Faktorizace velkých čísel

V červnovém čísle našeho časopisu jsme v článku “RSA v ohrožení” krátce informovali o novém projektu prof. Shamira, který by mohl výrazně ohrozit současnou asymetrickou kryptografii. Pojďme se nyní blíže podívat na zařízení jménem TWINKLE, které by mělo být výsledkem tohoto projektu, a na jeho aplikaci v oblasti kryptoanalýzy.

Na to vezmi LED!

Jedná se elektrooptické zařízení, které by umožnilo výrazně zrychlit současné faktorizační metody a tím posunout pomyslnou hranici (zatím na 465 b) mezi “snadno a těžko” faktorizovatelnými čísly zhruba o 100 až 200 bitů výš. Pro kryptosystém RSA, pro nějž tato hranice přímo určuje bezpečnou délku modulu, by to znamenalo akutní ohrožení implementací využívajících 512b modul. Toto je shodou okolností též maximální délka modulu, kterou NSA povolila pro export z USA. Dá se proto předpokládat, že tyto systémy budou v běžné komerční oblasti hojně rozšířeny. Na druhou stranu ale většina solidních systémů RSA používá modul o minimální velikosti 1024 b, který se zatím nezdá být konstrukcí zmíněného zařízení ohrožen. Tolik tedy na hrubý úvod. Více obecných informací o celém projektu můžete získat nahlédnutím do výše zmíněného článku.

Kdo je v ohrožení

Jak si ukážeme později, představuje zařízení TWINKLE de facto masivní paralelní prosévací stroj, jehož uplatnění v oblasti diskrétní matematiky zdaleka nespočívá jen v asistenci při řešení problému faktorizace. Nicméně faktorizační problém byl pro svou atraktivitu vybrán jako demonstrační vzor pro jeho předvedení. Pokud by zůstalo při využití TWINKLU ryze pro faktorizační účely, potom by byla ze známých asymetrických šifer výrazně ohrožena snad jen RSA. Už teď je ale zřejmé, že TWINKLE by mohl stejně dobře posloužit pro řešení problému diskrétního logaritmu, na kterém je vystaven například rozšířený podpisový standard DSS. I ten by tedy mohl být existencí tohoto zařízení ohrožen, avšak díky charakteru problému diskrétního logaritmu již zdaleka ne tak vážně jako RSA. Proč tomu tak je, to si ukážeme v příštím díle tohoto seriálu, který bude celý věnován řešení diskrétního logaritmu na multiplikativní grupě Zp.

Dnes se budeme zabývat výhradně použitím TWINKLU pro řešení problému faktorizace. Abychom lépe pochopili, proč je vlastně nalezení způsobu rychlého řešení tohoto problému noční můrou všech systémů na bázi RSA, zopakujeme si nejprve v krátkosti způsob, jakým RSA vlastně pracuje. Formálně můžeme RSA popsat jako trojici (Kx, Ky, n), kde Kx je veřejný klíč, Ky je jemu odpovídající klíč tajný a číslo n je modul určující multiplikativní grupu Zn. Hodnoty Kx a n jsou veřejné, Ky je tajný.

Dále platí, že n = p*q, kde p a q jsou prvočísla. Vztah mezi Kx a Ky je definován kongruencí $Kx*Ky \equiv 1 \pmod{\Phi(n)}$, kde $\Phi(n)$ je Eulerova funkce, která je v tomto případě definována jako $\Phi(n) = (p-1)*(q-1)$. Zde vidíme, že pokud by případný útočník chtěl z našeho veřejného klíče Kx získat tajný

klíč Ky, musel by umět spočítat hodnotu $\Phi(n)$, k čemuž potřebuje znát původní prvočísla p a q. Ta jsou samozřejmě tajná a leckdy nejsou po vygenerování dané instance RSA ani nijak dále archivována. Proto pro případného útočníka existuje jediný způsob, jak zjistit hodnotu tajného klíče Ky, který spočívá ve faktorizaci veřejného čísla n na součin prvočísel p a q. Při jejich znalosti je potom už výpočet Ky jako $Ky \equiv Kx^{-1} \pmod{(p-1)*(q-1)}$ jen technickou záležitostí.

Další informace o asymetrickém systému RSA můžete nalézt například v [VKLIMA95]. Pro naše účely nám zde postačuje, že jsme ukázali souvislost mezi napadením RSA a řešením problému faktorizace.

Problém faktorizace

Obecný problém faktorizace nějakého celého kladného čísla n spočívá v tom, že se snažíme nalézt jeho zápis ve tvaru $n = \Pi_{(i)} p_i^{e_i}$, kde p_i je i-té prvočíslo tohoto rozkladu a e_i je jeho exponent, $e_i \geq 1$. Tuto obecnou formulaci můžeme pro případ RSA, u něhož víme, že modul n je složen právě ze dvou prvočísel (p_1 a p_2), kde každé z nich navíc vystupuje v první mocnině ($e_1=e_2=1$), zjednodušit takto: mějme celé kladné číslo n, kde $n=p_1 * p_2$, kde p_1 a p_2 jsou prvočísla. Úkolem faktorizace je najít konkrétní prvočísla p_1 a p_2, pro která tento vztah platí. V souladu se zavedenou terminologií v popisu RSA budeme dále číslo p_1 značit jako p a p_2 jako q.

Existuje řada způsobů, jak můžeme problém faktorizace řešit [MENEZES96]. Společnou charakteristikou všech těchto metod je, že s rostoucí velikostí čísla n začíná jejich účinnost od jisté hranice velmi rychle klesat. Tato hranice potom určuje minimální doporučenou délku modulu pro RSA.

V následujícím textu se budeme věnovat pouze jedné z těchto metod, která se nazývá Quadratic Sieve, zkráceně QS. Důvod, proč si popíšeme právě QS, spočívá v tom, že je to právě ta metoda, kterou může existence TWINKLU výrazně urychlit. Poznamenejme, že pro účely praktické realizace nějakého útoku by byl TWINKLE zřejmě nakonec propojen s metodou NFS (nebo alespoň s nějakým derivátem QS), která byla použita pro zatím “nejdelší” faktorizaci 465bitového čísla. Vzhledem k tomu, že NFS vychází ideově ze stejných základů jako QS, která je navíc o poznání jednodušší pro výklad, budeme se dále věnovat právě QS. Ostatně i profesor Shamir si pro první přiblížení funkce TWINKLE ve svém dokumentu [SHAMIR99] vybral kvůli přehlednosti právě QS.

Algoritmus QS

Ještě před vlastním výkladem bych rád předeslal, že dále uvedená tvrzení nebudeme z důvodu přehlednosti a čtivosti celého textu doprovázet příslušnými důkazy. Kdo by měl o tyto důkazy zájem, tomu doporučuji použít jako výchozí bod publikaci [MENEZES96], kde jsou uvedeny odkazy na konkrétní ryze teoretické prameny.

Základní myšlenka pro nalezení rozkladu nějakého čísla n vychází u tohoto algoritmu z následujícího pozorování: pokud známe nějaká čísla r a s taková, že $n \mid rs$ a zároveň n nedělí ani r, ani s, potom $\gcd(r, n)$ je netriviálním faktorem čísla n. Vzhledem k našemu zjednodušení problému faktorizace pro $n = p*q$ můžeme rovnou psát, že $p = \gcd(r, n)$. Faktor q potom určíme už jednoduchou operací dělení: $q = n/p$. Tytéž vztahy je možné analogicky založit též na čísle s.

Jedním z elegantních způsobů, jak zmíněná čísla r a s najít, je na Zn nalézt netriviální řešení kvadratické kongruence $x^2 \equiv y^2 \pmod n$, tedy takové, že $x \neq \pm y \pmod n$. V takovém případě totiž platí, že n | (x-y)*(x+y) a zároveň n nedělí ani (x-y), ani (x+y). Na základě předchozího pozorování proto můžeme psát, že p = gcd((x-y), n).

Tímto úhybným manévrem jsme se však problému tak úplně nezbavili, neboť nalézt řešení uvedené kongruence rovněž není zrovna jednoduchou záležitostí. Podívejme se, jaká je základní filozofie jeho hledání. Číslo, které budeme faktorizovat, označíme jako n.

Začneme tím, že vybereme prvních t prvočísel a vytvoříme z nich množinu $S = \{ p_1, p_2, ..., p_t \}$. O nenulovém celém číslu x prohlásíme, že je p_t-smooth právě tehdy, když je možné jej kompletně faktorizovat na součin (včetně případných mocnin) prvočísel z množiny S. Jak uvidíme dále, je generování čísel, která jsou p_t-smooth, jednou z klíčových částí celého algoritmu.

Dalším krokem algoritmu je generování párů čísel (a_i, b_i) takových, že $a_i^2 \equiv b_i \pmod n$ a b_i je p_t-smooth. Prakticky to celé vypadá tak, že postupně generujeme čísla a_i, počítáme jejich druhé mocniny a testujeme, je-li dané číslo b_i p_t-smooth, či nikoliv. Pokud ano, uložíme si pár (a_i, b_i) do paměti, pokud ne, zrušíme jej.

Předchozí krok opakujeme tak dlouho, dokud nemáme alespoň t+1 párů čísel (a_i, b_i). Proč zrovna t+1, to záhy objasníme. Nyní se zaměřme na hlavní myšlenku celého postupu. Zaveďme si množinu I, která bude obsahovat hodnoty všech indexů i vygenerovaných párů (a_i, b_i), tedy $I = \{ 1, 2, 3, ..., t+1 \}$. Naším cílem teď bude nalézt její podmnožinu $T \subseteq I$ takovou, že součin všech čísel b_i s indexem $i \in T$ je modulo n kongruentní s druhou mocninou nějakého celého čísla c. Formálně to můžeme zapsat takto: $\Pi_{(i \in T)} b_i \equiv c^2 \pmod n$, $c \in Z$. Jakmile takovou podmnožinu T nalezneme, tak máme, dá se říci, vyhráno, neboť kromě výše uvedeného vztahu dále platí, že $\Pi_{(i \in T)} b_i \equiv (\Pi_{(i \in T)} a_i)^2 \pmod n$. Tato kongruence platí díky způsobu, jakým byly dvojice (a_i, b_i) konstruovány. Složíme-li teď střípky celé mozaiky dohromady, dostaneme, že $(\Pi_{(i \in T)} a_i)^2 \equiv c^2 \pmod n$, odkud již vidíme, že řešení výše uvedené kvadratické kongruence obdržíme jednoduchým dosazením: $x = \Pi_{(i \in T)} a_i$, y = c. Poznamenejme, že uvedený postup nezaručuje, že získané řešení nebude triviální. V takovém případě nám nezbývá nic jiného než se pokusit nalézt jinou podmnožinu T, která nás dovede k vytouženému netriviálnímu řešení. Někdy se nám může dokonce stát i to, že budeme muset některé dvojice (a_i, b_i) obměnit, avšak podle praktických zkušeností tato zvlášť smolná situace nenastává příliš často [MENEZES96].

Nyní, když už známe hlavní linii celého algoritmu, se můžeme zabývat některými vybranými detaily. Nejprve si ukážeme, proč jsme v předchozím odstavci uvedli, že budeme potřebovat alespoň t+1 párů (a_i, b_i). Před vlastním výkladem si připomeňme, že t je počet prvočísel v množině S. Dále víme, že ze všech vygenerovaných párů (a_i, b_i) jsme si nakonec svědomitě ponechávali pouze ty, u kterých byla čísla b_i p_t-smooth. To znamená, že pro každé takové b_i známe jeho zápis ve tvaru součinu prvočísel z množiny S, $b_i = \Pi_{j=1}^{t} p_j^{e_{ij}}$, $p_j \in S$, $e_{ij} \geq 0$. Všimněme si, že na rozdíl od výše uvedené definice faktorizace nějakého čísla jsme zde "dovolili", aby exponenty e_{ij} nabývaly nulových hodnot, a součin provádíme implicitně přes všechna prvočísla z množiny S. Důsledkem je, že každé číslo b_i můžeme popsat příslušným vektorem exponentů $E_i = (e_{i1}, e_{i2}, ..., e_{it})$.

Podívejme se nyní na způsob, jakým je možné nalézt výše uvedenou podmnožinu T. Víme, že součin čísel b_i, $i \in T$, musí být modulo n roven druhé mocnině nějakého čísla c. Toho můžeme dosáhnout tak, že každý exponent e_j součinu $\Pi_{(i \in T)} b_i$ bude sudé číslo. Poznamenejme, že pro vektor

exponentů E popisující součin $\Pi_{(i \in T)}\ b_i$ platí $E = \Sigma_{(i \in T)}\ E_i$, kde E_i jsou vektory exponentů příslušných rozkladů čísel b_i nad S.

Vzhledem k tomu, že nás zajímá pouze to, je-li vektor E složen ze sudých souřadnic, či nikoliv, můžeme ke každému vektoru E_i přiřadit paritní vektor $V_i = (v_{i1}, v_{i2}, ..., v_{it})$, kde $v_{ij} = e_{ij} \bmod 2$. S použitím vektorů V_i můžeme hledání podmnožiny T přeformulovat na úlohu hledání podmnožiny lineárně závislých vektorů takto: $\Sigma_{(i \in T)}\ V_i \equiv 0\ (\bmod 2)$. Tento problém již umíme řešit pomocí standardních algebraických operací nad Z_2.

Abychom si zaručili, že hledaná podmnožina lineárně závislých vektorů skutečně existuje, použijeme známé tvrzení, které říká, že máme-li t-rozměrný vektor $A = (a_1, a_2, ..., a_t)$, potom na množině o t+1 vektorech $\{ A_1, A_2, ..., A_{t+1} \}$ existuje jejich netriviální lineární závislost. Tolik ke slíbenému objasnění, proč hledáme alespoň t+1 párů čísel (a_i, b_i).

Poslední věcí, kterou nám zbývá uvést, je způsob generování párů (a_i, b_i). Pro tento účel algoritmus QS definuje polynom $Q(x) = (x + m)^2 - n$, kde $m = \lfloor \sqrt{n} \rfloor$.

Pro (a_i, b_i) potom platí, že $a_i = (x_i + m)$, $b_i = Q(x_i)$. Čísla x_i jsou přitom volena z posloupnosti: 0, ±1, ±2, ±3 atd. Tento způsob generování (a_i, b_i) má výhodu zejména v tom, že b_i se "drží" na relativně nízkých hodnotách a že je možné díky charakteru těchto čísel z množiny S vypustit některá "nepotřebná" prvočísla (je však třeba doplnit prvočíslo –1, protože Q(x) může nabývat záporných hodnot), více viz [MENEZES96].

Úskalí algoritmu QS

Podívejme se nyní na hlavní části popsaného algoritmu z pohledu rychlosti. Zde můžeme prohlásit, že hlavními kroky jsou generování párů čísel (a_i, b_i), kde jde především o rychlost, a potom hledání popsané lineární závislosti vektorů nad Z_2, kde jde dílem o rychlost a dílem o paměťový prostor. Abychom si udělali lepší představu o náročnosti jednotlivých kroků, můžeme se podívat na připojenou tabulku, která byla uveřejněna v materiálu [RSA99]. Zde vidíme náročnost zmíněných operací pro metodu NFS (odhady pro QS bohužel nebyly k dispozici) v závislosti na velikosti čísla n (délce modulu RSA).

Sloupec udávající velikost množiny S a prosévacího pole (viz dále) je určujícím faktorem pro rychlost první části algoritmu. Velikost matice pro řešení lineární závislosti zase udává, je-li tento problém vzhledem k paměťové kapacitě současných počítačů vůbec řešitelný.

Nabízí se logická otázka, zdali by nešlo některou z částí algoritmu QS urychlit. Odpovědí může být právě konstrukce zařízení TWINKLE, které umožňuje zhruba 500- až 1000krát zrychlit první fázi, tedy generování čísel (a_i, b_i).

Metoda síta

Abychom lépe pochopili způsob činnosti TWINKLU, podíváme se nejprve obecně na takzvanou prosévací metodu, která se normálně používá během procesu generování čísel (a_i, b_i).

Jedná se zde především o to, jak rychle rozhodnout, je-li dané číslo b_i p_t-smooth, či nikoliv. Nebo ještě lépe, jak rovnou generovat jenom taková b_i, která tuto vlastnost splňují. Standardním postupem "kanadských dřevorubců" by zřejmě bylo každou nově vygenerovanou hodnotu b_i zkoušet postupným dělením faktorizovat na součin čísel z množiny S a sledovat, zda je tento pokus úspěšný, či nikoliv. Tato metoda se však nezdá být zrovna optimální.

Velmi elegantní řešení celého problému se nám nabízí, pokud si uvědomíme, že dělitelnost čísla b_i nějakým prvočíslem p určuje kongruenci $b_i \equiv 0 \pmod p$. Přepíšeme-li nyní tento vztah s vyžitím polynomu Q(x), dostáváme kvadratickou kongruenci $(x + m)^2 \equiv n \pmod p$. Jejím řešením obdržíme kořeny r_1 a r_2. S jejich využitím nyní můžeme tvrdit, že prvočíslo p dělí b_i právě tehdy, když platí, že $b_i = Q(r_1 + k*p)$ nebo $b_i = Q(r_2 + k*p)$.

Metoda síta spočívá v tom, že v paměti vytvoříme pole A pokrývající rozsah zkoušených hodnot x_i. Všechny prvky pole inicializujeme nulovou hodnotou. Do každé položky pole potom pro každé prvočíslo $p \in S$ přičteme hodnotu log(p) právě tehdy, když daná položka odpovídá hodnotě x_i, která patří do množiny řešení výše uvedené kongruence (tj. $x_i = r_1 + k*p$, nebo $x_i = r_2 + k*p$). Takto vytvořené pole potom postupně procházíme a ty položky, pro které platí $A[x_i] \cong \log$

($Q(x_i)$), prohlásíme za kandidáty na p_t-smooth čísla, přičemž tuto domněnku potom ještě ověříme "kanadskou metodou" postupného dělení (zde si to již můžeme dovolit). Poznamenejme, že při práci s polem A jsme dovedně využili faktu známého už z éry logaritmických pravítek, který praví, že operace logaritmu umožňuje snadno převádět operaci násobení na sčítání.

V praxi se ukazuje, že metoda síta je i přesto, že například nepřipouští, aby dané prvočíslo bylo v rozkladu b_i zastoupeno ve vyšší mocnině než jedna, velmi efektivním nástrojem pro vyhledávání p_t-smooth čísel. Stále je tu však nutnost častého přístupu k rozsáhlému paměťovému prostoru, jehož rychlost má své hranice. Podíváme-li se na celou metodu pozorněji, zjistíme, že je doslova jako dělaná pro paralelní implementaci – a to je právě ta cesta, kterou se ubírá projekt TWINKLE.

Jak prosévá TWINKLE

Podívejme se nejprve na obrázek, kde je v hrubých rysech znázorněn svislý řez zařízením TWINKLE. V dolní části se nachází matice LED. Každá dioda zde odpovídá jednomu prvočíslu p_j z množiny S a je napojena na řídicí jednotku, která ji rozsvěcí právě v těch časových okamžicích t_i, pro které $p_j \mid Q(x_i)$. Toto řízení je odvozeno od vztahů popsaných výše.

Nad maticí LED je umístěn filtr, jehož propustnost je pro každou diodu jiná (rastr filtru odpovídá rastru matice LED) a je volena tak, aby výsledná intenzita procházejícího paprsku odpovídala hodnotě $\log(p_j)$. Procházející paprsky jsou dále pomocí spojné čočky soustředěny do jejího ohniska, kde je umístěn fotodetektor. Ten v jednotlivých okamžicích t_i vyhodnocuje výslednou intenzitu dopadajícího záření, které v čase t_i odpovídá hodnotě $\Sigma_{(j \in Li)} \log(p_j)$, kde $L_i = \{p: p \in S, p \mid Q(x_i)\}$.

Posledním klíčovým bodem celého zařízení je komparátor, který porovnává napětí získané na fotodetektoru s hodnotou odpovídající $\log (Q(x_i))$. Pokud se tyto hodnoty rovnají, potom je velmi pravděpodobné, že číslo $b_i = \log (Q(x_i))$ je p_t-smooth. Vzhledem k možným nepřesnostem je však třeba

ještě tuto hypotézu ověřit metodou postupného dělení na připojeném počítači, který na to má opět dostatek času.

Z uvedeného vyplývá, že TWINKLE představuje rychlý paralelní nástroj, který je schopen v jediném taktu t_i otestovat, je-li odpovídající číslo $Q(x_i)$ p_t-smooth, či nikoliv. Podívejme se, co tato vlastnost znamená pro jeho praktické použití. Předpokládá se [SHAMIR99], že matice LED bude obsahovat 200 000 diod neboli že pokryje množinu S o 200 000 prvočíslech. Dále se předpokládá taktovací frekvence 10 GHz (při použití GaAs technologie si to můžeme dovolit) a řídicí logika, která je schopna pracovat nad sítem pro 100 000 000 čísel. Prosévání tohoto intervalu pak bude trvat 0,01 s, přičemž stejná operace na PC by trvala 5 až 10 sekund. Odtud vidíme, že zařízení umožňuje opravdu 500- až 1000krát zrychlit první část algoritmu QS.

Poznamenejme, že výše uvedené parametry ohledně velikosti faktorizační báze S a prosévaného intervalu jednoho TWINKLE jsou prakticky pevné. Zvýšení těchto hodnot, které by pro konkrétní nasazení bylo nevyhnutelné (viz tabulka), by se provedlo paralelním spojením více jednotek. Pro 512bitový modul RSA se počítá s použitím 15 až 20 těchto zařízení [RSA99].

Stále není vyhráno

V dnešním článku jsme si ukázali, jak vypadá algoritmus QS a jakým způsobem je možné jej zrychlit pomocí zařízení TWINKLE. Ukázali jsme si, že toto zařízení může výrazně urychlit první část QS. Zároveň jsme však poznali, že tímto zrychlením ještě zdaleka není vše vyřešeno, neboť je tu ještě druhá část QS, která spočívá v řešení soustavy rovnic nad Z_2 a kterou už TWINKLE nijak nezrychluje. Velikost této soustavy přitom s rostoucí délkou modulu začíná být prakticky neúnosná. Zrychlení první části QS proto od jistého okamžiku není nic platné, a to ani za předpokladu, že bychom její trvání stáhli na pouhý jeden takt!

Střízlivým odhadem proto můžeme vznést domněnku, že existence TWINKLU by představovala akutní hrozbu hlavně pro RSA moduly délky 512 b, přičemž v současnosti používaných 1024 b zůstává daleko za obzorem jeho možností. Tento závěr však není dobré ani přecenit, ani podcenit. Zkrátka, jak praví klasik: Já neříkám tak ani tak, ale na má slova dojde...

Tomáš Rosa (tomas.rosa@decros.cz)

Literatura

[MENEZES96] – Menezes, A. J., van Oorschot, P. C., Vanstone, S. A.: Handbook of applied cryptography, CRC Press, 1996.

[RSA99] – internetový dokument *http://www.rsa.com/rsalabs/htm/twinkle.html*.

[SHAMIR99] – přednáška prof. Shamira – *ftp://ftp.decros.cz/support/pub/shamir.pdf*

[VKLIMA95] – Klíma, V.: Šifrový šampion, CHIP 4/95, str. 136 – 138.