Лекция 11

Криптосистеми, използващи задача за раницата

11.1 Криптосистема на Merkle-Hellman

През 1978 г. Меркл и Хелман предлагат асиметрична криптосистема, основаваща се на трудността на задачата за раницата [44]. Скоро след това се появяват редица модификации и подобрения на тази система. Тук ние ще опишем класическия вариант на криптосистемата на Меркл и Хелман.

Нека $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$, $a_i \in \mathbb{N}$ е n-мерен вектор и нека α е цяло положително число. Задача за раницата с вход двойката $(\boldsymbol{a}, \alpha)$, или накратко $KNAPSACK(\boldsymbol{a}, \alpha)$, наричаме следната задача:

$KNAPSACK(\boldsymbol{a}, \alpha)$:
Намерете решение на уравнението:

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n = \alpha, \tag{11.1}$$

за което $x_i \in \{0, 1\}$ при условие, че такова съществува.

Друг вариант на горната задача е да се определи дали (11.1) има решение. И двата варианта са NP-пълни задачи. Ще отбележим, че съществуват варианти на задачата за раницата, които не са дори в NP. Разбира се, намирането на $(0, 1)$-решение на (11.1) е поне толкова трудно, колкото и доказването на съществуване на решение. Основната идея на Меркл и Хелман е да използват вектора $\boldsymbol{a}$ за шифриране на блокове от n бита $\boldsymbol{c} = (c_1, c_2, \ldots, c_n)$, $c_i \in \{0, 1\}$, чрез пресмятане на

$$\alpha = \boldsymbol{a}\boldsymbol{c}^t = \sum_{i=1}^{n} a_i c_i.$$

Дешифрирането се състои във възстановяването на вектора $\boldsymbol{a}$ от α и $\boldsymbol{a}$. Тъй като дешифрирането трябва да е еднозначно, то е необходимо да разглежадме задачи

$KNAPSACK(\boldsymbol{a}, \alpha)$, които имат най-много едно решение за произволно $\alpha \in \mathbb{N}$. Вектори $\boldsymbol{a}$, за които това е изпълнено, се наричат *инективни*.

Пример 11.1. Векторът $\boldsymbol{a} = (3, 41, 5, 1, 21, 10)$ е инективен, докато

$$\boldsymbol{a}' = (14, 28, 56, 82, 90, 132, 197, 284, 341, 455)$$

не е.

Съществуват вектори $\boldsymbol{a}$, за които задачата $KNAPSACK(\boldsymbol{a}, \alpha)$ е лесна за всяко $\boldsymbol{a}$. Един вектор $\boldsymbol{a}$ наричаме *свръхнарастващ*, ако

$$a_j > \sum_{i=1}^{j-1} a_i, \quad j = 2, 3, \ldots, n.$$

Нека е даден свръхнарастващ. Можем да конструираме очевиден greedy-алгоритъм за решаване на $KNAPSACK(\boldsymbol{a}, \alpha)$. Най-напред проверяваме дали $\alpha \geq a_n$. Ако отговорът е "не", то $c_n = 0$; ако отговорът е "да", то $c_n = 1$ и полагаме

$$\alpha^{(1)} = \begin{cases} \alpha, & \text{ako } \alpha < a_n; \\ \alpha - a_n, & \text{ako } \alpha \geq a_n. \end{cases}$$

По-нататък извършваме същата процедура за $\alpha^{(1)}$ и a_{n-1}. Алгоритъмът завършва, когато извършим описаната стъпка за $\alpha^{(n-1)}$ и a_1. Описаната процедура доказва също, че ако $\boldsymbol{a}$ е свръхнарастващ, тоза всяко α задачата $KNAPSACK(\boldsymbol{a}, \alpha)$ има най-много едно решение.

$KNAPSACK(\boldsymbol{a}, \alpha)$ за свръхнарастващ вектор $\boldsymbol{a}$.

```
1) i=n;
2) while (i>=1)
3)      if (alpha>=a[i]) then x[i]=1;
4)                           alpha=alpha-a[i];
5)                       else x[i]=0;
6)      j=j-1;
7) if (alpha==0) then print(‘‘there exists a solution’’);
8)               else print(‘‘a solution does not exist’’);
```

При криптосистеми с публичен ключ, използващи свръхнарастващ вектор $\boldsymbol{a}$, самият вектор $\boldsymbol{a}$ трябва да бъде разглеждан като част от тайния ключ. В противен случай дешифрирането би било еднакво лесно както за законния получател, така и за криптаналиста. Затова $\boldsymbol{a}$ бива преобразуван така, че да изглежда като случаен вектор. Да дефинираме:

$$\max \boldsymbol{a} = \max\{a_j \mid 1 \leq j \leq n\};$$

$$(x \mod m) = x - \lfloor \frac{x}{m} \rfloor m, \; x \in \mathbb{Z}, m \geq 2.$$

Да разгледаме свръхнарастващ вектор $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$, цяло число m $\max(\boldsymbol{a})$, и естествено число t, $1 \leq t \leq m$, $(t, m) = 1$. Казваме, че векторът $\boldsymbol{b} = (b_1, b_2, \ldots, b_n)$ е

получен от $\boldsymbol{a}$ чрез *слабо умножение с* $\boldsymbol{t}$ *по модул* m, ако $b_i = ta_i \mod m$, $i = 1, \ldots, n$. Условието $(t, m) = 1$ гарантира съществуването на $t^{-1} \mod m$, т.е. на такъв елемент u, за който $tu \equiv 1 \mod m$. Следователно векторът $\boldsymbol{a}$ също се получава от $\boldsymbol{b}$ чрез слабо умножение с u по модул m.

Ако условието $m \geq \max \boldsymbol{a}$ бъде заменено с $m > \sum_{i=1}^{n} a_i$, казваме, че $\boldsymbol{b}$ се получава от $\boldsymbol{a}$ чрез *силно умножение с* t *по модул* m. В този случай не м,ожем да заключим, че и $\boldsymbol{a}$ се получава от $\boldsymbol{b}$ чрез силно умножение по модул m.[1]

За да изгради криптоситема потребителят U избира:

- свръхнарастващ вектор $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$;
- цяло число $m > \sum_{i=1}^{n} a_i$;
- t, $1 < t < m, (t, m) = 1$

и пресмята

- $\boldsymbol{b} = (b_1, b_2, \ldots, b_n)$, $b_i = ta_i \mod m$,

получен от $\boldsymbol{a}$ рчез силно умножение по модул m. Векторът $\boldsymbol{b}$ се публикува като шифриращ (публичен) ключ, а $\boldsymbol{a}, m, t$ се пазят в тайна и съставляват дешифриращия (тайния) ключ.

Шифрирането се извършва по следния начин. Откритият текст се разбива на блокове (вектори) над азбуката $\{0, 1\}$ с дължина n. [2] Нека $\boldsymbol{\mu} = (m_1, m_2, \ldots, m_n)$ е такъв блок. Криптотекстът β, съответстващ на $\boldsymbol{mu}$, се получава като скаларно произведение на $\boldsymbol{b}$ и $\boldsymbol{\mu}$:

$$\beta = \boldsymbol{b} \cdot \boldsymbol{\mu}^t.$$

При дешифрирането законният получател, притежаващ $\boldsymbol{a}$, t и m (а следователно и $u = t^{-1} \mod m$) пресмята $\alpha = u\beta \mod m$ и решава задачата $KNAPSACK(\boldsymbol{a}, \alpha)$, която е лесна. От своя страна криптаналистът е изправен пред задачата $KNAPSACK(\boldsymbol{b}, \beta)$, в която $\boldsymbol{b}$ изглежда като случаен вектор.

В [44] авторите на системата препоръчват следните параметри:

- $n \approx 100$;
- $(2^i - 1) \cdot 2^{100} \leq a_i < 2^i \cdot 2^{100}$, $1 \leq i \leq 100$;
- $2^{201} + 1 < m < 2^{202}$;

с което векторът $\boldsymbol{a}$ автоматично е свръх нарастващ.

Забележка 11.2. 1) Една добра идея е публикуването на компонентите на вектора $\boldsymbol{b}$ в разбъркан ред:

$$(b_{\pi(1)}, b_{\pi(2)}, \ldots, b_{\pi(n)}),$$

където $\pi \in S_n$ е произволна (тайна) пермутация. По този начин криптаналистът не знае, кое b_i в публичния вектор е получено от най-малката компонента a_1 на $\boldsymbol{a}$.

[1] Разбира се $\boldsymbol{a}$ винаги може да се получи чрзе слабо умножение снякакво u.

[2] За да получим открития текст като последователност от нули и единици можем да използваме произволно кодиране. То няма отношение към сигурността на системата.

2) Целта при умножаването на свръхнарастващия вектор $\boldsymbol{a}$ с t по модул m е получаването на вектор $\boldsymbol{b}$, който *изглежда* случаен. За усилване на този ефект Меркл и Хелман предлагат итериране на това умножение. Без да навлизаме в подробности ще отбележим, че итерирането на силно умножение по модул не води задължително до усилване на сигурността на системата. За криптаналиста може да се окаже полезно да изобрази публичния вектор $\boldsymbol{b}$ върху свръхнарастващ вектор $\boldsymbol{a}'$, различен от оригиналния, което да доведе до валиден криптанализ.

В следващата теорема е доказана коректността на криптосистемата на Меркл и Хелман.

Теорема 11.3. *Нека $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$ е свръхнарастващ вектор, а $\boldsymbol{b}$ се получава от $\boldsymbol{a}$ чрзе силно умножение ct по модул m. Нека β е произволно цяло число, $u \equiv t^{-1} (\mod m)$ и $\alpha = u\beta \mod m$. Тогава:*

(*i*) $KNAPSACK(\boldsymbol{a}, \alpha)$ е разрешима за линейно време; ако решение съществува, то то е единствено;

(*ii*) $KNAPSACK(\boldsymbol{b}, \beta)$ има най-много едно решение;

(*iii*) ако съществува решение за $KNAPSACK(\boldsymbol{b}, \beta)$, то то съвпада с единственото решрение на $KNAPSACK(\boldsymbol{a}, \alpha)$.

Доказателство. (*i*) Тази част следва от алгоритъма за решаване на задача за раницата при свръхнарастващи вектори.

(*ii*), (*iii*) Да допуснем, че векторът $\boldsymbol{x}$ е решение на $KNAPSACK(\boldsymbol{b}, \beta)$, т.е. $\boldsymbol{b}\boldsymbol{x}^t = \beta$. Оттук следва, че

$$\alpha \equiv u\beta = u\boldsymbol{b}\boldsymbol{x}^t \equiv u(t\boldsymbol{a})\boldsymbol{x}^t \equiv \boldsymbol{a}\boldsymbol{x}^t (\mod m).$$

Тъй като $m > \sum_{i=1}^{n} a_i$ в сила е и неравенството $m > \boldsymbol{a}\boldsymbol{x}^t$. По дефиниция, $\alpha < m$, откъдето $\alpha = \boldsymbol{a}\boldsymbol{x}^t$. Следователно $\boldsymbol{x}$ съвпада с единственото решение на $KNAPSACK(\boldsymbol{a}, \alpha)$. Тъй като започнахме с произволно решение $\boldsymbol{x}$ на $KNAPSACK(\boldsymbol{b}, \beta)$ и доказахме, че то съвпада с единственото решение на $KNAPSACK(\boldsymbol{a}, \alpha)$, то оттук следва и (*ii*). □

Пример 11.4. Нека $n = 10$ и да разгледаме свръхнарастващия вектор

$$\boldsymbol{a} = (103, 107, 211, 430, 863, 1718, 3449, 6907, 13807, 27610).$$

Нека изберем $m = 55207 = \sum a_i + 2$, $t = 25236$; тогава $u = t^{-1} = 1061$. След силно умножение на $\boldsymbol{a}$ с t по модул m получаваме

$$\boldsymbol{b} = (4579, 50316, 24924, 30908, 27110, 17953, 32732, 16553, 22075, 53620).$$

Да шифрираме открития текст `CRYPTOGRAPHY`. [3] В резултат получаваме:

`CR`	→	00011	10010	→	$30908 + 27110 + 17953 + 22075$	=	42837
`YP`	→	11001	10000	→	$4579 + 50316 + 27110 + 17953$	=	99958
`TO`	→	10100	01111	→	$4579 + 24924 + 32732 + 16553 + 22075 + 53620$	=	154483
`GR`	→	00111	10010	→	$24924 + 30908 + 27110 + 17953 + 22075$	=	122970
`AP`	→	00001	10000	→	$27110 + 17953$	=	45063
`HY`	→	01000	11001	→	$50316 + 17953 + 32732 + 53620$	=	154621

[3] Ще използваме кодирането `A`→ (00001), `B`→ (00010),`C`→ (00011), ..., `Z`→ (11010).

Така криптотекстът е

42837 99958 154483 122970 45063 154621.

Пример 11.5. Пример за дешифриране (to be written)

Редица изследователи изпитват съмненя по отношение на сигурността на криптосистемата на Меркл-Хелман. Причината за тях се корени най-вече в линейността на шифриращото преобразуване. В следващия раздел ще опишем една атака, предложена от A. Shamir срещу криптосистемата на Меркл-Хелман.

11.2 Криптанализ на Shamir

Нека е даден векторът $\boldsymbol{b}$, който се използва като публичен ключ в криптосистемата, описана по-горе.Нека е известно ,че $\boldsymbol{b}$ е получен от свръхнарастващия вектор $\boldsymbol{a}$ чрез силно умножение по модул m с множител t. Векторът $\boldsymbol{a}$, какрто и числата m и t са неизвестни. За дешифриране на произволен криптотекст са необходими m и $u = t^{-1}(\mathrm{mod} m)$. Ще отбележим, че пресмятането на u при зададено t може да бъде извършено бързо с помощта на алгоритъма на Евклид. Ще отбележим още, че описаният криптанализ се извършва при зададен ключ за шифриране, т.е. криптаналистът разполага с повече време, тъй като всички пресмятания могат да бъдат извършени преди изпращането на важни криптотекстове.

Сега ще изложим един метод за криптанализ, предложен от Шамир [55]. Ще започнем с фиксирането на ограничения за параметрите на системата, която ще атакуваме. Ще считаме, че

- $d > 1$ е константа;
- модулът m е с дължина dn бита;
- компонентите a_i, $1 \leq i \leq n$, на свръхнарастващия вектор $\boldsymbol{a}$ се състоят съответно от $dn - 1 - n + i = dn - (n - i + 1)$ бита;[4] водещият бит е винаги 1.

Тези ограничения гарантират, че $\boldsymbol{a}$ е винаги свръхнарастващ и че m може да бъде избран да удволетворява условието $m > \sum a_i$. В [44] са предложени параметрите $n = 100$, $d = 2$. Това означава, че m е с дължина 200 бита и дължината на компонентите $a_1, a_2, \ldots, a_{100}$ нараства от 100 до 199 бита.

Да започнем с това, че не е необходимо да намерим множителя u и модула m, реално използвани при шифриране за да компрометираме системата на меркл-Хелман. До правилно дешифриране води всяка двойка (u, m), отговаряща на условията:

- $\boldsymbol{b}$ се получава от $\boldsymbol{a}$ чрез силно умножение с множител $t = u^{-1}$ и модул m;
- $\boldsymbol{a}$ е свръхнарастващ;
- m надхвърля сумата от компонентите на $\boldsymbol{a}$.

[4] ако $d \notin \mathbb{Z}$, то dn се замества с $\lfloor dn \rfloor$

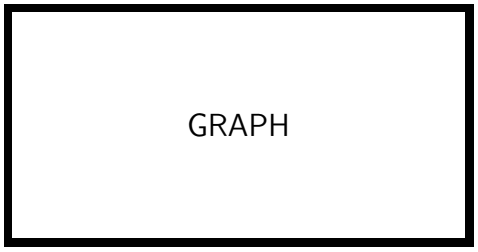

Figure 11.1: Това е графиката.

Ако успеем да конструираме такава двойка (trapdoor pair), то можем да дешифрираме съгласно Теорема 11.3. Очевидно съществува поне една подходяща двойка (u, m).

За да конструираме двойка (u, m), отговаряща на горните изисквания, разглеждаме графиките на функциите

$$f_i(u) = b_i u (\mathrm{mod}\ m), \quad i = 1, 2, \ldots, n.$$

Всяка от тях се с ъстои от части от успоредни прави като стойностите $u = jm/b_i$, $j = 1, 2, \ldots, b_i$, са точките на прекъсване.

Нека u_0 е актуално използваната стойност на uв криптосистема на Меркл-Хелман. Тъй като $a_1 = b_1 u_0 \mod m$ е първата компонента в свръхнарастващ вектор $\boldsymbol{a}$ и $m > \sum a_i$, то тя е много малка в сравнение с m. Това означава, че стойността на u_0 трябва да е близо до някой минимум на f_1. Аналогично, u_0 трябва да се намира близо до минимум на f_2, откъдето правим извода, че въпросните минимуми на f_1 и f_2 са близо един до друг. Можем да продължим с други функции f_i и да получим, че u_0 е близо до минимум на всяка от тях. Така вместо да търсим актуално използваното $u = u_0$, ние търсим "точки на натрупване" на минимуми за нашите криви. Така криптанализът се свежда до построяване на малък интервал, съдържащ минимумите на всяка крива. От построения интервал можем да определим и подходяща стойност на u. Евристични аргументи [55] показват, че при $d = 2$ е достатъчно да разгледаме графиките на 4 функции f_i за да получим обозримо множество от точки на натрупване на минимуми.

Да конкретизираме изложените по-горе идеи. Тъй като модулът m е неизвестен намаляваме мащаба на графиката на фигура X m пъти. Тази операция не засяга взаимното местоположение на точките на натрупване. Алгоритъмът за намиране на подходяща двойка (u, m) се състои от две части.

В първата част намираме кандидати за цяло число j имащо свойството, че j-тия минимум на f_1 е точка на натрупване. Във втората част се проверява дали намерените кандидати водят до построяване на подходяща двойка (u, m). Поне една от проверките трябва да бъде успешна, тъй като стойността на реално използваното u определя точка на натрупване.

Забележка 11.6. Първата част на алгоритъма може да даде много (в сравнение с размера на задачата) кандидати за j. В такъв случай фиксираме параметър R,

задаващ максималния брой разрешени кандидати. Ако алгоритъмът произведе $R+1$ кандидати за j, то той съобщава грешка и преустановява работа.

В първата част не е необходимо да разглеждаме графиките всички на функции $f_2, \ldots, f_n$. Разумно е да фиксираме $s < n$ и да разгледаме само $F_2, \ldots, f_s$. С други думи в първата част на алгоритъма генерираме такива цели числа j, за които j-тия минимум на f_1 е близо до някакъв минимум на $f_2, \ldots, f_s$. Графиките на f_k, $k > s$ не се разглеждат и е вероятно да бъдат получени съвсем погрешни стойности за j. Вече отбелязахме, че $s = 4$ е в повечето случаи разумно ограничение. Във втората си част алгоритъм ът проверява всички стойности всички стойности на i, $2 \leq i \leq n$. Един кандидат p се отхвърля, ако за някое i не съществува минимум на f_i близо до p-тия минимум на f_1.

Да разгледаме детайлно първата част на алгоритъма. Ясно е, че u-координатата на j-тия минимум на f_1 е j/b_1. Условието минимум на f_2 да лежи близо до j-тия минимум на f_1 се изразява чрез неравенствата

$$-\varepsilon < \frac{j}{b_1} - \frac{k}{b_2} < \varepsilon, \;\; 1 \leq j \leq b_1 - 1, 1 \leq k \leq b_2 - 1, \tag{11.2}$$

или $\delta < b_2 j - b_1 k < \delta$. Можем да напишем $s-1$ неравенства от този вид за всяка от функциите $f_2, \ldots, f_s$. По-нататък ще коментираме как следва да се избира δ. Тази стъпка завършва с получаване на списък от цели числа j, за които могат да бъдат намерени цели числа $k, \ldots$, за които са удовлетворени всички $k-1$ неравенства от вида 11.2.

Втората част на алгоритъма проверjава всички получени j до намиране на подходяща двойка (u, m). Нека разгледаме фиксирано j. Всички точки на прекъсване на n-те криви в затворения интервал $[j/b_1, (j+1)/b_1]$ се сорират в нарастващ ред. Нека x_ℓ и $x_{\ell+1}$ са две последователни точки в сортирания списък. Тогава в интервала $[x_\ell, x_{\ell+1}]$ всяка от кривите f_i е права линия и може да се представи аналитично във вида

$$f_i = b_i u - c_i^{(\ell)},$$

където $c_i^{(\ell)}$ е константа, зависеща от i, ℓ и j. Да разгледаме следните линейни неравенства по отношение на u [5]:

$$\begin{aligned} & x_\ell \leq u \leq x_{\ell+1} \\ & \textstyle\sum_{i=1}^{n} (b_i u - c_i^{(\ell)}) < 1 \\ & (b_1 u - c_1^{(\ell)}) + \ldots + (b_{i-1} u - c_{i-1}^{(\ell)}) < (b_i u - c_i^{(\ell)}), \; i = 1, \ldots, n-1. \end{aligned} \tag{11.3}$$

Множеството от решения на тези неравенства е отворен подинтервал (възможно празен) на $[x_\ell, x_{\ell+1}]$. Необходимо и достатъчно условие числата u и m да образуват подходяща двойка е u/m да принадлежи на така получения подинтервал. Описаното изследване се извършва за всички двойки (j, ℓ), където j е кандидат, генериран в първата част, а ℓ е индекс на точка в сортирания списък, съответстващ на j Работата се прекратява при намиране на непразен подинтервал-решение на $[x_\ell, x_{\ell+1}]$.

[5] Първото неравенство отразява принадлежността на u на интервала $[x_\ell, x_{\ell+1}]$; второто неравенство е условието $\sum a_i < m$; последните $n-1$ неравенства гарантират, че векторът $\boldsymbol{a}$ е свръхнарастващ.

Забележка 11.7. 1) В първата си част алгоритъмът генерира кандидати j за понататъшно изследване. Той може да бъде разглеждан като задача на целочисленото програмиране. Втората част се свежда до намирането на рационално число n/m, принадлежащо на някакъв интервал; това е задача от диофантови апроксимации. И двете задачи изискват полиномиално време (от n).

2) Да припомним, че алгоритъмът съобщава грешка, ако първата част генерира повече от R кандидати. Неравенствата в тази част използват и предварително фиксирана граница δ. В [54] е пресметнато, че ако изберем $\delta < \sqrt{b_1/2}$, то вероятността за съобщаване на грешка поради намиране на много кандидати за j не надхвърля $(2/R)^{s-1}$.

3) Степента на полинома, задаващ времето за изпълнение, зависи по сложен начин от избраните константи δ, R, s.

Пример 11.8. Нека публикуваният вектор е $\boldsymbol{b} = (7, 3, 2)$. В този случай е възможно да използваме директни пресмятания; лесно се забелязва, че $\boldsymbol{b}$ е дори свръхнарастващ в обратен ред. Все пак ще се придържаме към описания алгоритъм.

1. стъпка. Съществуват две двойни неравенства:

$$\begin{aligned} &-\delta < 3j - 7k < \delta, \quad 1 \le j \le 6, 1 \le k \le 2, \\ &-\delta < 2j - 7\ell < \delta, \quad \ell = 1. \end{aligned} \tag{11.4}$$

Избираме $\delta = \sqrt{b_1/2} \approx 1.87$. Този избор не дава целочислени решения за j, тъй като избраните параметри са малки. Затова във втората част на алгоритъма ние ще проверим всички възможности за j.

2. стъпка. Приемаме всички $j \in \{1, 2, \ldots, 6\}$ за кандидати и разделяме интервала $(0, 1)$ на подинтервали, във всеки от които графиките на функциите f_1, f_2, f_3 представляват част от (единствена) права линия:

$$\left(0, \frac{1}{7}\right), \left(\frac{1}{7}, \frac{2}{7}\right), \left(\frac{2}{7}, \frac{1}{3}\right), \left(\frac{1}{3}, \frac{3}{7}\right), \left(\frac{3}{7}, \frac{1}{2}\right), \left(\frac{1}{2}, \frac{4}{7}\right), \left(\frac{4}{7}, \frac{2}{3}\right), \left(\frac{2}{3}, \frac{5}{7}\right), \left(\frac{5}{7}, \frac{6}{7}\right), \left(\frac{6}{7}, 1\right).$$

Във всеки от тези интервали наравенствата 11.5 имат вида:

$$\begin{aligned} (7u - i') + (3u - i'') + (2u - i'') &< 1 \\ 7u - i' &< 3u - i'' \\ (7u - i') + (3u - i'') &< 2u - i''', \end{aligned} \tag{11.5}$$

където $0 \le i' \le 6$, $0 \le i'' \le 2$, $0 \le i''' \le 1$, в зависимост от подинтервала. Тези неравенства могат да бъдат записани като

$$12u < i, \ 4u < j, \ 8u < k,$$

където $i = i' + i'' + i'''$, $j = i' - i''$, $k = i' + i'' - i'''$. В таблицата по-долу са представени стойностите на константите в различните подинтервали заедно с индикация дали съответните неравенства са удовлетворени в целия интервал (Y), дали са удовлетворени в част от него $(-)$, или не саудовлетворени в никоя точка (N).

11.3 Криптосистема на Chor-Rivest

Криптосистемата на Chor-Rivest единствената система, използваща задача за раницата, която не използва умножение по модул за прикриване на лесен вариант на $KNAPSACK(\boldsymbol{a}, \alpha)$. Ще опишем стъпките, които потребителите трябва да извършат при генериране на ключове и използване на тази система за обмен на информация.

За да генерира публичен ключ и съответния му таен ключ всеки потребител извършва следните стъпки:

(1) Избира крайно поле $\mathbb{F}_q$ с характеристика p, $q = p^h$, $p \geq h$, в което намирането на дискретен логаритъм е "лесно" (например $q-1$ има само малки прости делители).

(2) Избира случаен неразложим полином $f(x) \in \mathbb{F}_q[x]$ от степен h със старши коефициент 1 (алгоритъм ??). Елементите на $\mathbb{F}_q$ се представят като полиноми над $\mathbb{F}_p$ от степен по-малка от h.

(3) Избира случаен примитивен елемент $g(x)$ на полето $\mathbb{F}_q$ (алгоритъм ??).

(4) За всеки елемент $i \in \mathbb{F}_p$ пресмята дискретния логаритъм $a_i = \log_{g(x)}(x + i)$.

(5) Избира случайна пермутация π на елементите на множеството $\{0, 1, \ldots, p-1\}$.

(6) Избира случайно цяло число d, $0 \leq d \leq p^h - 2$.

(7) Пресмята
$$b_i = \big(a_{\pi(i)} + d\big) (\mod p^h - 1),\ i = 0, 1, \ldots, p-1.$$

(8) Публичният ключ се състои от числата p, h и вектора $\boldsymbol{b} = (b_0, b_1, \ldots, b_{p-1})$.

Тайният ключ се състои от полиномите $f(x)$, $g(x)$, пермутацията π и числото d.

За да шифрира съобщение m потребителят B извършва следните стъпки:

(1) Получава автентичен публичен ключ $(c_0, c_1, \ldots, c_{p-1})$, p, h.

(2) Нека съобщението m е естествен числа, чието двоично представяне е с дължина $\lceil \log \binom{p}{h} \rceil$.

(3) B Трансформира m в двоичен вектор $\boldsymbol{m} = (m_0, m_1, \ldots, m_{p-1})$ с дължина p, имащ точно h единици:

- $l \leftarrow h$;
- за всяко i, $i = 1, \ldots p$:

  ако $m > \binom{p-1}{l}$, то $m_{i-1} \leftarrow 1$, $m \leftarrow m - \binom{p-i}{l}$, $l \leftarrow l - 1$;

  в противен случай $m_{i-1} \leftarrow 0$.

(4) Пресмята $c = \sum_{i=0}^{p-1} m_i b_i (\mod p^h - 1)$.

(5) Изпраща криптотекста на A.

За да дешифрира криптотекста c получен след шифриране на открит текст m чрез системата на Chor-Rivest с открития ключ, генериран по-горе, потребителят A извършва следните стъпки:

(1) Пресмята $r = (c - hd)(\mod p^h - 1)$.

(2) Пресмята $u(x) = g(x)^r(\mod f(x))$ (алгоритъм ??).

(3) Пресмята $s(x) = u(x) + f(x)$, който е полином над $\mathbb{F}_p$ състарши коефициент 1.

(4) Разлага $s(x)$ на линейни множители над $\mathbb{F}_p$: $s(x) = \prod_{j=1}^{h}(x + t_j)$, където $t_j \in \mathbb{F}_p$.

(5) Пресмята двоичния вектор $\boldsymbol{m} = (m_0, m_1, \ldots, m_{p-1})$. Компонентите на $\boldsymbol{m}$, които имат стойност 1са тези с индекси $\pi^{-1}(t_j)$, $1 \le j \le h$. Всички останали компоненти са 0.

(6) Съобщението (числото) m се получава от $\boldsymbol{m}$ както следва:

- Полага се $m \leftarrow 0$, $l \leftarrow h$.
- За всяко i, $i = 1, \ldots, p$, ако $m_{i-1} = 1$ $m \leftarrow m + \binom{p-1}{l}$ и $l \leftarrow l - 1$.

Ще докажем, че дешифриращият алгоритъм е коректен.

Не е известна смислена атака срещу системата на Chor-Rivest ако параметрите ѝ са внимателно подбрани. Плътността на вектора $\boldsymbol{b} = (b_0, b_1, \ldots, b_{p-1})$ е $\log(\max c_i)$, което е достатъчно голямо за да отхвърли атаки срещу вектори с малка плътност. Известно е, че системата е несигурна, ако са известни части от откритият ключ.

Въпреки че зададохме системата на Chor-Rivest само за прости полета $\mathbb{F}_p$, можем лесно да обобщим алгоритмите и за произволни степени на прости числа.

Алгоритъмъза генериране на параметрите на системата изисква такива цели числа p (просто) и q, че задачата за намиране на дискретен алгоритъм е "лесна". Тези числа могат да бъдат избрани така, че $q = p^h - 1$ има само малки прости делители. Известно е, че в този случай алгоритъмът на Pohlig-Hellman позволява ефективно пресмятане на дисцретен логаритъм в крайното поле $\mathbb{F}_q$.

Преопоръчваните параметри за криптосистемата на Chor-Rivest са $p \approx 200$ и $h \approx 25$. Първоначално предложените параметри са $p = 197$ и $h = 24$; в този случай най-големият прост делител на $197^{24} - 1$ е 10316017 и плътността на вектора $\boldsymbol{b}$ е около 1.077. Други множества от възможни параметри са $p = 211$, $h = 24$; $p = 3^5$, $h = 24$ (основно поле $\mathbb{F}_{3^5}$); и $p = 2^8$, $h = 25$ (основно поле $\mathbb{F}_{2^8}$).

Шифрирането е много бърза операция. Дешифрирането е бавно; тясното място е пресмятането на на $u(x)$ на стъпка (2). Корените на $s(x)$ на стъпка (4) могат да бъдат намерени просто чрез изчерпване на всички възможни елементи от $\mathbb{F}_p$.

Сериозен недостатък на криптосистемата на Chor-Rivest е големината на ключа – около $(ph \cdot \log p)$ бита. За параметрите $p = 197$ и $h = 24$ той е около 36.10^3 бита. Ще отбележим също, че се увеличава и размерът на предаваната информация ц множител от $p^h / \log \binom{p}{h}$. За $p = 197$, $h = 24$ този множител е 1.797.