešení byl použit jazyk C a knihovna GMP. Pro generování náhodných čísel je použito rovnoměrné rozdělení pomocí algoritmu Mersenne Twister knihovnou GMP. Jako seed byl použit aktuální čas.

Generování

Na začátku generování zjistím, zda je počet bitů modulu sudé nebo liché číslo. Pokud se jedná o sudé číslo, pak vygeneruji dvě prvočísla o délce polovina délky výsledného veřejného modulu. V případě lichého počtu bitů se generuje prvočíslo p o jeden bit větší než v prvním případě. Pokud se vygeneruje prvočíslo q stejné jako prvočíslo p, pak se vygeneruje nové prvočíslo q. V případě stejných prvočísel by se jedno o velmi lehce prolomitelný veřejný modulus (např. pomocí Frematovy metody).

Jakmile se vygenerují obě prvočísla, tak se navzájem vynásobí a zjistíme, jestli výsledný součin má na požadovaném bitu jedničku. Pokud ne, tak se generují nové dvě prvočísla. Jakmile máme požadovaná prvočísla, tak se spočítá phi_n, který se použije jako modulus při generování privátního klíče. Jako veřejný exponent jsem použil pět Fermatových prvočísel. Pokud je multiplikativní doplněk prvního veřejného exponentu různý od jedničky, pak jsme vygenerovali privátní klíč. V opačném případě se vyzkouší další veřejný exponent.

Funkci pro multiplikativní doplněk jsem si musel implementovat sám, i když je podporována knihovnou GMP. Použil jsem rozšířený Euklidův algoritmus, který jsem převzal z Public-key Cryptography od Jamese Nechvatala[1].

Nejsložitější část bylo generování prvočísel. Nejdříve vygeneruji náhodné číslo v intervalu od 0 do 2^bit_length – 1. Následně nastavím nejvyšší požadovaný bit na jedničku. Pokud se jedná o sudé číslo, pak se přičte jednička.

Následně probíhá testování na čísla na prvočíslo. Nejdříve se provede Fermatův test prvočselnosti[2]. Pokud dané číslo splňuje Fermatův test, pak se provede sto iterací Millerova-Rabinova testu prvočíselnosti. Jedná se o pravděpodobností test, který má šanci na chybu 1/4[3]. Sto iterací tedy zajišťuje dostatečně malou pravděpodobnost 2^(-200) omylu. Fermatův test je silnější než Miller-Rabin, a tedy každé číslo, které splní Fematův test by mělo splnit i test Millera-Rabina. Nicméně Fermatův test špatně určuje Carmichaelova čísla, proto je zde použit jenom jako rychlé ověření[4].

Pokud zkoumané číslo nesplní Fermatův test nebo některou z iterací Millerova-Rabinova testu, pak se k danému číslu přičte dvojka a opět se zkoumá, zda se nejedná o prvočíslo.

Šifrování a dešifrování

Šifrování a dešifrování je implementováno triviálně GMP funkcí mpz_powm, která řeší celý problém.

Faktorizace

Nejdříve se ověří, zda zadaný modulus není dělitelný dvěma, což vede k řešení 2. Následně se provede triviální dělení pro prvních milión čísel. Pokud některé číslo dělí zadaný modulus beze zbytku, pak byl

[1] Dostupné z: https://www.fit.vutbr.cz/study/courses/KRY/private/800-2.txt
[2] Dostupné z: http://mathcircle.berkeley.edu/BMC5/docpspdf/is-prime.pdf
[3] Dostupné z: http://mathworld.wolfram.com/Rabin-MillerStrongPseudoprimeTest.html
[4] Dostupné z: http://mathcircle.berkeley.edu/BMC5/docpspdf/is-prime.pdf

nalezen hledaný dělitel. Tato metoda je implementována pomocí for cyklu a operace modulo. Testují se pouze liché čísla, jelikož jsme si hned na začátku ověřili, že se nejedná o sudé číslo.

V případě neúspěchu triviálního dělení se použije Brentova varianta metody Pollardova rho[5]. Při výběru metody jsem se zaměřil především na rychlost faktorizace. Pollardova rho metoda je obecně výrazně rychlejší, než Fermatova metoda či Pollardova p – 1 metoda i přesto, že všechny spadají do exponenciální třídy složitosti[6]. Původně jsem zvažoval metodu Quadratic Sieve, ale byla zbytečně komplikovaná na implementaci. Také jsem nejdříve implementoval Pollardovu metodu, která využívá Floydův algoritmus hledání kruhů, která byla ovšem příliš pomalá a nestíhala faktorizovat 96 bitová čísla do 30s na serveru merlin.fit.vutbr.cz, což bylo v zadání požadováno. Brentova varianta využívá Brentův aloritmus pro hledání cyklů[7], což zrychlilo faktorizaci desetkrát až dvacetkrát.

Samotná metoda Pollardova rho využívá funkci nejvyššího společného dělitele, která je sice podporována přímo v GMP, ale dle zadání nebylo možné tuto funkci použít a musel jsem ji implementovat sám pomocí Euklidova algoritmu[8]. Rozhodl jsem se nevyužít žádnou rekurzivní variantu, jelikož mi přišlo, že by režie spojená s voláním funkce zbytečně výpočet zpomalila. Algoritmus je implementován pouze pomocí GMP funkce modulo. Při implementaci jsem narazil na problém, že jsem si operátory přepisoval, což vedlo k nesprávnému výsledku. Musel jsem si tedy vytvořit kopie operátorů, které jsem si následně mohl libovolně přepisovat.

Závěr

Zadání bylo splněno, nicméně největší problémy jsou s faktorizací na serveru merlin.fit.vutbr.cz zvláště nyní, když začínají projekty do předmětu IOS, kdy bývá server přetížen, jelikož si někteří prváci pustí 60 procesů, které se jim navíc i třeba zacyklí. Rychlost metody Pollardova rho je založena na tom, jak vhodné čísla se vygenerují. Nicméně dle aktuální zatíženosti serveru se mi pro stejné číslo rychlost faktorizace lišila víc jak desetinásobně.

[5] Dostupné z: https://comeoncodeon.wordpress.com/2010/09/18/pollard-rho-brent-integer-factorization
[6] Büttcher, S. *Factorization of Large Integers*.Ferienakademie 2001: Cryptography and Security of Open Systems. Dostupné z: http://www.stefan.buettcher.org/cs/factorization/essay.pdf
[7] Dostupné z: http://mathworld.wolfram.com/BrentsFactorizationMethod.html
[8] Dostupné z: http://blog.janmr.com/2009/10/computing-the-greatest-common-divisor.html