Nahrbtniški kriptosistemi

Projekt pri seminarju za kriptografijo

Peter Luževič

24.5.1999

Kazalo

1 Uvod

V svojem projektu predstavljam šifriranje podatkov na osnovi problema nahrbtnika ter kako so ga razbijali. Koncept tovrstnega šifriranja podatkov sta predlagala Merkle in Hellman leta 1978 v [2]. Svoje čase se je to zdela privlačna alternativa sočasno iznajdenemu šifrirnemu sistemu RSA, predvsem zaradi do 100-krat večje hitrosti ([3]). Po drugi strani pa je velikost tajnopisa ter javnega ključa dosti večja kot pri RSA. Že od začetka pa so se pojavljali tudi dvomi. Denimo, v [2] ni bila dokazana težavnost razbijanja ključa, pač pa sta se avtorja zanašala na NP-polnost odločitvenega problema za polnjenje nahrbtnika. Teorija NP-polnosti pa obravnava zgornje časovne meje reševanja problemov. Kaj pa, če je večina konkretnih problemov lahko rešljivih? In tudi če ne, bi pa kriptoanalitik morda lahko iz javnega ključa kako sklepal na privatnega. Še drug dvom je nastal zaradi Brassardovega rezultata ([3] navaja [13]), da če je razbijanje kriptosistema NP-težko, da je potem NP=co-NP, kar pa bi bilo zelo presenetljivo. Če je NP≠co-NP, potem razbijanje Merkle–Hellmanovega (MH) nahrbtnika ni NP-težko in bi utegnilo biti lažje kot reševanje splošnega problema nahrbtnika. Nazadnje, z informacijo, ali je vsota, ki je šifrirano besedilo, soda ali liha, že dobimo 1 bit informacije o čistopisu. Sicer (po [3]) tega nikomur ni uspelo izkoristiti, a že dejstvo, da sistem "pušča" informacijo, je bilo mnogim sumljivo.

Napadi na nahrbtniške kriptosisteme so se začeli že konec sedemdesetih let, najvažnejši pa je bil leta 1982 Shamirjev ([4]) v polinomskem času na osnovno verzijo MH nahrbtnika. Sledilo jih je še več, omeniti velja Brickellovega ([5]) leta 1984 na iteriran MH nahrbtnik [1]. Po vsakem napadu so kriptografi izdelali popravljene različice sistema, vendar se je za skoraj vse našel kdo, ki je razbil tudi te. Zato so nahrbtniški kriptosistemi v glavnem izgubili zaupanje strokovnjakov. Zdi pa se (po [9]), da so se dali ti sistemi razbiti zato, ker preslabo zamaskirajo lahko rešljiv nahrbtnik, ki se uporablja za ključ. Do danes je ostal delno nerazbit le Chor–Rivestov (CR) nahrbtniški kriptosistem, ki je hkrati tudi edini, ki za maskiranje ne uporablja množenja po modulu. (Leta 1995 sta Schnorr in Hörner —[7]— objavila napad na CR nahrbtnik, vendar ne za tako velike vrednosti parametrov, ki jih priporočata avtorja.) Občasno pa še danes kdo poskusi modificirati sistem ([11]).

2 Merkle–Hellmanov nahrbtnik

V tem razdelku po [2] in [3] opisujem prvotni sistem in omenjam možne različice.

2.1 Enostavni MH nahrbtnik

Problem, na katerem sloni koncept, pravzaprav ni problem nahrbtnika, ampak podoben problem vsote podmnožice ([1, str. 190]). Zastavljen je takole: imejmo n-terko števil $\mathbf{a} = (a_1, \ldots, a_n)$ ("velikosti predmetov") ter število S ("velikost nahrbtnika"). Najti je treba podmnožico velikosti predmetov, ki točno napolni nahrbtnik — če taka množica obstaja; oziroma vektor $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{Z}_2$, da je skalarni produkt $\mathbf{x}.\mathbf{a} = S$. Odločitvena oblika tega problema je NP-polna. Z metodo grobe sile je treba preveriti vseh 2^n možnosti za $\mathbf{x}$ v času, jasno, $O(2^n)$. Doslej časovno najugodnejša metoda je Schroepplova ([2], [3]), ki potrebuje $O(n2^{n/2})$ časa, vendar tudi $O(2^{n/2})$ prostora.

Ideja sistema je, da naj bodo velikosti predmetov javni ključ, niz $\mathbf{x}$ iz ničel in enk čistopis, vsota velikosti izbranih predmetov S pa tajnopis, ki ga pošiljatelj pošlje naslovniku. Pojavita se problema:

- ali je $\mathbf{x}$ enolično določen z S, ter
- kako naj sam naslovnik hitro dobi $\mathbf{x}$ iz S?

Naslovnik bo moral imeti tak privatni ključ $\mathbf{b}$, da bo z lahkoto iz S rekonstruiral $\mathbf{x}$. Lahko bi si recimo izbral $b_i = 2^{i-1}$, potem bi bil $\mathbf{x}$ kar enak binarnemu zapisu od S. Prav tako je ustrezna (ne rečem, da

[1] Ta dva napada sta slavna tudi zato, ker je Merkle (po [9], [10]) zaradi njiju izgubil stavi — plačal je 100 USD Shamirju in 1000 USD Brickellu.

dobra) izbira *supernaraščajoče zaporedje*: za vsak $i \in \{1, \ldots, n\}$ je

$$b_i > \sum_{j=1}^{i-1} b_j.$$

Tedaj lahko iz S enostavno dobimo $\mathbf{x}$:

$$x_i = \begin{cases} 1, & S - \sum_{j=i+1}^{n} x_j b_j \geq b_i, \\ 0, & \text{sicer.} \end{cases}$$

Toda $\mathbf{x}$ naj bi se kodiral z javnim ključem. Zagotoviti je treba, da se bo iz vsote, dobljene z javnim ključem, dal rekonstruirati $\mathbf{x}$. Naslovnik si skrivaj izbere dve veliki števili M in W tako, da je $M > \sum_{j=1}^{n} b_j$ in $\gcd(M, W) = 1$. Nato pa privatni ključ $\mathbf{b}$ pretvori v javnega $\mathbf{a}$ z množenjem po modulu:

$$a_i = W b_i \pmod{M}.$$

Zdaj ima $\mathbf{a}$ na prvi pogled naključne komponente. Pošiljatelj izračuna $S = \mathbf{a}.\mathbf{x}$ in ga odpošlje. Naslovnik si izračuna $S' = SW^{-1} \pmod{M}$, iz tega pa lahko z $\mathbf{b}$ rekonstruira $\mathbf{x}$, saj je

$$S' = W^{-1}S = W^{-1} \sum x_i a_i = W^{-1} \sum x_i W b_i = \sum x_i b_i \pmod{M}.$$

Modul mora biti dovolj velik prav zato, da zgornja enakost velja tudi v $\mathbb{Z}$, ne le v $\mathbb{Z}_M$. Da bi se še bolj zakrila struktura nahrbtnika, si lahko naslovnik izbere še permutacijo n indeksov in kot javni ključ objavi seznam

$$a_i = W b_{\pi(j)} \pmod{M}. \tag{1}$$

Priložen je programček v Mahematici, s katerim lahko generiramo superrastoče zaporedje za privatni ključ, tega z naključnim (a praštevilskim) modulom in naključnim številom (obrnljivim glede na modul) maskiramo v javni ključ, s tem zakodiramo dano število (oziroma njegov dvojiški zapis) v vsoto, to pa s privatnim ključem odkodiramo nazaj v prvotno dano število.

2.2 Variante

Množeči (ang. multiplicative) nahrbtnik je tak, pri katerem se velikosti predmetov ne seštevajo, ampak množijo. Z lahkoto je rešljiv, kadar so velikosti predmetov tuje. V običajni nahrbtnik se pretvori z logaritmi, ki jih je treba jemati po $\mathbb{Z}_M$, M praštevilo. Naslovnik si mora skrivaj izbrati M, osnovo b ter seznam $\{b_1, \ldots, b_n\}$ tujih števil za privatni ključ. Dodatno naj bo M tako praštevilo, da ima $M - 1$ same majhne faktorje, ker je v tem primeru dokaj lahko računati logaritme v $\mathbb{Z}_M$. Kot javni ključ objavi seznam logaritmov z osnovo b: $\{a_1 = \log_{b,M} b_1, \ldots, a_n = \log_{b,M} b_n\}$. Pošiljatelj ravna kot pri običajnem MH nahrbtniku: izračuna vsoto $S = \sum_{i=1}^{n} x_i a_i$. Naslovnik pa izračuna $S' = b^S$ in to faktorizira, saj je

$$S' = b^S = \prod_{i=1}^{n} b^{a_i x_i} = \prod_{i=1}^{n} b_i^{x_i} \pmod{M}.$$

Da se celoštevilsko računanje ujema s tistim v $\mathbb{Z}_M$, mora biti $M > \prod_{i=1}^{n} b_i$. Pri tem načinu pa pride do velikega podaljšanja dolžine podatkov. Če vzamemo recimo $n = 100$ in je vsak b_i naključno 100-bitno praštevilo, bi moral biti M dolg okoli 10000 bitov.

Maskiranje z množenjem po modulu služi za pretvorbo med lahko rešljivim in navidezno težko rešljivim nahrbtnikom. Pri tem ni važno, zakaj je prvi nahrbtnik láhko rešljiv. Lahkó da se ga dá na isti način dobiti iz še lažje direktno rešljivega tretjega nahrbtnika, pa tudi tega lahko pretvarjamo naprej... Skratka, z večkratnim množenjem po različnih modulih se malo bolje zakrije struktura nahrbtnika. (Seveda moramo na vsakem koraku paziti, da je trenutno izbrani modul večji od vsote vseh velikosti predmetov na tem koraku.) Tako dobimo *(enkrat ali večkrat) iterirani MH nahrbtnik.*

Priročno je še definirati *gostoto nahrbtnika.* To je razmerje med številom predmetov v ključu in dolžino (v bitih) največjega od teh elementov. Za enostavni MH nahrbtnik je npr. gostota $\approx 1/2$, za množeči pri $n = 100$ pa $\approx 1/100$. (Oznaka $\approx$ pomeni "približno enako".)

3 Shamirjevo razbitje enostavnega MH nahrbtnika

Obdržimo iste oznake kot doslej, dodajmo jim le še $\lesssim$, "manjše ali približno enako". Javni in privatni ključ naj bosta povezana z (1), privatni ključ $\mathbf{b}$ naj bo superrastoče zaporedje. Naj bo $U \equiv W^{-1} \pmod{M}$, $0 < U < M$. Enakost (1) pomeni, da za nek k_j velja

$$a_j U - k_j M = b_{\pi(j)}.$$

To preoblikujemo v

$$\frac{U}{M} - \frac{k_j}{a_j} = \frac{b_{\pi(j)}}{a_j M},$$

kar pomeni, da so vsi k_j/a_j blizu U/M. Sicer napadalec ne pozna U, M, π, k_j in b_j, pozna pa a_j. Privzame se lahko tudi, da je $b_1 \approx 2^n$, ker noben člen superrastočega zaporedja ne sme biti niti premajhen (recimo za $b_1 = 1$ bi se v $\mathbf{a}$ pojavil W) niti prevelik (tak nahrbtnik prostorsko ne bi bil učinkovit). Potem lahko ocenimo $b_n, M \approx 2^{2n}$ in $b_1, b_2, \ldots, b_5 \approx 2^n$. Uvedimo oznako $j_i = \pi^{-1}(i)$ in dobimo, da za $1 \le i \le 5$ velja:

$$\left| \frac{U}{M} - \frac{k_{j_i}}{a_{j_i}} \right| \lesssim \frac{2^n}{2^{4n}} = 2^{-3n}. \tag{2}$$

Nato odštejemo izraz za $i = 1$ od drugih, pa dobimo za $2 \le i \le 5$:

$$\left| \frac{k_{j_i}}{a_{j_i}} - \frac{k_{j_1}}{a_{j_1}} \right| \lesssim 2^{-3n},$$

od koder je

$$|k_{j_i} a_{j_1} - k_{j_1} a_{j_i}| \lesssim 2^n. \tag{3}$$

Ker so a_{j_i} in k_{j_i} reda velikosti 2^{2n}, so produkti $k_{j_{i_1}} a_{j_{i_2}} \approx 2^{4n}$, to pa bi pričakovali tudi za njihove razlike. Ker ni tako, morajo biti a_{j_i} in k_{j_i} prav neobičajni. V večini primerov so k_{j_i} kar enolično določeni z enačbami (3). Shamir ugotavlja, da lahko poiščemo k_{j_i} z Lenstrinim algoritmom za celoštevilsko linearno programiranje. (Ta postopek ima za fiksno število spremenljivk polinomsko časovno zahtevnost.) Čeprav ne poznamo π, pa lahko preverimo vseh $O(n^5)$ (ali $O(n^l)$ za l najmanjših superrastočih vrednosti) možnih izbir, kar nam še vedno doda le polinomski faktor v časovno zahtevnost. Pri napačnih izbirah linearni program zelo verjetno ne bo imel rešitve.

Ko dobimo $k_{j_1}, \ldots, k_{j_5}$, z njihovo pomočjo dobimo približek za U/M, iz tega pa U' in M' tako, da je U'/M' blizu U/M, števila $c_j \equiv U' a_j \pmod{M'}$ pa tvorijo superrastoče zaporedje (ko jih uredimo po velikosti). Ta napad sicer (običajno) ne najde prvotnih U in M, vendar obstaja neskončno parov U', M', ki dajo c_i, s katerimi lahko dešifriramo S. Žal pa je Lenstrin algoritem prepočasen za praktično uporabo. Hitrejše orodje je opisano v naslednjem razdelku.

4 Mreže (lattices) in LLL algoritem

Celoštevilska mreža L je aditivna podgrupa v $\mathbb{Z}^n$, ki vsebuje n linearno neodvisnih vektorjev iz $\mathbb{R}^n$. *Baza* $\mathbf{b}_1, \ldots, \mathbf{b}_n$ *za* L so takšni linearno neodvisni vektorji, da je vsak element iz L njihova linearna kombinacija. Bazo zložimo v $n \times n$ bazno matriko

$$B = \begin{bmatrix} \mathbf{b}_i \\ \vdots \\ \mathbf{b}_n \end{bmatrix}.$$

K dani urejeni bazi za mrežo L pridružimo še ortogonalizirane (seveda pa ne nujno normalizirane) vektorje $\mathbf{b}_i^*$, skupaj z Gramm–Schmidtovimi koeficienti $\mu_{i,j} = \mathbf{b}_i.\mathbf{b}_j^* / (\mathbf{b}_j^*.\mathbf{b}_j^*)$, $1 \le j < i \le n$.

Definicija 1 *([7] navaja [12]) Urejena baza* $\mathbf{b}_1, \ldots, \mathbf{b}_n$ *je* LLL-reducirana, *če obstaja* $\delta \in [1/4, 1)$, *da velja:*

1. $|\mu_{i,j}| \leq 1/2$ *za* $1 \leq j < i \leq n$, *ter*
2. $\delta.||\mathbf{b}^*_{k-1}||^2 \leq ||\mathbf{b}^*_k + \mu_{k,k-1}\mathbf{b}^*_{k-1}||^2$ *za* $k = 2, \ldots, n$.

V reducirani bazi prvi vektor ne more biti zelo dolg. Točneje:

Izrek 1 *([3] in [8] navajata [12]) Naj bo* $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ *reducirana baza n-razsežne mreže L. Potem je*

$$||\mathbf{b}_1||^2 \leq 2^{n-1} \min_{\substack{x \in L \\ x \neq 0}} ||x||^2.$$

V članku [12] je opisan tudi postopek polinomske časovne zahtevnosti, kako dobiti reducirano bazo[2].

(1) (* inicializacija *) Postavi $\beta := 0$

(2) (* Gramm–Schmidtova ortogonalizacija *)
While $\beta \leq n$ do begin $\beta := \beta + 1$;

For $j := 1$ to $\beta - 1$ do $\mu_{\beta,j} := \mathbf{b}_\beta.\mathbf{b}^*_j / B_j$;
$\mathbf{b}^*_\beta := \mathbf{b}_\beta - \sum_{j=1}^{\beta-1} \mu_{\beta j}\mathbf{b}^*_j$
$B_\beta := \mathbf{b}^*_\beta.\mathbf{b}^*_\beta$

end.

$m := 2$;

(3) While $m < \beta$ do begin

For $l := m - 1$ downto 1 do begin
(* reduciramo μ_{ml}, popravimo tudi druge koeficiente *)
If $|\mu_{ml}| < 1/2$ then begin
$t := round(\mu_{ml})$, $\mathbf{b}_m := \mathbf{b}_m - t\mathbf{b}_l$, $\mu_{ml} := \mu_{ml} - t$;
For $j := 1$ to $l - 1$ do $\mu_{mj} := \mu_{mj} - t\mu_{lj}$
end; (* If *)
(* preverimo LLL pogoj na m-tem nivoju *)
If $l = m - 1$ and $(B_m < (\frac{3}{4} - \mu^2_{m,m-1})B_{m-1})$ then goto (4);
end; (* For *)
$m := m + 1$; If $m > \beta$ then Exit.

end; (* While *)

(4) $\mu := \mu_{m,m-1}$; $B := B_m + \mu^2 B_{m-1}$;
$\mu_{m,m-1} := \mu B_{m-1}/B$; $B_m := B_m B_{m-1}/B$;
For $i := m + 1$ to β do $\begin{pmatrix} \mu_{i,m-1} \\ \mu_{i,m} \end{pmatrix} := \begin{pmatrix} 1 & \mu_{m,m-1} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -\mu \end{pmatrix} \begin{pmatrix} \mu_{i,m-1} \\ \mu_{i,m} \end{pmatrix}$;
(* zamenjamo $\mathbf{b}_{m-1}$ in $\mathbf{b}_m$ *)
$B_{m-1} := B$;
For $j := 1$ to $m - 2$ do begin

[2]Ker nisem mogel dobiti [12], sem za lažjo razumljivost poskusil poenostaviti postopek iz [8].

$$\begin{pmatrix} \mathbf{b}_{m-1} \\ \mathbf{b}_m \end{pmatrix} := \begin{pmatrix} \mathbf{b}_m \\ \mathbf{b}_{m-1} \end{pmatrix}; \begin{pmatrix} \mu_{m-1,j} \\ \mu_{m,j} \end{pmatrix} := \begin{pmatrix} \mu_{m,j} \\ \mu_{m-1,j} \end{pmatrix};$$

end;

If $m > 2$ then $m := m - 1$;

goto (3);

Postopek ima polinomsko časovno zahtevnost. Po [5] je najslabša časovna meja $O(n^6 D^3)$, kjer je $D = \lceil \log_2 \max_{1 \le i \le n} a_i \rceil$ največje število bitov v elementu javnega ključa. V praksi pa se menda obnese še bolje od te teoretične meje, in sicer kot $O(nD^3)$.

4.1 LLL napadi na nahrbtnike

Naslednji napad na nahrbtniške sisteme sta odkrila Lagarias in Odlyzko. Poiskati želimo rešitev nahrbtnika z vsoto S in javnim ključem $\{a_i;\ i = 1, \ldots, n\}$. Oblikujmo $(n+1)$-razsežno mrežo z bazo

$$B = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & -a_1 \\ 0 & 1 & 0 & \cdots & 0 & -a_2 \\ & & & \vdots & & \\ 0 & 0 & 0 & \cdots & 1 & -a_n \\ 0 & 0 & 0 & \cdots & 0 & S \end{pmatrix}.$$

Če z $\mathbf{b}_1, \ldots, \mathbf{b}_{n+1}$ označimo vrstice od B in če so x_j rešitve za ta nahrbtnik, potem je v mreži tudi vektor

$$\sum_{j=1}^{n} x_j \mathbf{b}_j + \mathbf{b}_{n+1} = (x_1, \ldots, x_n, 0), \tag{4}$$

ki pa je zelo kratek, saj so lahko x_j le 0 ali 1. Če so a_j zelo veliki, bo večina vektorjev v mreži velikih, (4) pa bo z veliko verjetnostjo najkrajši. Sedaj je treba le pognati postopek LLL na tej bazi, nato pa v dobljeni reducirani bazi preveriti, ali vsebuje vektor oblike (4). Lagarias–Odlyzkov napad se dá, za razliko od Shamirjevega, uporabiti za poljuben nahrbtnik nizke gostote, ker ne uporablja predpostavke, da je skrivni láhko rešljivi nahrbtnik zasnovan na superrastočem zaporedju.

Drugi napad je za iterirane MH nahrbtnike razvil Brickell v [5]. Za Y-krat iteriran MH nahrbtnik si je treba izbrati $m > D/\log(n-1)$, iz javnega ključa m števil $a'_1, \ldots, a'_m$, nato pa reducirati mrežo

$$B = \begin{pmatrix} a'_2 & a'_3 & a'_4 & \cdots & a'_m & n^{-1} \\ a'_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & a'_1 & 0 & \cdots & 0 & 0 \\ & & & \vdots & & \\ 0 & 0 & 0 & \cdots & a'_1 & 0 \end{pmatrix}$$

in v dobljeni reducirani bazi poiskati Y dovolj kratkih vektorjev, takih z dolžino $\lesssim O(2^{D - \log n})$. (Števila a'_i pa moramo izbrati tako, da med njimi ni $a_{\pi(n)}$, $a_{\pi(n-1)}$ ter $a_{\pi(n-2)}$.) Nato dobimo superrastoče zaporedje, s katerim odkodiramo vsoto s pomočjo determinant iz koeficientov zapisa ter kratkih vektorjev v bazi $\mathbf{b}_i$.

Brickellov napad je sicer uporaben le v primerih, ko je $\sum_{i=1}^{n} x_i \le n/2$, kar pa ni ovira, ker lahko v nasprotnem primeru rešujemo problem za vsoto $S'' = \sum_{i=1}^{n} a_i - S$.

5 Chor–Rivestov nahrbtnik

Naslednji nahrbtnik se v pomembni podrobnosti razlikuje od drugih: ni zasnovan na superrastočem zaporedju. Vsako tako zaporedje mora eksponentno hitro rasti, kar precej zmanjša informacijsko gostoto

nahrbtnika. Superrastoče zaporedje pa je bilo uporabljano zato, ker omogoča enolično določitev x_i. Bose in Chowla pa sta pokazala, da to gre tudi s polinomsko rastočimi zaporedji.

Izrek 2 (Bose–Chowla) *([6]) Naj bo p praštevilo in $h \geq 2$ naravno število. Potem obstaja zaporedje $A = \{a_i;\ 0 \leq i \leq p-1\}$, da je:*

(1) $1 \leq a_i \leq p^h - 1,\ i = 0, 1, \ldots, p-1$ *in*

(2) če sta $(x_0, \ldots, x_{p-1})$ *in* $(y_0, \ldots, y_{p-1})$ *različna vektorja z nenegativnimi koordinatami, da je* $\sum_{i=1}^{p-1} x_i \leq h$ *in* $\sum_{i=1}^{p-1} y_i \leq h$, *tedaj je*

$$\sum_{i=1}^{p-1} x_i a_i \neq \sum_{i=1}^{p-1} y_i a_i.$$

Sistem postavimo takole:

1. moramo si izbrati praštevilo p in naravno število tako, da bo $p^h - 1$ imelo relativno majhne prafaktorje (da bomo lahko računali diskretne logaritme v $GF(p^h)$).
2. Naključno izberemo $g, t \in GF(p^h)$ tako, da je g primitiven element v $GF(p^h)$ (ker nam bo služil kot generator), t pa algebraični element stopnje h nad $GF(p) = \mathbb{Z}_p$. (To pomeni: minimalni polinom s koeficienti iz $GF(p)$, ki ima t za koren, je stopnje h.) Naj bo $f(t)$ minimalni polinom za t, $GF(p^h)$ si predstavljajmo kot $GF(p)[t]/\langle f(t)\rangle$.
3. Za vse $\alpha_i \in GF(p)$ izračunamo $a_i := \log(t + \alpha_i)$.
4. Premešamo in dodamo motnjo: $c_i := a_{\pi(i)} + d$, $\pi \in S_n$, $0 \leq d \leq p-2$.

Zdaj imamo javni ključ $\{c_0, c_1, \ldots, c_{p-1}; p, h\}$. Kot privatni ključ obdržimo t, g, π^{-1} in d.

Kodiranje poteka podobno kot pri ostalih nahrbtnikih, z eno razliko: v vektorju $\mathbf{x} = (x_0, \ldots, x_{p-1})$ iz ničel in enic mora biti natanko h enic. Šifrirano sporočilo je torej

$$E(\mathbf{x}) = \sum_{i=0}^{p-1} x_i c_i.$$

Potek dekodiranja:

1. naj bo $r(t) = t^h \pmod{f(t)}$ polinom stopnje $\leq h-1$. (Izračunamo ga lahko enkrat za vselej pri postavitvi sistema.)
2. Za dan $s = E(\mathbf{x})$ izračunamo $s' = s - hd \pmod{p^h - 1}$.
3. Izračunamo $q(t) = g^{s'} \bmod f(t)$, ki je polinom stopnje $h-1$ v spremenljivki t.
4. $s(t) = t^h + q(t) - r(t)$ je polinom stopnje h iz $GF(p)[t]$.
5. Zdaj je $s(t) = (t + \alpha_{i_1}).(t + \alpha_{i_2}).\ldots.(t + \alpha_{i_h})$ produkt linearnih faktorjev, da jih določimo, potrebujemo kvečjemu p substitucij $x = t + \alpha_{i_j}$. Da spet dobimo originalna mesta enic, uporabimo π^{-1}.

(Za popoln postopek manjka še ena malenkost: pretvorba poljubnih dvojiških nizov v bloke dolžine p s točno h enicami.)

V članku [6] Chor in Rivest predlagata $p \approx 200$ in $h \approx 25$ ter ponudita nekaj možnih vrednosti. Za $p = 197$ in $h = 24$ je gostota nahrbtnika približno 1.077, zaradi česar naj bi postopek LLL zašel v težave in praviloma poiskal napačen kratek vektor. Vendar pa sta Schnorr in Hörner ([7]) leta 1995 z izboljšano verzijo LLL algoritma uspela razbiti primer CR nahrbtnika za $p = 103$ in $h = 12$, ki ima še večjo gostoto, tako da ne drži več domneva, da so nahrbtniki z večjo gostoto varnejši.

6 Zaključek

T.i. nahrbtniški kriptosistemi so nekoč precej obetali, danes pa so skoraj vsi razbiti. Pač pa so v polnem razmahu raziskave prav na področju celoštevilskih mrež, s pomočjo katerih so uničili te sisteme. To področje ponuja še dosti zanimivosti, recimo leta 1996 predlagana sistem z javnim ključem in zgoščevalna funkcija, oboje na osnovi celoštevilskih mrež. Zdi se, da se dá za vsak kriptosistem, ki se ga razbije, z dobljenim znanjem sestaviti novega. Verjetno boljšega, ampak to se vidi šele po nekaj letih.

A Primer MH nahrbtnika

A.1 Program v jeziku Mathematica

Sledi nekaj ukazov, narejenih v jeziku Mathematica. Sprogramiral sem prvotni, Merkle–Hellmanov nahrbtnik, za njegovo razbijanje po Lagarias–Odlyzkovi metodi pa sem (namesto da bi sprogramiral postopek iz [8]) uporabil kar v Mathematico že vgrajeni ukaz `LatticeReduce`. Oznake v programu so skladne s tistimi v člankih. Izbrano naključno število shranimo v spremenljivki `test`. Nato generiramo naključno superrastoče zaporedje `a`, v seznam `k` pa damo 3 skrivne podatke: skrivni modul M (kot `k[[1]]`), skrivni faktor W (kot `k[[2]]`) in še njegov inverz po M (`k[[3]]`). V `aa` shranimo javni ključ, ki ga dobimo z maskiranjem z W po modulu M.

```
(* ime datoteke: nahrbt.m *)
(* Merkle - Hellmanovi nahrbtniki *)

(* generiranje nakljucnega superrastocega zaporedja za privatni kljuc *)
Superrast[n_Integer:10]:=Table[
Random[Integer,{(2^(i-1)-1)*2^n+1, 2^(i-1)*2^n}],{i,1,n}];
(* skrivni modul in faktor, m je prastevilo, sicer so vcasih tezave s Solve *)
Skrivni[n_Integer:10]:=Module[{m,w},
 m=4;While[!PrimeQ[m],m=Random[Integer,{2^(2*n+1)+1, 2^(2*n+2)-1}]];
 w=Random[Integer,{2,m-2}];w=w/GCD[m,w];
 winv=Solve[w*x==1 && Modulus==m, {x}][[1,2,2]];
 Return[{m,w,winv}];
];
(* javni kljuc iz privatnega kljuca a *)
Javni[a_List, m_, w_]:= Mod[w*#,m]&/@a;
(* sifriranje - podatek je dvojiski zapis stevila x<2^n *)
sifriraj[x_Integer, aa_List]:=Module[{ost,i,sum},
 ost=x;i=Length[aa];sum=0;While[i>0,
  If[ost>=2^(i-1),
   sum=sum+aa[[i]]; ost=ost-2^(i-1)
  ]; i-- ];
 Return[sum]
];
(* desifriranje - a je privatni kljuc, sum vsota, m modul, winv pa w^{-1} *)
desifriraj[sum_, a_List, m_, winv_]:=Module[{sum1,x,i},
 sum1=Mod[winv*sum, m];
 i=Length[a];x=0;While[i>0,
  If[sum1>=a[[i]],
   x=x+2^(i-1); sum1=sum1-a[[i]]
  ]; i-- ];
 Return[x]
```

```
];

(* nakljucno testiranje *)
test=Random[Integer, {1,100000}];
a=Superrast[20];
k=Skrivni[20];
aa=Javni[a, k[[1]], k[[2]] ];
S=sifriraj[test, aa];
test1=desifriraj[S, a, k[[1]], k[[3]] ];

(* matrika mreze za Lagarias-Odlyzkov napad *)
L=Table[Table[
    If[j==i,1,0],
  {j,1,Length[aa]} ]~Join~{-aa[[i]]},
{i,1,Length[aa]} ]~Join~{Table[0,{i,1,Length[aa]}]~Join~{S}};

(* Lagarias-Odlyzkov napad *)
Lag:=Module[{L1,niz,rezultat},
  L1=LatticeReduce[L];
  niz=L1[[ Position[Union/@L1,{0,1}][[1,1]] ]];
  rezultat=Table[niz[[i]]*2^(i-1),{i,1,Length[niz]-1}]/.{List->Plus};
  Return[rezultat]
];
```

A.2 Primer uporabe

Tu je potek šifriranja in dešifriranja naključno izbranega števila. Najprej v Mathematico naložimo program (iz prejšnjega podrazdelka). Nato pogledamo, kakšna je matrika mreže za Lagarias–Odlyzkov napad in na njej poženemo reducirni postopek. Ogledamo si prvi vektor, pričakujoč, da bo najkrajši, in ker vsebuje le ničle in enke, ga obravnavamo kot seznam binarnih števk nečesa — tisto nekaj pa je ravno enako skrivnemu šifriranemu številu `test`! (In tudi številu `test1`, dobljenem po regularnem dešifrirnem postopku s skrivnim ključem.) Za konec te korake uporabimo vse naenkrat na `L` s funkcijo `Lag`.

```
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

U:\NOVO\Kripto\projekt>math
Mathematica 3.0 for Microsoft Windows
Copyright 1988-97 Wolfram Research, Inc.
License valid through 31 May 2000.
 -- Terminal graphics initialized --

In[1]:= <<nahrbt.m

In[2]:= L

Out[2]= {{1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
>     -239379078612}, {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
>     0, 0, -2366361162238}, {0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
>      0, 0, 0, 0, -3070205250884},
>     {0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
>      -1270540684577}, {0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
>      0, 0, -3021943012191}, {0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
>      0, 0, 0, 0, -196161881115},
>     {0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
>      -3700549578369}, {0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
>      0, 0, -3013309425367}, {0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
>      0, 0, 0, 0, -3363287888155},
>     {0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
>      -1134411855468}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
>      0, 0, -703567273942}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
>      0, 0, 0, 0, -3461453435851},
>     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
>      -1463571070054}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
>      0, 0, -581106394765}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
>      0, 0, 0, 0, -1299570913905},
>     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
>      -197310761603}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
>      0, 0, -3126666565616}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
>      0, 1, 0, 0, -3259826227327},
>     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
>      -489269860081}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
>      0, 1, -3059550220281}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
>      0, 0, 0, 0, 8834333100692}}

In[3]:= %//LatticeReduce

Out[3]= {{0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0},
>     {-1, 2, 1, 0, -2, 1, -1, 1, 0, -2, 1, 2, 1, -1, -1, -2, 0, 1, 0, 0, 0},
>     {1, 1, -1, -1, 0, 2, -2, -2, 3, 1, -3, -1, -1, 1, 1, 0, 1, -1, 0, 0, 0},
>     {4, 0, 0, -1, -3, 1, 0, -3, -1, 1, 0, 2, -3, 1, 0, -1, 1, -1, 0, 0, 0},
>     {2, 0, -1, 1, -2, -3, 1, 1, 2, 1, 3, 0, -1, 0, -1, -2, 0, 1, 0, 0, 0},
>     {-4, -1, 1, -2, -2, 1, 0, 1, -1, 0, -2, -1, -1, -1, 1, 1, 0, -1, 0, 0,
>      0}, {1, -3, -1, 4, -2, 2, -2, 0, 0, 0, -2, 3, -3, 1, 1, -2, 2, -1, 0,
>      0, 0}, {3, 0, -2, 3, 1, 1, -1, 4, -2, -2, 3, -1, -2, 1, 0, -2, 1, 0, 0,
>      0, 0}, {-4, 2, -1, 1, 2, -1, -1, -3, 2, 0, 0, 2, -1, 0, -1, -2, 0, 1,
>      0, 0, 0}, {-1, -3, -1, 2, 0, -2, 4, -1, 0, -2, -1, 3, -3, -1, 0, -2, 0,
>      1, 0, 0, 0}, {-3, 0, 3, -1, 0, 0, 0, 0, 0, 0, 4, -4, 3, -1, 0, 0, 0, 0,
>      0, 0, 0}, {0, 0, 0, -2, -1, -1, 1, 0, -2, -3, 2, 4, -2, 2, -1, 0, 0, 0,
>      0, 0, 0}, {-1, 1, 0, -1, 0, -2, -2, -1, -1, 1, -2, 1, 0, 2, 0, 2, 2,
>      -2, 0, 0, 0}, {2, -1, 0, 2, -2, 0, -2, 0, 0, 2, -1, 0, -2, 3, 1, -2, 1,
>      -3, 1, 0, 0}, {-1, -2, 1, 1, -1, 0, -1, 0, 2, -1, 0, 0, -2, -1, 2, 0,
>      1, -3, 1, 0, 0}, {0, -1, -1, 1, -3, -1, 0, 0, 2, 1, -3, 1, -2, -1, 0,
>      -1, -2, 1, -2, 1, 0}, {0, 0, -1, 1, 1, 0, -2, 2, 0, 0, 0, 2, 3, 0, 1,
>      0, 1, 0, -2, 1, 0}, {0, -1, 0, 1, 0, -1, -1, -1, -6, 1, 1, 0, -2, -1,
>      2, 1, 1, 0, 1, -1, 0}, {-1, 3, 1, -1, 1, 1, -2, 0, 1, 0, 0, -2, 2, -1,
>      0, 1, 0, -2, 0, 1, -3}, {1, 0, -3, -1, 1, 0, -1, -1, -1, 0, -1, -1, -1,
>      2, 1, -1, -1, 0, 0, 2, 2},
>     {0, -2, -1, 2, -1, 0, 1, 0, -2, -1, 0, 4, 1, 2, 0, 0, -2, 0, -1, -2, 1}}
```

```
In[4]:= %[[1]]

Out[4]= {0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0}

In[5]:= Table[%[[i]]*2^(i-1),{i,1,Length[%]}]/.{List->Plus}

Out[5]= 49226

In[6]:= test

Out[6]= 49226

In[7]:= test1

Out[7]= 49226

In[8]:= Lag

Out[8]= 49226
```

B Primer iteriranega MH nahrbtnika

B.1 Program v jeziku Mathematica

Sledi dodatek k prejšnjemu programu, ki implementira iterirane MH nahrbtnike. Posebna procedura za večkratno maskiranje skrivnega ključa ni potrebna, dovolj je večkrat uporabiti tisto za (enojno) maskiranje iz `nahrbt.m`, pač pa sem napisal `desifriraj1` za dešifriranje večkrat maskiranega sporočila. Tu sta še funkciji za pretvorbo iz desetiškega v dvojiški sistem in nazaj, če bi koga zanimala struktura šifriranih zaporedij, in pa malce spremenjena funkcija `Skrivni`, ki ji podamo seznam velikosti predmetov, da poišče modul, ki je večji od njihove vsote. Matriko L zdaj sestavimo s funkcijo `LagariasMatrix[aa,S]`, uporabljeno v (sicer nespremenjenem) napadu `Lag2`.

```
(* ime datoteke: nahrbt2.m *)
(* najprej nalozimo funkcije za enojne MH nahrbtnike *)
<<nahrbt.m;

(* popravljena funkcija Skrivni[a] pazi, da je modul vecji od vsote *)
(* vseh elementov podanega zaporedja a *)
Skrivni[a_List]:=Module[{m,w,vsota},
 vsota=Plus@@a;
 m=4;While[!PrimeQ[m],m=Random[Integer,{vsota+1, 2*vsota}]];
 w=Random[Integer,{2,m-2}];w=w/GCD[m,w];
 winv=Solve[w*x==1 && Modulus==m, {x}][[1,2,2]];
 Return[{m,w,winv}];
];
(* funkciji desifriraj1 moramo podati seznama modulov in inverzov,      *)
(* in to v takem vrstnem redu, kot smo jih uporabili za izdelavo kljuca *)
(* seznama m in winv morata seveda biti enako dolga                     *)
desifriraj1[sum_, a_List, m_List, winv_List]:=Module[{sum1,x,i},
 i=Length[m];x=sum;
 While[i>0,
```

```
   sum1=Mod[x*winv[[i]], m[[i]] ];x=sum1;
   i-- ];
  sum1=x;
  i=Length[a];x=0;While[i>0,
   If[sum1>=a[[i]],
    x=x+2^(i-1); sum1=sum1-a[[i]]
   ]; i-- ];
  Return[x]
];
(* sledita se dve pomozni funkciji za delo z binarnim zapisom stevil:  *)
(* funkcija b vrne seznam nicel in enic iz binarnega zapisa argumenta, *)
(* ce je treba, podaljsanega z niclami na dolzino 30 znakov            *)
b[x_]:=Module[{i,a},
 i=x;a={};
 While[i>0,
  a=a~Join~{If[EvenQ[i],0,1]};
  i=Floor[i/2];
];Return[a~Join~Table[0,{i,1,Max[30-Length[a],0]}] ] ];
(* funkcija d vrne stevilo x, ce ji podamo seznam b[x] *)
d[a_List]:=a.Table[2^i,{i,0,Length[a]-1}];

(* sledi testiranje *)
test=Random[Integer, {1,500}];
Print["Izbrano stevilo: ", test];
a1=Superrast[10];                    (* skrivni kljuc *)
k1=Skrivni[a1];          (* modul, faktor in inverz za prvo maskiranje *)
aa1=Javni[a1, k1[[1]], k1[[2]] ];    (* vmesni kljuc *)
k2=Skrivni[aa1];          (* modul, faktor in inverz za drugo maskiranje *)
aa2=Javni[aa1, k2[[1]], k2[[2]] ];   (* javni kljuc *)

(* preizkusimo, ali dvakrat maskirani kljuc dela v redu *)
test0=desifriraj1[sifriraj[test,aa2], a1,
 {k1[[1]], k2[[1]]}, {k1[[3]], k2[[3]]}];
Print["Po kodiranju in dekodiranju: ", test0];

(* matrika mreze za Lagarias-Odlyzkov napad *)
LagariasMatrix[aa_List,S_Integer]:=Module[{},Return[Table[Table[
    If[j==i,1,0],
  {j,1,Length[aa]} ]~Join~{-aa[[i]]},
{i,1,Length[aa]} ]~Join~{Table[0,{i,1,Length[aa]}]~Join~{S}}]];

(* Lagarias-Odlyzkov napad *)
Lag2:=Module[{L1,niz,rezultat},
  L1=LatticeReduce[LagariasMatrix[aa2,sifriraj[test,aa2]]];
  niz=L1[[ Position[Union/@L1,{0,1}][[1,1]] ]];
  rezultat=Table[niz[[i]]*2^(i-1),{i,1,Length[niz]-1}]/.{List->Plus};
  Return[rezultat]
];
```

B.2 Primer uporabe

```
U:\NOVO\Kripto\p>math
Mathematica 4.0 for Microsoft Windows
Copyright 1988-1999 Wolfram Research, Inc.
 -- Terminal graphics initialized --

In[1]:= <<nahrbt2.m
Izbrano stevilo: 453
Po kodiranju in dekodiranju: 453

In[2]:= Lag2

Out[2]= 453

In[3]:= <<nahrbt2.m
Izbrano stevilo: 419
Po kodiranju in dekodiranju: 419

In[4]:= Lag2

Out[4]= 419

In[5]:= <<nahrbt2.m
Izbrano stevilo: 223
Po kodiranju in dekodiranju: 223

In[6]:= Lag2

Out[6]= 223
```

Lagarias–Odlyzkov napad se, kot kaže, obnese tudi tokrat.

B.3 Zaključek glede iteriranih MH nahrbtnikov

Ker se pri konstrukciji iteriranih MH nahrbtnikov informacijska gostota zmanjšuje, so ti še bolj občutljivi na Lagarias–Odlyzkov napad kot pa enojni MH nahrbtniki. Po drugi strani se prav z večanjem informacijske gostote daljšajo ključi, s tem pa postaja uporaba nerodnejša. Zato so ti sistemi zanimivi le še zgodovinsko.

Literatura

Splošno:

[1] Douglas R. Stinson. *Cryptography: Theory and Practice.* CRC Press, Boca Raton - New York - London - Tokyo, 1995.

Članki, uporabljeni pri pisanju seminarja:

[2] R. C. Merkle, M. E. Hellman. *Hiding information and signatures in trapdoor knapsacks.* IEEE Trans. Info. Theory **IT-24** (1978), 525-530.

[3] A. M. Odlyzko. *The rise and fall of knapsack cryptosystems.* Proc. Symposia in Applied Mathzematics, vol. 42 (1990), 75-88.

[4] Adi Shamir. *A polynomial-time algorithm for breaking the basic Merkle–Hellman cryptosystem.* IEEE Trans. Info. Theory, **IT-30** (1984), 699-704.

[5] E. F. Brickell. Breaking iterated knapsacks. Advances in cryptology (Proc. Crypto 84) (Lecture notes in computer science 218), Springer-Verlag, Berlin, 1985, 342-358.

[6] B. Chor, R. L. Rivest. *A knapsack-type public key cryptosystem based on arithmetic in finite fields.* IEE Trans. Info. Theory, **IT-34**(1988), 901-909.

[7] C. P. Schnorr, H. H. Hörner. *Attacking the Chor–Rivest cryptosystem by improved lattice reduction.* Advances in Cryptology (Eurocrypt '95) (Lecture notes in computer science 921), Springer-Verlag, 1995, pp. 1-12.

[8] M. Pohst. *A modification of the LLL reduction algorithm.* J. Symbolic Computation (1987) **4**, 123-127.

Viri na internetu:

[9] *http://www.ieor.berkeley.edu/courses/archives/ieor215/spring97/knapsack/index.html*
(projekt Remote Interactive Knapsack Allocation, avtorji Vincent Lin, Kengyi Kyle Lin in Chun-Yi Johhny Wu, mentor Ken Goldberg)

[10] *http://www.rsa.com/rsalabs/faq/html/3-6-8.html*
(nekaj zgodovine kriptosistemov z javnim ključem)

[11] *http://www.bertaut.com/cryptography.html*
(avtor zhang@metrocall.com trdi, da njegov sistem odpravi določene slabosti običajnih nahrbtnikov)

Le posredne reference:

[12] A. K. Lenstra, H. W. Lenstra Jr., L. Lovász, *Factoring polynomials with rational coefficients.* Math. Ann. **261** (1982), 515-534.

[13] G. Brassard. *A note on the complexity of cryptography.* IEEE Trans. Info. Theory, **IT-25** (1979), 232-233.