An Overview of the Attack Methods Directed Against the RSA Algorithm

Ing. Cristian MARINESCU, prof.dr.ing. Nicolae ŢĂPUŞ
Facultatea de Automatică şi Calculatoare, Universitatea Politehnica Bucureşti

The necessity of developing secure communication methods over the Internet has become lately a very important issue. More and more cryptographic modules are deployed every year, but choosing the "best" algorithm for the application is not a trivial task. This seems to be one of the general problems of cryptography today: "which algorithm is the best (most secure) algorithm"? There is no single answer to this question; different applications require different levels of security, forcing different requirements on the software. This is why a detailed analysis of the different algorithms, their vulnerabilities and methods of attack is very important.
The article presents one of the most commonly used algorithms in computer security (RSA), and analyzes all known methods of attack against it. Although there are many known methods of attack, the RSA algorithm has never been compromised (until now). The elegance and the apparent simplicity are notable features of the RSA algorithm. However, implementation of a secure RSA version remains an extremely difficult problem. Improper use of the algorithm can be very dangerous, threatening its security. This is why, from a software developer's perspective, a robust and correct implementation of the algorithm that takes all the potential pitfalls into consideration, is very important.
Keywords*: cryptography, security, RSA.*

Introducere

Algoritmul RSA a fost publicat pentru prima oară în 1977 de R. Rivest, A. Shamir şi L. Adleman în revista "Scientific American" şi se compune din calcule numerice în inelul Z_n. Sistemele de tipul RSA fac parte din categoria sistemelor criptografice cu cheie publică. Securitatea algoritmului se bazează pe problema factorizării numerelor foarte mari. Algoritmul poate fi utilizat atât pentru criptare, cât şi pentru autentificare (semnături digitale) şi este foarte larg răspândit la ora actuală. El este întâlnit în servere şi browsere de web, în clienţi şi servere de e-mail, reprezentând practic coloana vertebrală a sistemului de plăţi electronice prin card-uri de credit.
Algoritmul funcţionează după cum urmează:

- Se generează două numere prime p şi q, de lungime $\frac{n}{2}$ biţi (de exemplu 512, 1024, 2048 biţi lungime). Deoarece mulţimea numerelor prime este suficient de densă, numerele prime pot fi generate alegând aleator numere întregi de $\frac{n}{2}$ biţi şi testându-le cu ajutorul unui test probabilistic. Apoi, fie $N=p*q$, de lungime n biţi.
- Numărul e trebuie ales astfel încât să îndeplinească următoarele condiţii: $1 < e < N$ iar e şi (p - 1) * (q - 1) să fie relativ prime, sau altfel spus, să nu aibă factori primi în comun.
- Se calculează d cu ajutorul algoritmului euclidian extins, astfel încât acesta să fie multiplul invers al lui e sau altfel spus d * e - 1 să fie divizibil cu (p-1) * (q-1). În practică, d se poate obţine foarte simplu căutând rezolvarea ecuaţiei $d = \frac{x*(p-1)*(q-1)+1}{e}$ astfel încât d şi x să fie numere întregi. Valorile d şi e sunt numite exponentul privat, respectiv exponentul public al algoritmului.
- Funcţia de criptare/semnare arată astfel: $C = M^e \bmod N$, unde M reprezintă mesajul de criptat (un întreg pozitiv mai mic decât N).
- Funcţia de decriptare/verificare arată astfel: $M = C^d \bmod N$, unde C reprezintă textul criptat. Cheia publică este reprezentată de perechea *(N, e)*, iar cheia privată de perechea (*N, d)*. Numărul *d* mai este cunoscut şi sub

numele de "trap door", deoarece cunoaşterea sa permite inversarea rapidă a funcţiei RSA.

Viteza algoritmului RSA depinde în mare măsură de lungimea cheilor utilizate, de tipul de implementare, de procesorul pe care se rulează aplicaţia, dar şi de protocolul ce trebuie implementat. Deseori, pentru a obţine o viteză sporită în aplicaţiile practice, sunt utilizaţi exponenţi publici mici, acest fapt implicând însă şi riscuri corespunzătoare. Există chiar grupuri întregi de utilizatori care folosesc acelaşi exponent public, doar modulul N fiind diferit. În acest caz există însă reguli stricte ce trebuiesc respectate pentru cele două numere prime p şi q, astfel încât siguranţa algoritmului să nu fie periclitată. Utilizând, cum spuneam, exponenţi publici mici, se obţine o viteză mai mare de criptare şi verificare în comparaţie cu procesele inverse de decriptare şi semnare a datelor. Utilizând algoritmii generali de calcul ai exponenţialului, operaţiile cu cheie publică consumă un timp proporţional cu $O(n^2)$, iar operaţiile cu cheie privată necesită aproximativ $O(n^3)$, unde n reprezintă numărul de biţi ai lui N. Tehnicile de multiplicare rapidă, necesită de obicei mai puţini paşi, sunt însă destul de rar folosite datorită complexităţii lor, şi a faptului că pentru lungimi tipice de chei, ele sunt totuşi mai lente.

Dacă comparăm viteza algoritmului RSA cu cea a unui algoritm cu cheie simetrică (DES de exemplu), putem observa că în funcţie de implementare (HW sau SW) cel din urmă este cu până la aproximativ 1000 de ori mai rapid decât RSA. Cu toate acestea, utilizarea RSA în algoritmi de distribuire de chei (simetrice) sau în alte aplicaţii, în care viteza este mai puţin importantă, prezintă avantaje de netăgăduit.

Securitatea sistemelor RSA se bazează pe presupunerea că funcţia $C = M^e \bmod N$ este unidirecţională, fiind computaţional dificil de a se găsi mesajul iniţial M în absenţa exponentului de decriptare d. Există însă posibilitatea, cel puţin teoretică, de a încerca factorizarea lui N prin metoda forţei brute sau prin alte metode, fapt ce ar duce la aflarea numerelor p şi q. Apoi utilizând algoritmul euclidian extins se poate calcula exponentul de decriptare d, ceea ce ar duce la compromiterea cheii private şi la descifrarea textului criptat.

Încă de la publicarea sa, algoritmul RSA a fost studiat de o mulţime de cercetători, fiind supus la nenumărate teste. Cu toate că de-a lungul celor mai bine de 25 de ani de utilizare au rezultat diverse vulnerabilităţi, algoritmul s-a dovedit suficient de rezistent (până în prezent) pentru a putea oferi un grad ridicat de securitate. Metodele de atac rezultate nu fac decât să ilustreze încă o dată pericolul utilizării RSA în condiţii necorespunzătoare, programarea unei versiuni sigure de RSA nefiind deloc o problemă simplă.

Metode de analiză şi atac. O clasificare generală a acestora.

Dificultatea de a inversa funcţia RSA pentru mesaje aleatoare duce la concluzia că, dat fiind tuplul (N, e, C), un atacator nu va putea obţine mesajul criptat M. Cu toate acestea, se pot obţine diverse informaţii despre M, astfel încât RSA nu este un sistem sigur din punct de vedere semantic (date fiind N, e, C, se pot obţine relativ uşor proprietăţi ale lui M - de exemplu simbolul Jacobi al lui M peste N). RSA poate fi făcut însă sigur din punct de vedere semantic dacă se adaugă informaţie aleatoare în procesul de criptare [1].

O clasificare a metodelor de atac este relativ dificil de făcut, cu toate acestea se poate aborda problema din două direcţii. În primul rând se poate încerca clasificarea metodelor de atac după impactul pe care îl au asupra algoritmului. Astfel, se pot identifica două categorii de atacuri:

- atacuri ce compromit cheia privată, compromiţând practic întreaga securitate a sistemului;
- atacuri ce compromit un singur mesaj criptat, cheia privată putând fi însă utilizată în continuare.

Din prima categorie putem enumera metoda forţei brute, atacul exponentului privat mic etc. Atacurile din cea de a doua categorie sunt în general mult mai subtile, folosind metode relativ complicate de aflare a mesajului. Cu toate acestea, timpul utilizat pentru aflarea unui mesaj trebuie să fie mult mai mic decât timpul necesar metodei forţei brute,

pentru ca atacul să aibă o anumită relevanţă. Totodată sunt importante şi condiţiile ce trebuiesc îndeplinite astfel încât metoda să poată fi aplicată.

O altă metodă de a aborda o eventuală clasificare a metodelor de analiză şi atac o reprezintă împărţirea acestora pe categorii după proprietăţile utilizate în cadrul atacului. Putem enumera astfel:

- metode elementare de atac;
- metoda exponentului privat mic;
- metodele exponentului public mic;
- atacuri îndreptate împotriva diverselor implementări.

Factorizarea numerelor întregi

O metodă de a ataca algoritmul RSA este aceea de a încerca factorizarea modulului N. Aceasta mai este numită şi metoda forţei brute. Dacă factorizarea lui N reuşeşte, este foarte simplu de a se calcula apoi (p-1)*(q-1) şi de a se găsi exponentul de decriptare. În practică aceasta însemnă compromiterea cheii private, permiţând atacatorului atât citirea mesajelor, cât şi falsificarea semnăturilor.

O perioadă de timp s-a crezut că securitatea RSA este direct proporţională cu problema factorizării numerelor prime, dar cercetări recente au dezvăluit faptul că "spargerea" unui sistem RSA nu este echivalentă cu factorizarea lui N. Cu alte cuvinte, s-a dovedit că a "sparge" un sistem RSA cu exponent privat mic este mai simplă decât rezolvarea problemei factorizării lui N. Aceasta însă nu înseamnă neapărat găsirea unei vulnerabilităţi a algoritmului [2].

Securitatea RSA ar fi puternic afectată dacă s-ar reuşi găsirea unei metode uşoare de factorizare a numerelor mari şi foarte mari. Doar avansurile tehnologice în domeniul hardware (în particular viteza crescută de procesare) nu pot duce la o ameninţare reală a algoritmului, deoarece prin alegerea unor chei cu lungime tot mai mare, această problemă se rezolvă de la sine. De notat este şi faptul că algoritmul cel mai rapid de factorizare este "General Number Field Sieve", care rulează pe numere de *n* biţi lungime în

$\exp\left((c+O(1))*n^{1/3}*\log^{2/3} n\right)$, unde c < 2.

De aceea, pentru ca un sistem criptografic de tip RSA să fie sigur se recomandă ca în practică N să aibă o lungime de cel puţin 1024 de biţi, dat fiind faptul că în ultima vreme s-au obţinut rezultate bune în rezolvarea factorizării lui N (cu mijloacele actuale s-a reuşit factorizarea unor numere N de lungime de 512 biţi).

O altă metodă de a ataca RSA, dar care nu este echivalentă cu problema factorizării numerelor prime, este găsirea unei metode rapide de calcul a rădăcinii la *e*. Deoarece $C = M^e \bmod N$, acest atac ar permite oricui decriptarea datelor şi falsificarea semnăturilor, chiar şi în absenţa cheii private. Nu se cunoaşte însă o aplicaţie practică în care această metodă să fi fost folosită cu succes, dar în anumite cazuri, dacă se foloseşte un exponent public mic, este teoretic posibilă aplicarea ei şi decriptarea unor mesaje.

Cele două metode descrise anterior sunt metode cu ajutorul cărora se pot recupera toate mesajele criptate cu o anumită cheie. Există însă şi metode ce au ca ţintă aflarea textului unui singur mesaj criptat. Un succes în această direcţie nu va permite însă atacatorului să decripteze şi alte mesaje criptate cu aceeaşi cheie. Dacă, de exemplu, metoda factorizării duce practic la compromiterea cheii private, nu acesta este rezultatul în cazul altor tipuri de atac.

Metode elementare de atac

Există un număr de metode elementare de atac, care deşi nu au o relevanţă mare, demonstrează totuşi că folosirea diletantă a sistemului poate duce la compromiterea ideii de securitate. Cel mai simplu atac de acest gen este încercarea de a ghici textul. Atacatorul presupune textul şi îl criptează cu cheia publică pentru a verifica dacă rezultatul obţinut este acelaşi cu mesajul criptat. Metoda cea mai simplă de a preîntâmpina acest atac este adăugarea unor biţi aleatori în mesajul original.

O altă metodă relativ banală o reprezintă metoda modulului comun N. Pentru a evita generarea unor numere prime mari pentru fiecare utilizator, s-a încercat, de exemplu, utilizarea aceluiaşi modul N pentru un grup de utilizatori. O autoritate centrală distribuie în această situaţie perechile (e_i, d_i) pentru fie-

care utilizator. La o analiză atentă se poate observa însă faptul că fiecare membru al grupului poate utiliza perechea sa (e_i, d_i), pentru a factoriza N. Odată N factorizat, utilizatorul poate restaura orice cheie privată a unui alt membru al grupului. De aici rezultă şi concluzia că modulul N nu trebuie folosit niciodată în comun de către membrii unui grup. O altă metodă clasică, numită "blinding", facilitează obţinerea unei semnături false asupra unui mesaj M. Cel ce doreşte obţinerea semnăturii produce un mesaj $M' = r^e * M \bmod N$, apoi îl trimite spre semnare victimei. În cazul în care destinatarul semnează mesajul şi îl trimite înapoi, nu a făcut altceva decât să producă o semnătură $S' = M'^d \bmod N$. Falsificatorul nu trebuie decât să calculeze $S = S'/r \bmod N$ pentru a obţine semnătura pentru mesajul M. De obicei însă semnăturile se aplică asupra amprentei unui mesaj (obţinută prin aplicarea unui hash unidirecţional), şi nu direct asupra mesajului, caz în care atacul îşi pierde însemnătatea. Această proprietate a RSA este totuşi foarte importantă pentru implementarea a ceea ce se numeşte "anonymous digital cash" (atunci când identitatea persoanei ce semnează nu trebuie dezvăluită).

Metoda exponentului privat mic

Această metodă de atac se concentrează asupra cazului în care se foloseşte un exponent de valoare mică. Alegerea unui exponent privat cât mai mic are loc atunci când se doreşte reducerea timpului de decriptare, sau a celui de semnare a datelor. În acest caz se poate observa o îmbunătăţire a performanţei algoritmului cu un factor de ordinul de mărime 10. M. Wiener a demonstrat că, tocmai în aceste cazuri, sistemul RSA prezintă o vulnerabilitate excesivă, atacatorul putând afla exponentul d. El a formulat teorema ce stă la baza acestei metode de atac:

*Fie N=p*q astfel încât q<p<2*q şi* $d < \frac{1}{3} * N^{1/4}$. *Dată fiind ecuaţia de forma: e * d=1mod(p-1)(q-1), se poate afla d printr-un număr rezonabil de calcule aritmetice.*

Demonstraţia se bazează pe aproximaţii ce utilizează fracţii continue. Din existenţa unui k întreg, ecuaţia mai sus amintită se poate rescrie după cum urmează:

$\left|\frac{e}{(p-1)(q-1)} - \frac{k}{d}\right| = \frac{1}{d(p-1)(q-1)}$, rezultând că $\left|N - (p-1)(q-1)\right| < 3\sqrt{N}$. Ulterior se obţine o inecuaţie de forma: $\left|\frac{e}{N} - \frac{k}{d}\right| < \frac{1}{d * N^{1/4}} < \frac{1}{2d^2}$.

Numărul de fracţii $\frac{k}{d}$ cu d < N ce aproximează $\frac{e}{N}$ nu este mai mare de $\log_2 N$. Această metodă poate fi utilizată ca algoritm liniar de aflare a valorii d, având ca efect compromiterea cheii private. Pentru un modul N de lungime 1024 de biţi, d ar trebui să aibă o lungime de cel puţin 256 biţi pentru a evita acest atac (ceea ce reprezintă câteodată o problemă în domeniul smartcard-urilor). Wiener a prezentat şi alte câteva metode de a preîntâmpina acest atac [3], printre care enumerăm alegerea unui exponent privat e suficient de mare. Dacă e > $N^{1,5}$ atacul nu mai poate fi realizat, din păcate însă odată cu creşterea lui e creşte şi timpul necesar criptării sau verificării semnăturilor. Nu s-a demonstrat totuşi cât de sigure sunt aceste metode de preîntâmpinare a acestui atac. Boneh şi Durfee au demonstrat că pentru $d < N^{0,292}$, exponentul privat se poate recupera în mod eficient, indiferent de alegerea celorlalte variabile ale sistemului [4].

Metodele exponentului public mic

În practică se preferă de multe ori utilizarea unui exponent public mic pentru a îmbunătăţi timpul de criptare sau de verificare a semnăturii. Cu toate că şi metodele exponentului public atacă valorile mici ale acestuia, cu rezultate adeseori satisfăcătoare, pericolul de compromitere a cheii private este mai mic decât în cazul metodei precedente. Cea mai mică valoare ce poate fi aleasă pentru exponentul public este 3, dar de obicei pentru a preîntâmpina anumite atacuri se alege valoarea 65537, ce pare a fi mai sigură.

❖ **Teorema lui Coppersmith. Metoda Hastad ("broadcast attack")**

S-a observat că, unele dintre cele mai puternice atacuri îndreptate împotriva algoritmului

RSA ce utilizează un exponent public mic au la bază o teoremă enunțată de cercetătorul Coppersmith:

Fie N un număr întreg şi $f \in Z_N[x]$ *un polinom monic de gradul d. Fie* $X = N^{\frac{1}{d}-\varepsilon}$, *unde* $\varepsilon > 0$. *Dată fiind perechea (N, f), se pot afla într-un timp eficient toți întregii* $|x_0| < X$, *ce satisfac relația* $f(x_0) = 0 \bmod N$. *Timpul de rulare este dominat de timpul în care algoritmul LLL rulează pe o latice de dimensiunea O(w) cu* $w = \min(1/\varepsilon, \log_2 N)$.

Teorema oferă o cale eficientă de a afla rădăcinile lui f mod N, iar demonstrația sa se bazează la rândul său pe o lemă datorată lui Howgrave-Graham ce poate fi studiată în [5].

O primă aplicație a teoremei lui Coppersmith este îmbunătățirea unei metode mai vechi, atribuită lui Hastad, ce poate fi întâlnită în literatura de specialitate şi sub numele de "broadcast attack". Pentru a trimite un mesaj M unui număr k de destinatari, expeditorul trebuie să cripteze mesajul cu cheile publice corespunzătoare (N_i, e_i). Presupunând, pentru simplitate că toți membrii grupului au e_i = 3 şi că grupul este alcătuit din 3 persoane (k=3), se obțin următoarele relații:

$$\begin{cases} C_1 = M^3 \bmod N_1 \\ C_2 = M^3 \bmod N_2 \\ C_3 = M^3 \bmod N_3 \end{cases}.$$

Se poate presupune şi faptul că cmmdc(N_i, N_j)=1, pentru orice $i \neq j$. Aplicând teorema chinezească a restului se obține egalitatea C'=M³ mod $N_1 N_2 N_3$, criptanalistul având posibilitatea de a recupera mesajul M, cu condiția de a putea calcula rădăcina cubică a lui C'. După cum lesne se poate observa, pentru exponenți publici mici, acest atac devine realizabil de îndată ce $k \geq e$.

Hastad menționează în continuare un atac şi mai puternic: dacă se încearcă adăugarea unei secvențe de forma $i * 2^m$ la mesajul inițial, aceasta nu va împiedica atacul descris anterior. Hastad a demonstrat, enunțând o teoremă ce îi poartă numele, că aplicarea oricărui polinom liniar de completare, ce aduce mesajul inițial la forma $M_i = i * 2^m + M$, nu împiedică atacul. Singura posibilitate de a preîntâmpina atacul în condițiile date este completarea mesajului cu o secvență aleatoare [6].

❖ **Metoda mesajelor înrudite (Franklin-Reiter). Metoda secvenței de completare (Coppersmith).**

O altă metodă interesantă ce poate fi utilizată pentru a descifra mesaje aflate într-o anumită relație unul față de celălalt, este cunoscută şi sub numele de metoda Franklin-Reiter, după numele celor doi cercetători care au descoperit-o:

Presupunând că mesajele M_1 *şi* M_2 *sunt distincte şi satisfac o relație de forma* $M_1 = f(M_2) \bmod N$, *şi că* C_1, C_2 *şi funcția* $f \in Z_N[x]$ *sunt cunoscute, atunci se pot afla mesajele* M_1 *şi* M_2 *pentru orice exponent public e mic.*

Atacul se bazează pe următoarea lemă:

Fie e = 3 şi (N, e) o cheie publică RSA. Fie $M_1 \neq M_2$, *ce satisfac relația* $M_1 = f(M_2) \bmod N$, *unde f este o funcție liniară* $f = ax + b$ *în* $Z_N[x]$, *şi* $b \neq 0$. *Cunoscându-se tuplul (N, e,* C_1, C_2, *f), atunci se pot recupera* M_1 *şi* M_2 *într-un timp pătratic în log N.*

Lema funcționează şi pentru e ≠ 3, dar atâta timp cât e este mic. O demonstrație a acesteia poate fi găsită în [7].

Metoda Franklin-Reiter pare pentru început un pic artificială, apare firesc întrebarea de ce ar dori cineva să trimită mesaje M_1 şi M_2 ce se află într-o anumită relație? Coppersmith a prezentat o extindere a atacului împotriva unei secvențe de completare aleatoare, ce se bazează tocmai pe metoda Franklin-Reiter, enunțând următoarea teoremă:

Fie (N, e) o cheie publică RSA în care modulul N are n biți lungime, m = $\left\lfloor \frac{n}{e^2} \right\rfloor$, *şi M este un mesaj de lungime n - m biți. Se fixează* $M_1 = 2^m * M + r_1$ *şi* $M_2 = 2^m * M + r_2$, *unde* r_1 *şi* r_2 *sunt doi întregi distincți astfel încât:* $0 < r_1, r_2 < 2^m$. *Dacă se cunosc* C_1 *şi* C_2,

atunci se pot recupera M_1 *şi* M_2 *într-un mod eficient.*

Dacă se definesc $g_1(x,y) = x^e - C_1$, $g_2(x,y) = (x+y)^e - C_2$, şi $y = r_2 - r_1$, atunci cele două polinoame au ca rădăcină comună M_1. Cu alte cuvinte, $\delta = r_2 - r_1$ este o rădăcină a lui $h(y) = res_x(g_1, g_2)$, iar gradul lui h este cel mult e². Se cunosc totodată următoarele: $|\delta| < 2^m < N^{1/e^2}$, şi δ este o rădăcină mică a lui $h \bmod N$. De aici rezultă că se poate calcula δ utilizând teorema anterioară a lui Coppersmith. Odată δ calculat, se poate aplica metoda Franklin-Reiter.

Metoda funcționează pentru valoarea e=3 atâta timp cât secvența de completare este mai scurtă decât $\frac{1}{9}$ din lungimea mesajului.

Pentru valoarea e = 65537, atacul nu mai poate fi aplicat.

❖ **Expunerea parțială a cheii**

S-a pus deseori întrebarea ce se întâmplă atunci când o parte a cheii private, să zicem un sfert din biții acesteia sunt expuşi? Poate duce aceasta la compromiterea totală a cheii private? Răspunsul la această întrebare este afirmativ, în cazul în care exponentul public este mic.

Recent Boneh, Durfee şi Frankel [8] au demonstrat că este posibil să se reconstruiască cheia privată, dacă se cunoaşte o secvență din biții acesteia. Condiția ce trebuie îndeplinită este ca $e < \sqrt{N}$. Acest rezultat demonstrează încă odată cât de importantă este păstrarea secretă a întregii chei private. Iată şi teorema ce stă la baza acestui atac:

Fie (N, d) o cheie privată RSA, în care modulul N are n biți lungime. Dacă se cunosc cei mai puțin semnificativi $\frac{n}{4}$ *biți ai lui d, cheia privată poate fi reconstruită într-un timp liniar cu* $O(e \log_2 e)$.

Demonstrația teoremei se bazează pe o alta teoremă, datorată lui Coppersmith:

*Fie N = p * q un modul RSA de n biți lungime. Dați fiind cei mai semnificativi* $\frac{n}{4}$ *biți ai lui p sau cei mai puțini semnificativi* $\frac{n}{4}$ *biți ai lui p, se poate factoriza N într-un mod eficient.*

Metoda funcționează atât pentru un exponent public mic, cât şi pentru valori ceva mai mari, dar numai cât timp se respectă inegalitatea $e < \sqrt{N}$, în ultimul caz calculele fiind însă ceva mai complicate. Este interesant de reținut şi aspectul că sistemele criptografice ce se bazează pe logaritmi discreți (de exemplu ElGamal), nu sunt susceptibile la atacul expunerii parțiale a cheii private.

Se poate lesne observa că dintre metodele prezentate în clasa exponentului public mic, acest ultim atac este unul dintre cele mai periculoase. Deşi presupune cunoaşterea unei fracțiuni din cheia privată, această condiție nu este imposibil de îndeplinit în practică. De multe ori, folosirea diletantă a sistemului sau neglijența în utilizarea acestuia au avut rezultate catastrofale şi compromiterea ideii de securitate.

Atacuri îndreptate împotriva diverselor implementări

O clasă aparte între metodele prezentate o constituie atacurile îndreptate împotriva diverselor implementări. Există un număr de atacuri ce țintesc nu atât algoritmul RSA, ci diverse implementări ale acestuia. Acestea nu pot fi considerate însă puncte slabe ale algoritmului, ci ale implementărilor respective. Pentru a obține o securitate ridicată nu este suficient doar de a avea o cheie cu un număr cât mai mare de biți, este necesară atât păstrarea acesteia într-un loc sigur, cât şi o implementare cât mai robustă a algoritmului (atacurile împotriva diverselor implementări se concentrează de obicei pe slăbiciuni în managementul cheilor). Fără a avea pretenția de a epuiza acest capitol, prezentăm în continuare câteva exemple practice de atacuri încununate de succes, şi domeniul de aplicație al acestora.

❖ **Atacul "random faults"**

Multe din implementările RSA folosesc teorema chinezească a restului ("Chinese Remainder Theorem" - CRT) în cadrul operațiilor de decriptare sau semnare, pentru a

calcula cât mai repede $M^d \bmod N$. În loc de a lucra cu modulul N, se lucrează cu numerele p şi q, combinând apoi rezultatele cu ajutorul CRT. Obţinem astfel $C_p = M^{dp} \bmod p$ şi $C_q = M^{dq} \bmod q$, unde $d_p = d \bmod (p-1)$ şi $d_q = d \bmod (q-1)$. Semnătura se obţine în final prin combinarea celor doi termeni astfel:

$C = T_1 * C_p + T_2 * C_q$, unde $T_1 = \begin{Bmatrix} 1 \bmod p \\ 0 \bmod q \end{Bmatrix}$,

$T_2 = \begin{Bmatrix} 0 \bmod p \\ 1 \bmod q \end{Bmatrix}$

Deoarece p şi q au jumătate din lungimea modulului N, şi deoarece implementările simple ale multiplicărilor necesită un timp pătratic, rezultă că, calculele modulo p vor fi de 4 ori mai rapide decât calculele modulo N. În acest caz, timpul de realizare al semnăturii este de 4 ori mai mic decât în mod normal.

S-a demonstrat că, atunci când se foloseşte această metodă, dacă dintr-un anumit defect (hardware: un bit resetat în mod incorect; software: un bug în program) se realizează o semnătură falsă, aceasta va duce la compromiterea cheii, N putând fi uşor factorizat [10].

Presupunem în continuare că la generarea semnăturii apare o singură eroare. Aceasta are ca rezultat faptul că unul din cei doi termeni C_p sau C_q este incorect (fie $C_q^{'}$ acesta).

Semnătura rezultată $C' = T_1 * C_p + T_2 * C_q^{'}$ este falsă, deoarece $C'^e \neq M \bmod N$. Avem însă şi alte două relaţii: $C'^e = M \bmod p$ şi $C'^e \neq M \bmod q$. De aici rezultă că cel mai mare divizor comun al perechii (N, $C'^e - M$) expune un factor al lui N.

Pentru ca atacul să funcţioneze, se presupune cunoaşterea lui N şi nefolosirea unor adaosuri aleatoare la mesajul M. Dacă se adaugă o serie aleatoare de biţi la mesaj, înainte ca acesta să fie semnat, atacul nu mai poate fi instrumentat. O altă metodă de a preîntâmpina acest atac, sau atacuri similare, în special atunci când se foloseşte metoda CRT pentru a obţine o viteză mai mare la producerea semnăturii, este aceea de a verifica semnătura, înainte de a o face publică.

❖ **Metode bazate pe durata de execuţie ("Timing Atacks")**

Aceasta este o metodă des utilizată în implementările RSA pentru smartcard-uri. Presupunând că nu se poate accesa cheia privată stocată în smartcard, există totuşi posibilitatea de a măsura exact timpii [9] necesari pentru ca smartcard-ul să efectueze diverse operaţii (decriptare/semnare). Examinarea acestor timpi duce în final la descoperirea exponentului privat d, având ca efect compromiterea cheii private.

Pentru exemplificare, considerăm în continuare o implementare simplă a algoritmului RSA, ce calculează $C = M^d \bmod N$ folosind 2*n multiplicări modulare. Fie $d = d_n..d_0$ reprezentarea binară a exponentului privat. Dacă se rescrie d ca sumă, obţinem: $\sum_{i=0}^{n} 2^i * d_i$, unde $d_i \in \{0,1\}$. Putem rescrie textul criptat după cum urmează:

$$C = \prod_{i=0}^{n} M^{2^i * d_i} \bmod N$$

Algoritmul funcţionează astfel: se atribuie lui z valoarea M şi lui C valoarea 1, apoi se repetă paşii pentru i=0..n

- dacă $d_i = 1$ se calculează $C = C * z \bmod N$;
- se calculează $z = z^2 \bmod N$.

La sfârşit C are valoare $M^d \bmod N$. Pentru a instrumenta atacul, se cere smartcard-ului generarea unor semnături pentru un număr mare de mesaje, măsurând timpul necesar pentru calcularea fiecărei semnături (T_i). În acest timp se descoperă exponentul privat d bit cu bit. d_0 este 1, deoarece d trebuie să fie impar. Iniţial $z = M^2 \bmod N$, iar $C = M$. Dacă d_1 = 1, smartcard-ul va calcula produsul $C * z = M * M^2 \bmod N$. Fie timpul t_i, care este necesar smartcard-ului pentru a calcula $M_i * M_i^2 \bmod N$. Valorile t_i vor depinde de valorile M_i, dar persoana ce instrumentează atacul are posibilitatea de a măsura off-line aceşti timpi, dacă posedă specificaţiile card-ului. Astfel, se poate stabili dacă d_1 are valoarea 0 sau 1. Procedând mai departe

după acelaşi raţionament se pot afla valorile biţilor d_2, d_3 , etc.

O metodă de apărare împotriva acestui atac este introducerea unor întârzieri, care să aibă ca rezultat calculul exponentului modular într-un timp fix, indiferent de mesajul de criptat. Există şi posibilitatea de a utiliza metoda de "blinding" (prezentată în capitolul "*Metode elementare de atac*"), datorată lui Rivest. Smartcard-ul alege un număr r aleator, calculează $M' = M * r^e \bmod N$, apoi, cu noul M' se calculează C'. În final $C = C' / r \bmod N$. Aceasta are ca rezultat faptul că smartcard-ul efectuează calcule asupra unui mesaj M' aleator, atacul nemaiputând fi instrumentat.

Kocher a prezentat recent o altă metodă asemănătoare de atac, numită criptanaliza puterii. El a arătat că, măsurând cu atenţie puterea consumată de smartcard în timpul generării unei semnături, se poate descoperi destul de uşor cheia secretă. După cum a rezultat, în timpul unei multiplicări de precizie, consumul smartcard-ului este mai ridicat decât de obicei. Măsurând lungimea acestor perioade de consum ridicat se poate stabili câte multiplicări face smartcard-ul, dezvăluind astfel valoarea bitului respectiv [9].

❖ **Metoda Bleichenbacher (PKCS1)**

Dacă N este un modul RSA de n biţi lungime, iar M este un mesaj alcătuit din m biţi astfel încât m < n, unele versiuni de software obişnuiesc de a completa mesajul M cu un număr de biţi aleatori până la lungimea n - o versiune mai veche a PKCS1 procedează asemănător. După completare, mesajul arată astfel:

0x02	Random	00	M

În varianta respectivă, valoarea 0x02 ocupă 16 biţi şi specifică faptul că s-a utilizat o completare aleatoare a mesajului. La recepţie, destinatarul decriptează mesajul, încercând să îndepărteze completarea aleatoare. Unele implementări au returnat în acest moment eroarea "text cifrat invalid", dacă nu au întâlnit codul 0x02 la începutul secvenţei. Bleichenbacher a demonstrat că, profitând de acest mesaj, un atacator poate decripta texte cifrate [11].

Presupunând că atacatorul recepţionează textul cifrat C, el alege un r aleator şi calculează $C' = r * C \bmod N$ şi trimite noul C' destinatarului. Pe baza răspunsului ce vine de la destinatar, el va afla dacă decriptarea primilor 2 octeţi a avut ca rezultat valoarea 0x02 sau nu. De fapt, atacatorul are la dispoziţie un "oracol" care îi va prezice întotdeauna cu exactitate dacă primii doi octeţi se decriptează în valoarea 0x02 sau nu, pentru orice r ales aleator. Bleichenbacher a demonstrat că un astfel de "oracol" este suficient pentru a decripta mesajul criptat.

Algoritmul RSA în aplicaţii practice

În practică, RSA este foarte des utilizat împreună cu algoritmi cu cheie simetrică (de exemplu DES). Se generează o cheie DES, cu care se criptează mesajul. Apoi, cheia simetrică se criptează cu ajutorul cheii publice a persoanei căreia îi este destinat mesajul şi se trimite destinatarului împreună cu mesajul criptat (acestea două formează un "plic digital" RSA). Destinatarul va decripta mai întâi cheia DES cu ajutorul cheii sale private, apoi mesajul, cu ajutorul cheii simetrice, obţinută din prima decriptare. Cheia DES poate fi în continuare utilizată şi ca o cheie de sesiune. Pentru semnarea unui mesaj, mai întâi se creează o amprentă digitală ("message digest") a acestuia cu ajutorul unei funcţii hash. Aceasta se criptează cu ajutorul cheii private, rezultatul urmând a fi trimis destinatarului. Pentru verificarea semnăturii, se decriptează mesajul cu ajutorul cheii publice a semnatarului, obţinând astfel amprenta digitală, care va fi comparată cu cea obţinută aplicând din nou funcţia hash asupra mesajului. Dacă cele două amprente sunt identice, rezultă faptul că semnătura digitală este autentică.

În momentul de faţă, RSA este utilizat într-o varietate de produse, platforme şi standarde. El poate fi întâlnit în sisteme de operare, precum: Microsoft, Apple, Sun sau Novell, în componente hardware, precum: sisteme telefonice, card-uri de reţea sau smartcard-uri, în protocoale de comunicaţie, precum: S/MIME, SSL, IPSec, PKCS sau S/WAN. El este în mod sigur cel mai răspândit algoritm cu cheie publică utilizat la ora actuală.

Concluzii

La ora actuală există o multitudine de metode de atac şi o mulţime de vulnerabilităţi ce trebuie luate în considerare pentru a putea programa o versiune robustă de RSA. Numai o analiză atentă şi o cunoaştere amănunţită a acestora poate duce la realizarea unei implementări robuste. Metodele prezentate au atât o valoare teoretică cât şi una practică, în funcţie de impactul pe care îl au asupra algoritmului. Prezentăm în continuare un set de măsuri ce trebuie îndeplinite pentru a preîntâmpina vulnerabilităţile prezentate anterior:

- folosirea unor chei de lungime potrivită (de obicei se caută un compromis între nivelul de securitate şi viteză);
- adăugarea unei secvenţe aleatoare de completare a mesajului;
- folosirea unor valori ale exponentului privat ce îndeplinesc condiţia $d < N^{0,292}$, pentru a împiedica atacul exponentului privat;
- evitarea folosirii unor anumite valori întregi pentru exponentul public (de exemplu 3);
- semnarea amprentei (hash-ului) unui mesaj, nu a mesajului în sine;
- evitarea producerii de semnături sau criptări false, eventual prin verificarea lor ulterioară înainte de publicare, dacă nu există altă metodă;
- introducerea de timpi aleatori în algoritm, acolo unde este posibilă o analiză temporală a acestuia;
- evitarea de a răspunde automat la toate cererile primite, pentru a nu transforma serverul în "oracol", eventual realizarea unui jurnal pentru a putea păstra o istorie a cererilor primite.

În ultimul timp a devenit clar că sistemele cu chei publice sunt un mecanism indispensabil atât pentru managementul cheilor cât şi pentru comunicaţiile sigure. Ceea ce este mai puţin clar este modalitatea de a alege cel mai bun sistem într-o anumită situaţie. Unul dintre criteriile cele mai des folosite pentru a alege îl constituie tehnica utilizată de algoritm. Fără o cunoaştere profundă a acesteia, a vulnerabilităţilor, dar mai ales a metodelor de atac, cu greu mai putem concepe astăzi programarea unui versiuni robuste şi sigure a unui algoritm criptografic.

Mai mult de două decenii de atacuri împotriva RSA au produs o serie de atacuri interesante, dar nu au fost găsite (până în prezent) metode astfel încât algoritmul să fie compromis. Atacurile prezentate şi în această lucrare subliniază în mare parte capcanele ce ar trebui ocolite atunci când se implementează algoritmul. Se poate deci presupune că implementările RSA, ce respectă un set de reguli bine stabilit, pot furniza un grad ridicat de securitate [12] [13].

Bibliografie

1. Bellare, M., Rogaway, P.: Optimal asymmetric encryption, Eurocrypt '94, Lecture Notes in Computer Science, Springer-Verlag, vol. 950, 1994, p. 92-111.
2. Boneh, D., Venkatesan, R.: Breaking RSA may not be equivalent to factoring, Advances in Cryptology - Eurocrypt '98, Springer-Verlag, 1998, p. 59-71.
3. Wiener, M.: Cryptanalysis of short RSA secret exponents, IEEE Transactions on Information Theory, 1990, vol.36, p.553-558.
4. D. Boneh, D., Durfee, G.: New results on cryptanalysis of low private exponent RSA, Preprint, 1998.
5. Howgrave-Graham, N.: Finding small roots of univariate modular equations revisited, Cryptography and Coding, Lecture Notes in Computer Science, Springer-Verlag, vol. 1355, 1997, p. 131-142.
6. Hastad, J.: Solving simultaneous modular equations of low degree, SIAM Journal of Computing, 1988, p.336-341.
7. Coppersmith, D., Franklin, M., Patarin, J., Reiter, M.: Low-exponent RSA with related messages, Eurocrypt '96, Lecture Notes in Computer Science, Springer-Verlag , vol. 1070, 1996, p. 1-9.
8. D. Boneh, D., Durfee, G., Frankel, Y.: An attack on RSA given a fraction of the private key bits, AsiaCrypt '98, Lecture Notes in Computer Science, Springer-Verlag, vol. 1514, 1998, p. 25-34.
9. Kocher, P.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems, Crypto '96, Lecture Notes in Computer Science, Springer-Verlag, vol. 1109, 1996, p. 104-113.
10. Boneh, D., DeMillo, R., Lipton, R.: On the importance of checking cryptographic protocols for faults", Eurocrypt '97, Lecture Notes in Computer Science, Springer-Verlag, vol. 1233, 1997, p.37-51.
11. Bleichenbacher, D.: Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1, Crypto '98, Lecture Notes in Computer Science, Springer-Verlag, vol. 1462, 1998
12. Boneh., D.: Twenty years of attacks on the RSA cryptosystem, American Mathematical Society, vol. 46, No. 2, 1999, p. 203-213.
13. Wiener, M. J.: Performance Comparison of Public-Key Cryptosystems, CryptoBytes, Vol.4, Nr.1, 1998