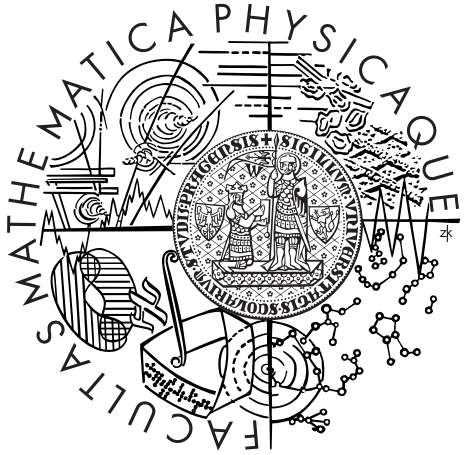
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Martin Jureček

Útoky na bitově orientované proudové šifry obsahující LFSR

Katedra algebry

Vedoucí diplomové práce: RNDr. Bohuslav Rudolf
Studijní program: Matematika
Studijní obor: Matematické metody informační bezpečnosti

Praha 2012

Ďakujem za vytrvalú pomoc, množstvo cenných rád a pripomienok vedúcemu tejto diplomovej práce-RNDr. B. Rudolfovi.

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne a výhradne s použitím citovaných prameňov, literatúry a ďalších odborných zdrojov.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona v platnom znení, predovšetkým skutočnost, že Univerzita Karlova v Prahe má právo na uzavretie licenčnej zmluvy o použití tejto práce ako školského diela podľa §60 odst. 1 autorského zákona.

V ..... dňa ..... Podpis autora

Názov práce: Útoky na bitovo orientované prúdové šifry obsahujúce LFSR
Autor: Martin Jureček
Katedra (ústav): Katedra algebry
Vedúci diplomovej práce: RNDr. Bohuslav Rudolf, Národný bezpečnostný úrad

Abstrakt: V predloženej práci sa venujeme kryptoanalýze jednej z najznámejších prúdových šifier - šifre *A5/1*. Táto šifra sa používa na zabezpečenie komunikácie mobilných telefónov, ktoré sú v súlade so štandardom *GSM*. Základným prvkom šifry *A5/1* je *LFSR*(*posuvný register s lineárnymi spätnými väzbami*), ktorý sa používa v prúdových šifrách, pretože produkuje postupnosť bitov s vysokou periódou, má dobré štatistické vlastnosti a ľahko sa analyzuje pomocou rôznych algebraických metód. V práci sme popísali a implementovali tri útoky na šifru, ktoré predpokladajú znalosť otvoreného textu. Prvé dva útoky sú typu „uhádni a odvoď“ a posledný je korelačný. Ťažiskom práce je kryptoanalýza od Golića, ktorá predpokladá len 64 bitov otvoreného textu. Charakter jeho implementácie dovoľuje úlohu rozdeliť na paralelne prebiehajúce výpočty, vďaka čomu je možné používať program aj v praxi. Na záver práce sa venujeme korelačnému útoku, ktorý je podstatne rýchleší, ale predpokladá znalosť pomerne veľkého množstva otvoreného textu.

Kľúčové slová: LFSR, A5/1, kryptoanalýza prúdovej šifry

Title: Attacks against bit-oriented stream ciphers with LFSRs
Author: Martin Jureček
Department: Department of Algebra
Supervisor: RNDr. Bohuslav Rudolf, National Security Authority

Abstract: In this work we study cryptanalysis one of the most current stream ciphers - *A5/1*. The cipher is used to provide mobile communication privacy in the GSM cellular telephone standard. An essential element of the cipher *A5/1* is *LFSR*(*Linear feedback shift register*) which is used in stream ciphers because it produces a sequence of bits with high periodicity, has good statistical properties and is easily analyzed using various algebraic methods. At work, we describe and implement three known-plaintext attacks on the cipher. The first two attacks are of the type „Guess and Determine“ and the last one is correlation attack. The focus of the work is cryptanalysis by Golić, which assumes only 64 bits of plaintext. The character of implementation allows to split the work and use parallel-computing, making it possible to use the program in practice. At the end of the work we devote to correlation attack, that is considerably faster, but it assumes knowledge of the relatively large amount of plaintext.

Keywords: LFSR, A5/1, cryptanalysis of stream cipher

Obsah

Úvod **2**

1 Popis šifry A5/1 **3**
1.1 Úvod . . . 3
1.2 *LFSR* . . . 3
1.3 Štruktúra šifry *A5/1* . . . 5
1.4 Implementácia šifry . . . 7

2 Kryptoanalýza šifry A5/1 od Bihama a Dunkelmana **9**
2.1 Popis útoku . . . 9

3 Kryptoanalýza šifry A5/1 od Golića **14**
3.1 Útok na vnútorný stav $R^{(101)}$ . . . 14
3.2 Implementácia útoku na vnútorný stav $R^{(101)}$ . . . 21
3.2.1 Implementácia 1.verzie útoku a jej výsledky . . . 21
3.2.2 Implementácia 2.verzie útoku a jej výsledky . . . 26
3.2.3 Porninovo a Sternovo urýchlenie algoritmu . . . 28
3.2.4 Implementácia Porninovho a Sternovho vylepšenia . . . 30
3.3 Problém s existenciou lineárne závislých rovníc . . . 31
3.4 Poznámky k útoku na vnútorný stav $R^{(101)}$ . . . 32
3.5 Útok na počiatočný vnútorný stav $R^{(0)}$ . . . 33

4 Korelačný útok od Ekdahla a Johanssona **38**
4.1 Idea útoku . . . 38
4.2 Popis útoku . . . 40
4.3 Implementácia útoku . . . 45

5 Poznámky k implementácii **50**

Závěr **54**

Zoznam použitej literatúry **55**

6 Prílohy **56**
6.1 Príloha A . . . 56
6.2 Príloha B . . . 58
6.3 Príloha C . . . 60

Úvod

V tejto práci sa zaoberáme kryptoanalýzou jednej z najpoužívanejších šifier na svete, ktorej meno je *A5/1*. Táto šifra patrí medzi prúdové šifry a používa sa na zabezpečenie mobilnej komunikácie v sieťach GSM. Predpokladom útokov spomenutých v tejto práci je znalosť niekoľkých bitov *hesla*, ktoré šifra vyprodukuje. Cieľom útokov je nájsť *tajný kľúč*, pomocou ktorého odvodíme ľubovoľný počet bitov *hesla*, vďaka ktorému už nie je problém získať otvorený text.

V prvej kapitole si predstavíme šifru *A5/1* a uvedieme si poznámky k jej implementácii. Vysvetlíme si, akú funkciu v štruktúre šifry hrá *LFSR*(*posuvný register s lineárnymi spätnými väzbami*) pri generovaní pseudonáhodnej postupnosti bitov. Potom si spomenieme, akým spôsobom je implementovaná šifra *A5/1*, pretože ju budeme používať v implementácii útokov.

V druhej kapitole si uvedieme popis útoku od E.Bihama a O.Dunkelmana, ktorý je typu „uhádni a odvoď“. To znamená, že na začiatku útoku „uhádneme“ niektoré bity vnútorného stavu šifry a pomocou nich spolu so znalosťou štruktúry šifry „odvodíme“ zvyšné neznáme bity. Tento útok má oproti ostatným vyššiu výpočtovú zložitosť, ktorá je 2^{47} taktov šifry. Na druhej strane je zaujímavý svojou elegantnou myšlienkou a je akousi prípravou pre Golićov útok z kapitoly 3.

V jednotlivých podkapitolách kapitoly 3 si postupne uvedieme niekoľko verzií Golićovho útoku spolu s výsledkami implementácie. Tento útok je tiež typu „uhádni a odvoď“ a je špecifický tým, že predpokladá znalosť len 64 bitov *hesla*. Tento počet je najmenší možný pre získanie 64-bitového *tajného kľúča*.

V štvrtej kapitole si uvedieme korelačný útok od P.Ekdahla a T.Johanssona, ktorý je z uvedených útokov s prehľadom najrýchlejší. Avšak tento útok predpokladá znalosť prvých 40 bitov *hesla* pre 70000 rámcov, čo je približne v prepočte až 5 minút GSM komunikácie. V piatej kapitole si popíšeme jednotlivé programy, ktorých výsledky sme uvádzali v práci a vysvetlíme si ich používanie. V závere zhrnieme výsledky celej práce a na konci v prílohách uvedieme vnútorné stavy so špecifickými vlastnosťami a konkrétne matice bitov, ktoré sa používajú počas útokov.

1. Popis šifry A5/1

1.1 Úvod

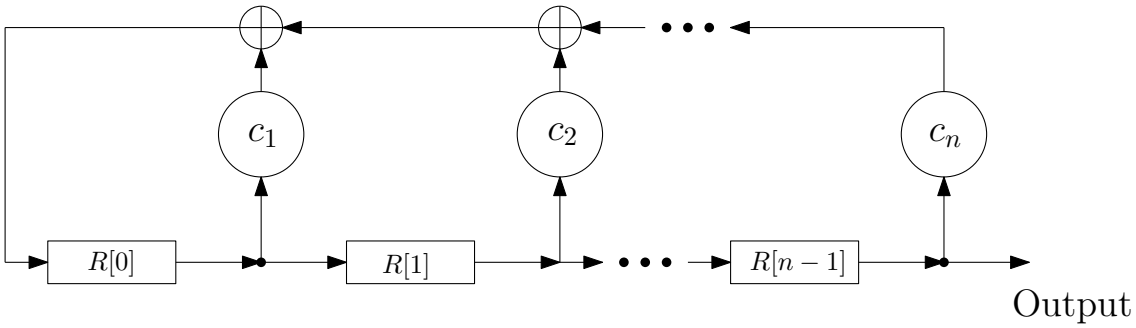
A5/1 je názov prúdovej šifry, ktorá sa používa na zabezpečenie komunikácie mobilných telefónov, ktoré sú v súlade so štandardom *GSM*. Komunikácia pozostáva v prenose postupnosti rámcov(frames) dĺžky 228 bitov, pričom 114 bitov sa využíva na zašifrovanie dát vyslaných z *GSM* centra k užívateľskému mobilnému telefónu a pomocou ďalších 114 bitov sa zašifrujú dáta vyslané z mobilného telefónu do centra.

Priebeh šifrovania vyzerá nasledujúco. *A5/1* vyprodukuje od času t pseudonáhodné *heslo*(keystream), ktoré pozostáva z postupnosti bitov, označme si ju $k_t, k_{t+1}, k_{t+2}, \ldots$ Detailnejší priebeh generovania *hesla* si uvedieme neskôr v tejto kapitole. Po vyprodukovaní *hesla*, t.j. postupnosti $k_i, i = t, t+1, t+2, \ldots,$ zašifrujeme otvorený text, ktorý si môžme predstaviť ako postupnosť bitov $m_i, i = t, t+1, t+2, \ldots$ Samotné šifrovanie prebieha pomocou operácie *XOR*, ktorú budeme značiť $\oplus$, nasledujúco: $c_i = k_i \oplus m_i, i = t, t+1, t+2, \ldots,$ kde c_i sú bity výsledného šifrového textu.

1.2 *LFSR*

Základným kameňom šifry *A5/1* je *posuvný register s lineárnou spätnou väzbou*, ďalej *LFSR*. Najprv si v krátkosti predstavíme štruktúru *LFSR*, uvedieme si jeho vlastnosti a na príklade si ukážeme jeden posun registru.

Štruktúru *LFSR* si vysvetlíme na nasledujúcom obrázku a ďalej register budeme označovať ako R.



Obr. 1.1: Štruktúra *LFSR*.

Políčka $R[0], \ldots, R[n-1]$ sa nazývajú *počiatočný vnútorný stav* registra R. V každom políčku je uložený jeden bit. Počet políčok určuje dĺžku registru, v našom prípade uvažujeme register dĺžky n. Posun registru definujeme nasledujúcimi tromi krokmi:

1. obsah políčka $R[n-1]$ bude výstupným bitom registra,
2. obsah políčka $R[i-1]$ priradíme do políčka $R[i]$ postupne pre $i = n-1, \ldots, 1,$

3. nový obsah políčka $R[0]$ získame pomocou spätnej väzby z predošlých obsahov *význačných* políčok.

Políčko $R[i]$, $i = 0, \ldots, n-1$, je *význačné*, ak konštanta $c_{i+1} \in \mathbb{Z}_2$ je rovná jednej. Do nového políčka $R[0]$ sa potom priradí *XOR* všetkých *význačných* políčok. Konštanty c_i určujú tzv. *charakteristický polynóm* $p(x) = 1 + c_1 x + c_2 x^2 + \ldots + c_n x^n \in \mathbb{Z}_2[x]$. Register R sa nazýva *nesingulárny*, ak $c_n = 1$.

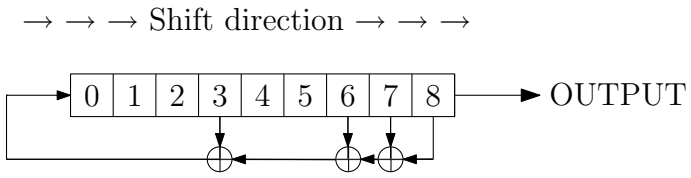
Po n posunoch registra R dostaneme výstupnú postupnosť bitov $R[n-1], \ldots, R[0]$. Výstupné bity $R[i]$ pre $i < 0$ dostaneme pomocou nasledujúceho vzťahu:

$$R[i] = \bigoplus_{j=1}^{n} c_j R[i+j], i < 0.$$

Poznamenajme, že súčin bitov odpovedá logickému operátoru *AND*. V každom posune registru sa vyprodukuje jeden výstupný bit. Výstupné bity tvoria tzv. *výstupnú postupnosť* registru, t.j. postupnosť $R[i], i = n-1, n-2, \ldots$, ktorá sa používa pri generovaní pseudonáhodných postupností bitov.

Ďalej si uveďme jedno tvrdenie o maximálnej perióde *výstupnej postupnosti*. Predpokladajme, že náš register R je *nesingulárny*. Potom platí, že ak *charakteristický polynóm* $p(x)$ registru R je *primitívny*, tak každý z jeho $2^n - 1$ nenulových *inicializačných vnútorných stavov* produkuje *výstupnú postupnosť* maximálnej možnej periódy $2^n - 1$.

Nakoniec si ukážme príklad *LFSR*. Ten je ilustrovaný na obrázku 1.2.

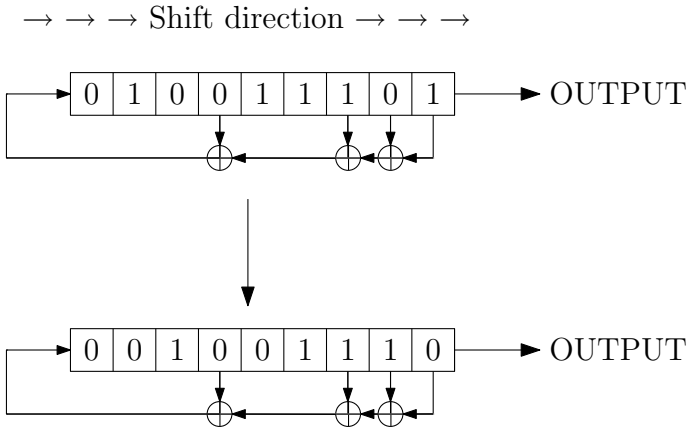


Obr. 1.2: Príklad registru.

Register na obrázku 1.2 má dĺžku 9, jeho políčka sú pre jednoduchosť označené len indexami. Spätná väzba je definovaná *charakteristickým polynómom* $p(x) = 1 + x^4 + x^7 + x^8 + x^9$. Po vykonaní posunu registra bude nová hodnota políčka $R[0]$ rovná hodnote $R[3] \oplus R[6] \oplus R[7] \oplus R[8]$. Na obrázku 1.3, ktorý nájdeme na nasledujúcej strane, je znázornený posun registra.

Z obrázku jasne vidno, že všetky políčka, okrem posledného, presunuli svoje hodnoty do pravého susedného políčka. Do nového políčka s indexom 0 sa priradila hodnota $R[3] \oplus R[6] \oplus R[7] \oplus R[8] = 0 \oplus 1 \oplus 0 \oplus 1 = 0$. Počas tohto kroku sa vyprodukoval výstupný bit s hodnotou 1.

Výhoda *LFSR* spočíva v tom, že môže produkovať postupnosti bitov s vysokou periódou a s dobrými štatistickými vlastnosťami. *LFSR* je tiež vhodné pri hardwarovej implementácii a vďaka jeho štruktúre sa ľahko analyzuje pomocou rôznych algebraických metód. Viac informácií o *LFSR* nájdeme v [1], z ktorého sme prevažne čerpali v tejto podkapitole.

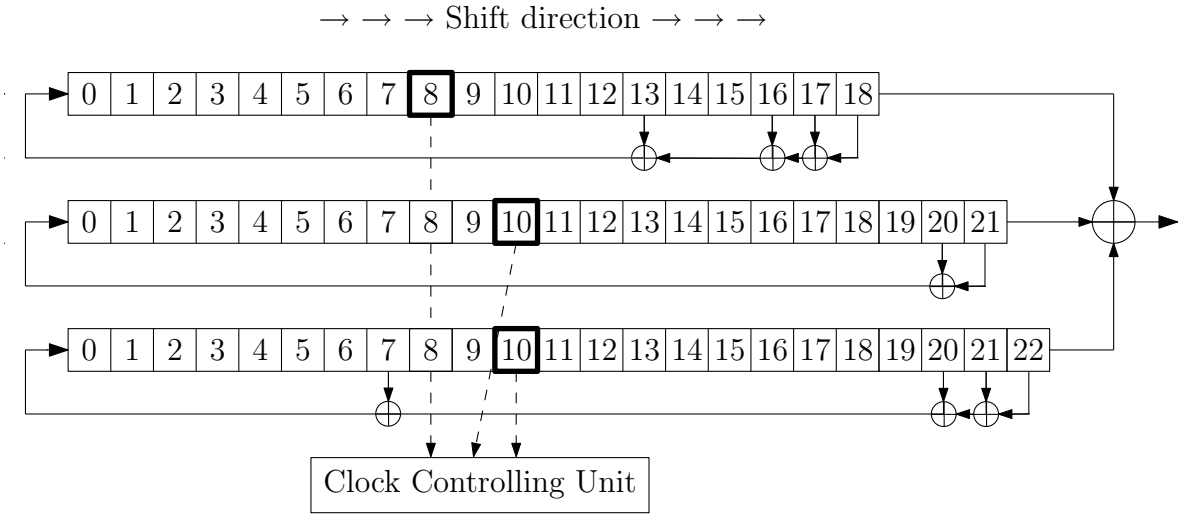


Obr. 1.3: Posun registru.

1.3 Štruktúra šifry *A5/1*

Teraz si predstavíme štruktúru šifry *A5/1*, ktorá pozostáva z kombinácie troch *LFSR* dĺžky 19, 22 a 23, označme si ich postupne R_1, R_2 a R_3. Ich príslušné *charakteristické polynómy* sú po rade $p_1(x) = x^{19}+x^{18}+x^{17}+x^{14}+1$, $p_2(x) = x^{22}+x^{21}+1$ a $p_3(x) = x^{23}+x^{22}+x^{21}+x^8+1$. Pretože každý z uvedených polynómov je *primitívny,* tak *výstupné postupnosti* registrov R_1, R_2 a R_3 sú maximálnej možnej periódy a to po rade $2^{19}-1, 2^{22}-1$ a $2^{23}-1$.

Obrázok 1.4 ukazuje, akým spôsobom šifra *A5/1* pracuje.



Obr. 1.4: Štruktúra šifry A5/1.

Hodnoty, ktoré sú uložené v jednotlivých políčkach, sú bity, kde bit $i-$tého registra v $j-$tom políčku v čase t si označme ako $R_i^{(t)}[j]$.

Posun jednotlivých registrov popisujú nasledujúce vzťahy:

1. $R_1^{(t)}$:

   $N = R_1^{(t)}[18] \oplus R_1^{(t)}[17] \oplus R_1^{(t)}[16] \oplus R_1^{(t)}[13]$
   Do $R_1^{(t+1)}[18-i]$ priraď hodnotu $R_1^{(t)}[18-i-1]$ postupne pre $i = 0, 1, \ldots, 17$
   $R_1^{(t)}[0] = N$.

2. $R_2^{(t)}$:

   $N = R_2^{(t)}[21] \oplus R_2^{(t)}[20]$
   Do $R_2^{(t+1)}[21-i]$ priraď hodnotu $R_2^{(t)}[21-i-1]$ postupne pre $i = 0, 1, \ldots, 20$
   $R_2^{(t)}[0] = N$.

3. $R_3^{(t)}$:

   $N = R_3^{(t)}[22] \oplus R_3^{(t)}[21] \oplus R_3^{(t)}[20] \oplus R_3^{(t)}[7]$
   Do $R_3^{(t+1)}[22-i]$ priraď hodnotu $R_3^{(t)}[22-i-1]$ postupne pre $i = 0, 1, \ldots, 21$
   $R_3^{(t)}[0] = N$.

Bit uložený pod najvyšším indexom sa nazýva *výstupný bit*. *XOR* výstupných bitov všetkých troch registrov v nejakom čase t určuje hodnotu príslušného bitu *hesla*, teda $k_t = R_1^{(t)}[18] \oplus R_2^{(t)}[21] \oplus R_3^{(t)}[22]$. Túto rovnicu nazvime *výstupnou rovnicou*. Ďalším význačným bitom registru je tzv. taktovací bit, ktorý ovplyvňuje posun registra. Taktovacie bity registrov R_1, R_2, R_3 sú po rade $R_1^{(t)}[8], R_2^{(t)}[10]$ a $R_3^{(t)}[10]$.

V jednom takte šifry sa posunú vždy buď dva, alebo tri registre. To, ktoré registre sa posunú, určuje tzv. *Majoritná funkcia*, ktorá sa značí Maj. Táto funkcia má na vstupe taktovacie bity všetkých troch registrov a na výstupe vydá hodnotu, ktorá je totožná s dvomi, alebo s tromi taktovacími bitmi. Tie registre, ktorých taktovacie bity sú rovnaké ako hodnota výstupu funkcie Maj, sa posunú. Rozobrané možnosti posunutia registrov prehľadne ilustruje obrázok 1.5.

Clocking bit of R_1 : $R_1[8]$	0 0 0 1 0 1 1 1
Clocking bit of R_2 : $R_2[10]$	0 0 1 0 1 0 1 1
Clocking bit of R_3 : $R_3[10]$	0 1 0 0 1 1 0 1
	↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
Majority	0 0 0 0 1 1 1 1
Clock R_1?	✓ ✓ ✓ × × ✓ ✓ ✓
Clock R_2?	✓ ✓ × ✓ ✓ × ✓ ✓
Clock R_3?	✓ × ✓ ✓ ✓ ✓ × ✓

Obr. 1.5: Tabuľka posunov registrov.

Funkcia Maj má nelineárny charakter, ktorý prispieva k väčšej bezpečnosti šifry. Mechanizmus posunutia registrov, respektíve takt šifry, máme uvedený a stačí už len popísať inicializáciu registrov.

Pre každý rámec dĺžky 228 bitov sa vykoná inicializácia registrov, pri ktorej sa použije 64-bitový kľúč K a 22-bitové číslo rámca, označme si ho $Frame$. Číslo rámca je pre každý rámec rôzne a je verejne známe, pričom kľúč K je počas celej konverzácie rovnaký, t.j. pre všetky rámce, z ktorých sa skladá konverzácia sa nemení a mení sa len pri začatí novej konverzácie.

Inicializačný algoritmus šifry *A5/1* je nasledujúci:

1. Nastav všetky políčka všetkých troch registrov na nulu.
2. `FOR` $i = 0$ `TO` 63 `DO`
   - $R_1[0] = R_1[0] \oplus K[i]$
   - $R_2[0] = R_2[0] \oplus K[i]$
   - $R_3[0] = R_3[0] \oplus K[i]$
   - Posuň všetky tri registre, t.j. pre $j > 0$ $R_i[j] = R_i[j-1]$, a do políčka $R_i[0]$ priraď spätnú väzbu z predchádzajúcich hodnôt R_i.
3. `FOR` $i = 0$ `TO` 21 `DO`
   - $R_1[0] = R_1[0] \oplus Frame[i]$
   - $R_2[0] = R_2[0] \oplus Frame[i]$
   - $R_3[0] = R_3[0] \oplus Frame[i]$
   - Posuň všetky tri registre rovnako ako v 2.kroku

   Stav, ktorý vznikol bezprostredne po tejto fáze inicializácie sa nazýva *inicializačný vnútorný stav šifry.*
4. `FOR` $i = 0$ `TO` 99 `DO`
   - Vykonaj takt šifry (t.j. už použijeme funkciu Maj)

Po inicializácii registrov šifra *A5/1* vyprodukuje 228 bitov *hesla.*

1.4 Implementácia šifry

V tejto kapitole si predstavíme softwarovú reprezentáciu registrov a taktu šifry. Pri implementácii sme sa inšpirovali článkom [2].

Popíšeme si, ako je v programe reprezentovaný *vnútorný stav šifry A5/1. Vnútorný stav šifry A5/1* pozostáva z troch registrov, ktoré obsahujú určitý počet bitov. Každý register je v programe reprezentovaný dostatočne dlhým poľom, ktorého zložkami sú prvky množiny $\{0, 1\}$. Jednotlivé zložky poľa sú indexované celými číslami, kde prvý prvok poľa má index 0, druhý prvok poľa má index 1, atď. Označme si, že pole r_1 bude reprezentovať register R_1, pole r_2 bude reprezentovať register R_2 a konečne pole r_3 bude reprezentovať register R_3.

V popise šifry *A5/1* sú *výstupné bity* jednotlivých registrov v čase $t = 101$, t.j. bity $R_1^{(101)}[18], R_2^{(101)}[21], R_3^{(101)}[22]$, označené najvyššími indexmi. V programe sú reprezentované naopak najnižšími indexmi a celé indexovanie je v opačnom smere, ako je v popise šifry. Tým sa myslí to, že bit $R_1^{(101)}[18]$ je v poli r_1 uložený

pod zložkou $r_1[0]$, bit $R_1^{(101)}[17]$ je v poli r_1 uložený pod zložkou $r_1[1]$, atď., až nakoniec bit $R_1^{(101)}[0]$ je uložený pod zložkou $r_1[18]$. Analogicky si vieme odvodiť, pod ktorou zložkou budú bity registra R_2, resp. R_3, uložené v poli r_2, resp. r_3. Toto indexovanie v opačnom smere slúži k tomu, aby sme nemuseli pracovať so zápornými indexmi. Napr. bit $R_1^{(101)}[-1]$ bude v poli r_1 uložený v zložke $r_1[19]$.

Teraz si popíšeme, ako je v programe reprezentovaný takt šifry. V čase $t = 101$, keď sa ešte v programe nevykonal žiadny takt, sú všetky *výstupné bity* registrov R_1, R_2, R_3 uložené v poliach r_1, r_2, r_3 pod zložkami s nulovými indexmi. Pre každý register si vytvoríme premennú, ktorá bude počítať počet posunutí registra od času $t = 101$. Pre register R_1 si túto premennú nazvime $citacR_1$, pre register R_2 si túto premennú nazvime $citacR_2$ a konečne pre register R_3 si túto premennú nazvime $citacR_3$. Takže v čase $t = 101$ bude platiť, že $citacR_1 = 0, citacR_2 = 0$ a $citacR_3 = 0$.

Na základe hodnôt taktovacích bitov registrov sa spočíta hodnota *Majoritnej funkcie*. Potom na základe tejto hodnoty sa posunú len určité registre. Samotný posun registru bude v programe vykonaný aktualizovaním príslušnej premennej na počítanie posunutí. Teda ak sa register R posunie, tak hodnotu $citacR$ zvýšime o jedna. A ak sa neposunie, tak hodnotu $citacR$ nezmeníme. Nech napríklad sa majú posunúť len registre R_1 a R_3. Potom v čase $t = 102$ bude platiť, že $citacR_1 = 1, citacR_2 = 0$ a $citacR_3 = 1$. To znamená, že počas taktu šifry sa v poliach nebudú premiestňovať žiadne zložky o zložku ďalej, ale len sa aktualizuje hodnota $citacR_i$ pre každý register R_i, $i = 1, 2, 3$. Z uvedeného môžeme odvodiť, že v každom čase $t \geq 101$ budú výstupné bity uložené v poliach r_1, r_2, r_3 pod zložkami $r_1[citacR_1], r_2[citacR_2], r_3[citacR_3]$. Podobne taktovacie bity budú uložené v zložkách $r_1[10 + citacR_1], r_2[11 + citacR_2], r_3[12 + citacR_3]$.

Na záver ešte spomenieme odkaz [3], v ktorom nájdeme pedagogickú implementáciu šifry *A5/1* napísanú v jazyku C. Spolu s kompletným a funkčným zdrojovým kódom tam nájdeme aj popis jednotlivých podprogramov.

2. Kryptoanalýza šifry A5/1 od Bihama a Dunkelmana

Ako prvý si predstavíme útok na šifru $A5/1$, ktorý popísali Biham a Dunkelman v článku [4]. Tento útok počíta so znalosťou otvoreného textu a jeho príslušného šifrového textu. Takýto typ útoku je v kryptografii známy pod pojmom *known-plaintext attack*. Pretože samotné šifrovanie prebieha pomocou operácie *XOR*, t.j. bity otvoreného textu sčítame pomocou tejto operácie s príslušnými bitmi *hesla*, vieme si bity *hesla* jednoducho odvodiť.

Cieľom útoku bude nájsť obsah všetkých troch registrov R_1, R_2 a R_3, resp. vnútorný stav šifry $A5/1$, čo je spolu 64 bitov. Časová zložitosť nasledujúceho útoku je 2^{47}, pričom časová jednotka je definovaná ako jeden takt šifry. Tento útok vyžaduje znalosť približne 2^{20} bitov *hesla*.

2.1 Popis útoku

Idea útoku je založená na čakaní na špeciálny prípad, vďaka ktorému môžeme využiť štruktúru šifry a ktorý nám poskytne viac informácií o vnútornom stave šifry. Nech postupnosť $k_t, k_{t+1}, k_{t+2}, \ldots$ označuje postupnosť bitov *hesla*, ktorú vyprodukuje šifra s vnútorným stavom $R^{(t)} = (R_1^{(t)}, R_2^{(t)}, R_3^{(t)})$ od času t. Cieľom útoku nie je nájsť obsah vnútorného stavu $R^{(t)}$ pre konkrétny čas t. Pomocou tohto útoku nájdeme len obsah takého vnútorného stavu $R^{(t)}$, ktorý je účastníkom spomínaného špeciálneho prípadu.

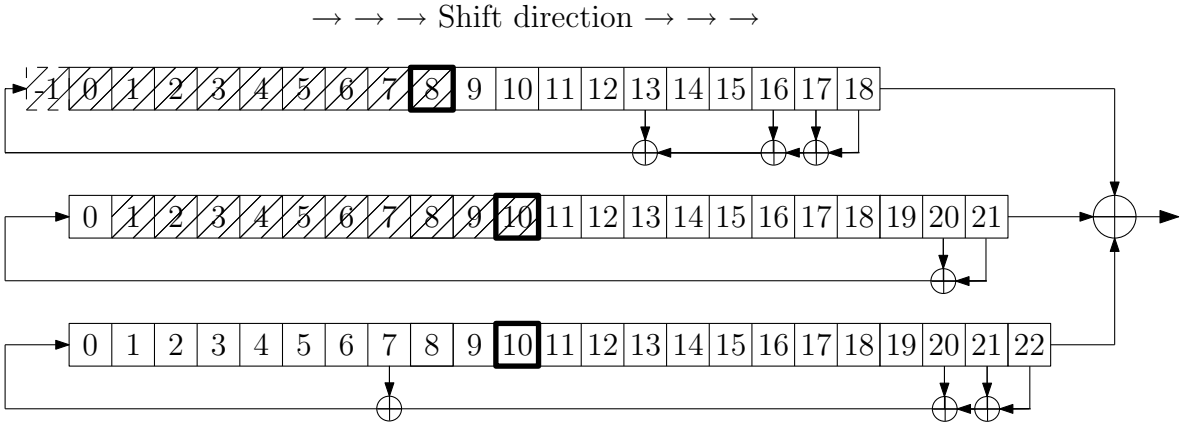
Predpokladajme, že sa tretí register $R_3^{(t)}$ nasledujúcich desať taktov šifry neposunie. Tento špeciálny prípad nastane s pravdepodobnosťou 2^{-20}. Ak predpokladáme znalosť taktovacieho bitu $R_3^{(t)}[10]$, ihneď získame hodnotu výrazu zo spätnej väzby $R_1^{(t)}[13] \oplus R_1^{(t)}[16] \oplus R_1^{(t)}[17] \oplus R_1^{(t)}[18]$, ďalej získame znalosť bitov $R_1^{(t)}[0], \ldots, R_1^{(t)}[8]$ a bity $R_2^{(t)}[1], \ldots, R_2^{(t)}[10]$, pričom všetky tieto hodnoty spočítame ako komplement bitu $R_3^{(t)}[10]$ vďaka predpokladu, že tretí register sa nasledujúcich desať krokov neposunie. To znamená, že taktovacie bity prvých dvoch registrov pre nasledujúcich 10 taktov šifry musia byť navzájom zhodné a zároveň rôzne od bitu $R_3^{(t)}[10]$.

Ak by sa register $R_3^{(t)}$ nasledujúcich desať taktov šifry neposunul a poznali by sme hodnotu taktovacieho bitu $R_3^{(t)}[10]$, tak obrázok 2.1 na ďalšej strane ilustruje, ktoré bity budeme vedieť odvodiť. Tieto bity sú zvýraznené vyšrafovaním a políčko s indexom -1 reprezentuje bit $R_1^{(t+1)}[0] = R_1^{(t)}[13] \oplus R_1^{(t)}[16] \oplus R_1^{(t)}[17] \oplus R_1^{(t)}[18]$.

Pripomeňme, že bit hesla v čase t spočítame ako *XOR* príslušných výstupných bitov všetkých troch registrov: $k_t = R_1^{(t)}[18] \oplus R_2^{(t)}[21] \oplus R_3^{(t)}[22]$. Výstupný bit tretieho registra sa nasledujúcich 10 taktov nebude meniť, preto predpokladajme znalosť aj tohto bitu. Potom budeme poznať hodnotu týchto výrazov $R_1^{(t)}[i] \oplus R_2^{(t)}[i+3], i = 8, \ldots, 18$. Ďalej predpokladajme, že poznáme nasledujúce bity:

$$R_1^{(t)}[18], R_1^{(t)}[17], R_1^{(t)}[16], R_1^{(t)}[15], R_1^{(t)}[14], R_1^{(t)}[12], R_1^{(t)}[11], R_1^{(t)}[10], R_1^{(t)}[9].$$

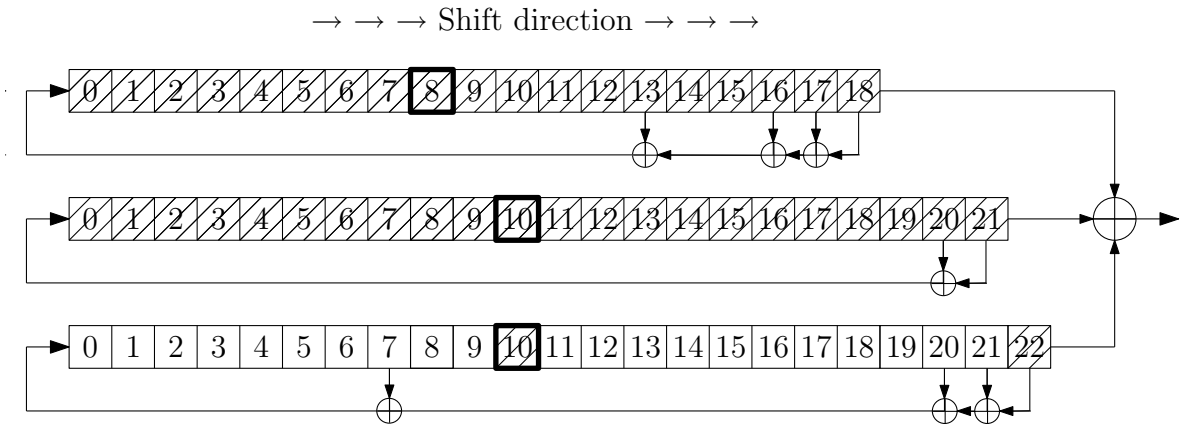
Poznamenajme, že bit $R_1^{(t)}[13]$ už dokážeme spočítať spočítať pomocou spätnej



Obr. 2.1: 20 odvodených bitov zvýraznených vyšrafovaním.

väzby prvého registru: $R_1^{(t)}[13] = R_1^{(t)}[18] \oplus R_1^{(t)}[17] \oplus R_1^{(t)}[16] \oplus R_3^{(t)}[10] \oplus 1$. Vzhľadom k tomu, že poznáme hodnoty výrazov $R_1^{(t)}[i] \oplus R_2^{(t)}[i+3], i = 8, \ldots, 18$ a bity celého prvého registra, dopočítame si hodnoty bitov $R_2^{(t)}[i+3]$, pre $i = 8, \ldots, 18$. Za predpokladu znalosti bitu $R_2^{(t)}[0]$ poznáme všetky bity druhého registra.

Pre prehľadnosť si na obrázku 2.2 ilustrujeme všetky doteraz známe bity vnútorného stavu. Tieto bity sú zvýraznené vyšrafovaním.



Obr. 2.2: Známe bity zvýraznené vyšrafovaním.

Teraz uvažujme situáciu bezprostredne po desiatich taktoch šifry $A5/1$, v ktorých sa tretí register ani raz neposunul. Získali sme všetky bity prvého a druhého registra, pričom sme potrebovali uhádnuť iba týchto 12 bitov:

- bity $R_1 : R_1^{(t)}[9], R_1^{(t)}[10], R_1^{(t)}[11], R_1^{(t)}[12], R_1^{(t)}[14], R_1^{(t)}[15], R_1^{(t)}[16], R_1^{(t)}[17], R_1^{(t)}[18]$,
- bit $R_2 : R_2^{(t)}[0]$,
- bity $R_3 : R_3^{(t)}[10], R_3^{(t)}[22]$.

V poslednej fáze tohto útoku na šifru $A5/1$ získame znalosť zvyšných bitov tretieho registra, ktoré ešte nepoznáme. Pretože poznáme taktovací bit $R_3^{(t)}[10]$ a taktovacie bity prvých dvoch registrov, vieme, ktoré registre sa v najbližšom takte

posunú. Preto počkáme na situáciu, keď sa tretí register prvýkrát po desaťtaktovej pauze posunie. Potom bit $R_3^{(t)}[21]$, ktorý bude v aktuálnom momente výstupným bitom tretieho registra, jednoducho získame z výstupnej rovnice.

Nakoniec predpokladajme, že poznáme bity $R_3^{(t)}[0], \ldots, R_3^{(t)}[9]$. Každý z týchto desiatich bitov bude v určitom čase taktovacím bitom, čím budeme v každom čase vedieť, ktoré registre sa posunú. Pri znalosti *hesla* a všetkých bitov prvých dvoch registrov môžeme spočítať, analogickým postupom ako v prípade bitu $R_3^{(t)}[21]$, zvyšných 10 neznámych bitov tretieho registra: $R_3^{(t)}[11], \ldots, R_3^{(t)}[20]$. Napríklad ak už poznáme bit $R_3^{(t)}[21]$, tak si počkáme, kým sa tretí register opäť posunie, čím sa bit $R_3^{(t)}[20]$ dostane na pozíciu výstupného bitu a jednoducho tento bit odvodíme z výstupnej rovnice. Týmto spôsobom získame celý obsah vnútorného stavu.

Zhrnieme si, ktoré bity hľadaného vnútorného stavu sme predpokladali, že poznáme. Sú to nasledujúce bity, označme si ich ako *hádané bity*:

- bity R_1: $R_1^{(t)}[9], R_1^{(t)}[10], R_1^{(t)}[11], R_1^{(t)}[12], R_1^{(t)}[14], R_1^{(t)}[15], R_1^{(t)}[16], R_1^{(t)}[17], R_1^{(t)}[18]$,
- bity R_2: $R_2^{(t)}[0]$,
- bity R_3: $R_3^{(t)}[0], \ldots, R_3^{(t)}[10]$ a $R_3^{(t)}[22]$.

Teraz uvedieme pseudokód popísaného útoku. Vstupom algoritmu bude dostatočný počet bitov *hesla*: $k_t, k_{t+1}, k_{t+2}, \ldots$ V priemere ich budeme potrebovať približne 2^{20}. Výstupom bude vnútorný stav $R^{(j)} = (R_1^{(j)}, R_2^{(j)}, R_3^{(j)})$ a čas j, v ktorom sa bude nachádzať jeho tvar. To znamená, že tento vnútorný stav bude produkovať výstupné bity $k_j, k_{j+1}, k_{j+2}, \ldots$

ALGORITMUS 2.1

VSTUP: *heslo*: $k_t, k_{t+1}, k_{t+2}, \ldots$
VÝSTUP: vnútorný stav $R^{(j)}$, čas j

1. Priraď $j = t$, *Nenájdený vnútorný stav*$= True$,
2. `WHILE` *Nenájdený vnútorný stav* `DO`
   - 2.1 `FOR EACH` *hádané bity* `DO`
     - 2.1.1 *Výpočet ostatných bitov*(*heslo*),
     - 2.1.2 *Overenie Správnosti Kandidáta*(*heslo*),
     - 2.1.3 `IF` správny kandidát `THEN`
       Priraď *Nenájdený vnútorný stav* $= False$ a *BREAK*,
       (príkazom *BREAK* sa ukončí FOR-cyklus)
   - 2.2 Priraď $j = j + 1$,
3. `RETURN` $R^{(j-1)}$, čas $j - 1$.

V 1.kroku si do premennej j priradíme index prvého uvažovaného bitu *hesla*, t.j. postupnosti $k_t, k_{t+1}, k_{t+2}, \ldots$, pre ktorú hľadáme vnútorný stav šifry $R^{(t)} = (R_1^{(t)}, R_2^{(t)}, R_3^{(t)})$. Ďalej si do premennej *Nenájdený vnútorný stav* uložíme informáciu, že sme ešte hľadaný vnútorný stav nenašli. Ťažiskom Algoritmu 2.1 je v 2.kroku WHILE-cyklus, ktorý sa opakuje dovtedy, pokým nenájdeme hľadaný

vnútorný stav. Vo vnútri WHILE-cyklu v kroku 2.1 vstupujeme do ďalšieho cyklu - FOR-cyklu, pomocou ktorého prechádzame všetky možné hodnoty *hádaných bitov*. Na základe konkrétnych hodnôt *hádaných bitov* si v kroku 2.1.1 vypočítame ostatné neznáme bity vnútorného stavu. To vykoná procedúra *Výpočet ostatných bitov*, ktorú si podrobnejšie vysvetlíme neskôr. Keď už budeme poznať všetky bity uvažovaného vnútorného stavu, v kroku 2.1.2 si overíme správnosť tohto kandidáta pomocou *hesla* takto: kandidáta necháme taktovať a budeme porovnávať jeho vyprodukované výstupné bity s príslušnými bitmi *hesla*. Ak sa tieto bity budú zhodovať, náš kandidát prešiel overovacou fázou. v opačnom prípade kandidát neprešiel overovacou fázou a výpočet pokračuje opäť v kroku 2.1, kde sa pomocou FOR-cyklu zvolia v poradí ďalšie ohodnotenia *hádaných bitov*.

Keď získame správny vnútorný stav po overovacej fáze, tak v kroku 2.1.3 do premennej *Nenájdený vnútorný stav* priradíme informáciu, že už máme výsledok a nemusíme znovu prechádzať WHILE-cyklus. Zároveň vykonaním príkazu *BREAK* ukončíme prechádzanie FOR-cyklu.

Ak kandidát na hľadaný vnútorný stav šifry neprejde overením správnosti a to pre všetky možnosti *hádaných bitov*, tak to znamená, že tento vnútorný stav nesplňuje predpoklad, na ktorom je postavený tento útok, t.j. že tretí register sa od času j nasledujúcich 10 taktov šifry neposunul. Potom zvýšime index j o jedna a výpočet znovu pokračuje od kroku 2.1. V tomto duchu zvyšujeme index j toľkokrát, až kým natrafíme na takú postupnosť bitov *hesla*, ktorá nám prinesie hľadaný vnútorný stav.

Teraz sa detailnejšie pozrime na procedúru *Výpočet ostatných bitov*. Na vstupe dostane procedúra postupnosť bitov *hesla* $k_t, k_{t+1}, k_{t+2}, \ldots$ od určitého indexu t a 22 *hádaných bitov*. Výstupom bude kandidát na vnútorný stav šifry $R^{(j)}$ a čas j. V tejto procedúre sa dopočítajú hodnoty bitov, ktoré sme nemuseli "hádať". Nasleduje pseudokód tejto procedúry.

Výpočet ostatných bitov

VSTUP: *heslo*: $k_t, k_{t+1}, k_{t+2}, \ldots$, *hádané bity*
VÝSTUP: kandidát na vnútorný stav šifry $R^{(j)}$ a čas j

1. `FOR` i=0,...,8 `DO`
   1.1 Priraď $R_1^{(t)}[i] = 1 \oplus R_3^{(t)}[10]$,
2. Priraď $R_1^{(t)}[13] = R_1^{(t)}[18] \oplus R_1^{(t)}[17] \oplus R_1^{(t)}[16] \oplus R_3^{(t)}[10] \oplus 1$,
3. `FOR` i=1,...,10 `DO`
   3.1 Priraď $R_2^{(t)}[i] = 1 \oplus R_3^{(t)}[10]$,
4. `FOR` i=0,...,10 `DO`
   4.1 $R_2^{(t)}[21-i] = R_1^{(t)}[18-i] \oplus R_3^{(t)}[22] \oplus k_{t+i}$,
5. `FOR` i=0,...,10 `DO`
   5.1 `WHILE` R_3 sa neposunul `DO`
      5.1.1 *Vykonaj takt šifry*,
   5.2 *Spočítaj*$(R_3^{(t)}[21-i])$,
6. `RETURN` $R^{(j)}$ a čas j.

V krokoch 1. až 4. priraďujeme podľa popisu útoku do neznámych bitov hodnoty *hádaných bitov*. V 5.kroku dopočítame zvyšných jedenásť neznámych bitov kandidáta na vnútorný stav, t.j. bitov $R_3^{(t)}[11], \ldots, R_3^{(t)}[21]$. Pre každý neznámy

bit budeme v kroku 5.1.1 vykonávať takt šifry dovtedy, pokým sa posunie tretí register. Akonáhle sa posunie tretí register, tak sa neznámy bit dostane na pozíciu výstupného bitu registru a v kroku 5.2 ho ľahko získame z výstupnej rovnice, v ktorej budeme poznať všetky ostatné bity.

Nakoniec si odvodíme celkovú zložitosť celého algoritmu. Špeciálny prípad, keď sa tretí register 10-krát po sebe neposunie, nastane v priemere v jednom z 2^{20} prípadov. Ak nastane tento špeciálny prípad, potrebujeme poznať hodnoty 22 *hádaných bitov*, pričom všetkých možností ohodnotení týchto bitov je 2^{22}. V procedúre *Výpočet ostatných bitov* musíme v 5.kroku vykonať niekoľko taktov, kým získame posledné neznáme bity tretieho registra. Pretože sa register pri každom takte posunie s pravdepodobnosťou 3/4, tak v kroku 5.1.1 sa vykoná takt šifry v priemere 16-krát. Poslednou zložkou výpočtovej zložitosti je zložitosť overovacej fázy.

Spočítame si zložitosť overovacej fázy. Pre každého kandidáta sa musí vykonať aspoň jeden takt šifry, aby sa jeho vyprodukovaný výstupný bit mohol porovnať s príslušným bitom *hesla*. Ak sú tieto dva bity rôzne, čo nastane s pravdepodobnosťou 1/2, tak overovacia fáza končí. Ak sú tieto dva bity rovnaké, tak pre zvyšnú polovicu kandidátov, pre ktorých ešte neprebehla overovacia fáza, sa musí vykonať ďalší takt šifry, aby sa overovacia fáza mohla ukončiť. Analogicky pre štvrtinu kandidátov sa musia vykonať aspoň tri takty, aby sa overovacia fáza ukončila, atď. Celkovo sa v overovacej fáze v priemere vykonajú $1 + 1/2 + 1/4 + 1/8 + ... = 2$ takty. Potom celková zložitosť útoku je rovná $2^{20} \cdot 2^{22} \cdot 2^4 \cdot 2 = 2^{47}$.

3. Kryptoanalýza šifry A5/1 od Golića

Nasledujúci útok je rovnako ako predchádzajúci typu *known-plaintext attack*, t.j. počíta so znalosťou časti otvoreného textu a príslušnej časti šifrového textu. Najprv bude naším cieľom získať vnútorný stav $R^{(t)} = (R_1^{(t)}, R_2^{(t)}, R_3^{(t)})$ šifry *A5/1* v čase $t = 101$. $R^{(101)}$ je vnútorný stav šifry, ktorý vyprodukuje prvý bit hesla nejakého úseku(frame). Druhým cieľom bude určiť vnútorný stav $R^{(t)}$ v čase $t = 0$, tzv. *inicializačný vnútorný stav šifry*. Je to stav, ktorý vznikne ihneď po naplnení registrov číslom rámca v inicializačnej fáze. Počas jeho vytvorenia sa ešte nepoužívala *Majoritná funkcia*. V tejto druhej fáze útoku už nebudeme mať k dispozícii heslo.

Na šifru *A5/1* sa môžeme pozerať ako na funkciu, ktorá v jednom takte transformuje jeden vnútorný stav šifry na druhý. Golić vo svojom článku [5] dokázal, že táto funkcia nie je prostá. To znamená, že množina všetkých dosiahnuteľných stavov $R^{(t)}$, pre čas $t \geq 1$, ktoré získame po t taktoch z *inicializačného vnútorného stavu* $R^{(0)}$, je len vlastnou podmnožinou množiny všetkých 2^{64} možných vnútorných stavov. Ďalej uvádza, že len $5 \cdot 2^{61}$, čo je približne $2^{63,32}$, vnútorných stavov je dosiahnuteľných pre čas $t = 1$. Z tohto tvrdenia vyplýva, že rôzne *inicializačné vnútorné stavy* môžu dosiahnúť rovnaký vnútorný stav v nejakom čase t, alebo môžu vyprodukovať rovnaké heslo.

Pre náš útok budeme predpokladať znalosť prvých 64 bitov hesla $k_1, k_2, \ldots, k_{64}$ (v nejakom 228-bitovom úseku). Bit k_1 hesla je teda prvý výstupný bit hľadaného vnútorného stavu $R^{(101)}$. Týchto 64 bitov hesla použijeme na overenie správnosti kandidáta na hľadaný vnútorný stav. Golić v [5] pomocou "theory of branching processes"([6], [7]) dokázal, že počet kandidátov na hľadaný vnútorný stav $R^{(101)}$, ktorých výstup je rovnaký ako naše 64-bitové heslo, môže byť viac ako jeden. Pričom uvádza, že ich počet je s veľkou pravdepodobnosťou malý.

3.1 Útok na vnútorný stav $R^{(101)}$

Teraz si predstavíme útok na vnútorný stav $R^{(101)}$ a pritom budeme postupne počítať aj zložitosť jednotlivých krokov výpočtov. Pretože stav $R^{(101)}$ pozostáva zo 64 neznámych bitov, tak nájdenie obsahu $R^{(101)}$ môžeme dosiahnúť nájdením a vyriešením sústavy 64 lineárnych a navzájom nezávislých rovníc so 64 neznámymi. Premenné tejto sústavy lineárnych rovníc potom budú:

$$R_1^{(101)}[0], R_1^{(101)}[1], \ldots, R_1^{(101)}[18],$$
$$R_2^{(101)}[0], R_2^{(101)}[1], \ldots, R_2^{(101)}[21],$$
$$R_3^{(101)}[0], R_3^{(101)}[1], \ldots, R_3^{(101)}[22]$$

Ako bolo uvedené, počet všetkých dosiahnuteľných stavov $R^{(101)}$ z *inicializačného vnútorného stavu* $R^{(0)}$ nie je väčší ako približne $2^{63,32}$. Golić v [5] uvádza spôsob, ako nedosiahnuteľné stavy možno popísať lineárnymi rovnicami, my sa však v

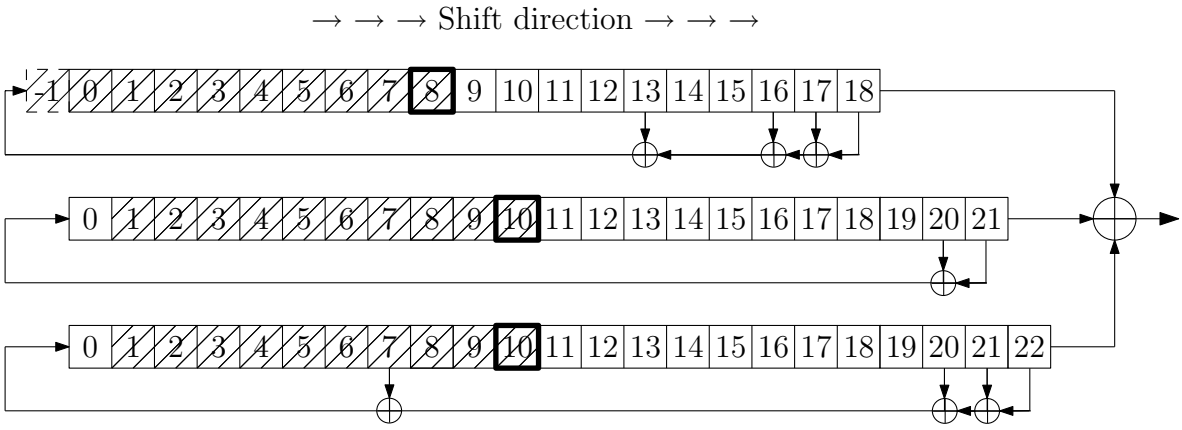
tejto práci nebudeme ich popisom zaoberať. Preto namiesto uvažovania sústavy 64 rovníc nám k získaniu obsahu $R^{(101)}$ stačí len v priemere 63,32 rovníc.

V každom registri predpokladáme, že poznáme po nasledujúcich 10 bitov:

- bity prvého registra R_1: $R_1^{(101)}[-1], R_1^{(101)}[0], R_1^{(101)}[1], \ldots, R_1^{(101)}[8]$,
- bity druhého registra R_2: $R_2^{(101)}[1], R_2^{(101)}[2], \ldots, R_2^{(101)}[10]$,
- bity tretieho registra R_3: $R_3^{(101)}[1], R_3^{(101)}[2], \ldots, R_3^{(101)}[10]$.

Spolu týchto 30 bitov budeme ďalej označovať ako *hádané bity*. Pojmom *hádané bity* sa myslí to, že na začiatku algoritmu si zvolíme nejaké ohodnotenie týchto 30-tich bitov a s ním ďalej pokračujeme vo výpočte, pričom predpokladáme, že zvolené ohodnotenie je správne. Neskôr v algoritme prebehne overovacia fáza, ktorá určí, či zvolené ohodnotenie *hádaných bitov* je správne, alebo nie. Ak je nesprávne, znovu sa dostaneme na začiatok algoritmu a zvolíme ďalšie ohodnotenie *hádaných bitov*.

Vo vnútornom stave $R^{(101)}$ sú *hádané bity* na obrázku 3.1 zvýraznené vyšrafovaním.



Obr. 3.1: 30 *hádaných bitov* zvýraznených vyšrafovaním.

Tento útok vznikol v čase, keď ešte nebola odhalená štruktúra šifry *A5/1*. Preto Golić pracoval s verziou šifry, ktorá sa podľa neho najviac podobala skutočnej šifre *A5/1*. Jeho verzia bola odhadnutá veľmi presne až na pár malých detailov. Pretože my sa budeme venovať kryptoanalýze skutočnej verzii šifry *A5/1*, budeme si musieť Golićov postup prispôsobiť. Jediný z rozdielov medzi oboma verziami, ktorý by mohol ovplyvniť priebeh útoku je ten, že Golićova verzia šifry mala v prvom registri taktovací bit až na pozícii $R_1^{(t)}[9]$ a nie na pozícii $R_1^{(t)}[8]$, ako má pôvodná šifra *A5/1*. Túto odlišnosť vykompenzujeme tým, že predpokladáme znalosť bitu $R_1^{(101)}[-1]$, čo je bit, ktorý bude mať v nasledujúcom takte označenie $R_1^{(102)}[0]$. Pretože $R_1^{(101)}[-1]$ nie je premenná v našej sústave rovníc, musíme si ju vyjadriť pomocou premenných sústavy. To vykonáme výrazom $R_1^{(t)}[-1] = R_1^{(t)}[18] \oplus R_1^{(t)}[17] \oplus R_1^{(t)}[16] \oplus R_1^{(t)}[13]$, ktorý je odvodený od spätnej väzby prvého registra. Týmto spôsobom sme síce zmenili tvar jednej rovnice sústavy, ale nezmenilo sa riešenie, ani výpočtový čas a celkovo sa nezmenil ani

charakter útoku. Ostatné rovnice, ktoré sme dostali dosadením hodnôt *hádaných bitov* do príslušných premenných sústavy, sú v tvare $x = k$, kde x je premenná sústavy rovníc a $k \in \{0, 1\}$.

Podrobnejšie si ukážeme, ako vyzerá jeden riadok homogénnej časti matice pre nejakú konkrétnu rovnicu. V prvom rade si zadefinujeme, ktorá premenná našej sústavy prísluší ku ktorému stĺpci matice. Stĺpce matice budú príslušné nasledujúcim premenným v tomto poradí:

$$R_1^{(101)}[0], \ldots, R_1^{(101)}[18], R_2^{(101)}[0], \ldots, R_2^{(101)}[21], R_3^{(101)}[0], \ldots, R_3^{(101)}[22]$$

Takže 1. stĺpec patrí premennej $R_1^{(101)}[0]$, 20.stĺpec patrí premennej $R_2^{(101)}[0]$, ďalej 42. stĺpec patrí premennej $R_3^{(101)}[0]$ a posledný 64. stĺpec bude patriť premennej $R_3^{(101)}[22]$. Preto keď chceme doplniť do sústavy napríklad rovnicu $R_2^{(101)}[1] = k$, tak riadok homogénnej časti matice bude mať tvar $(0, \ldots, 0, 1, 0, \ldots, 0)$, kde 1 bude na pozícii 21. Hodnota konštanty k bude v pravej strane matice na príslušnom riadku. Na druhej strane, keď budeme pridávať do sústavy rovnicu $R_1^{(101)}[-1] = k$, tak ako sme spomenuli vyššie, budeme v skutočnosti pridávať rovnicu $R_1^{(t)}[18] \oplus R_1^{(t)}[17] \oplus R_1^{(t)}[16] \oplus R_1^{(t)}[13] = k$. Riadok homogénnej časti matice bude mať potom tvar $(0, \ldots, 0, 1, 0, 0, 1, 1, 1, 0, \ldots, 0)$, kde prvá jednotka zľava prísluší premennej $R_1^{(t)}[13]$, a preto je na 14.-tej pozícii.

Golićov útok je postavený na myšlienke, že v každom registry od času $t = 101$ poznáme jeho 10 nasledujúcich taktovacích bitov. Za silný predpoklad znalosti 30 bitov, ktorý nám prinesie 30 lineárnych a navzájom nezávislých rovníc, musíme zaplatiť daň v podobe výpočtovej zložitosti 2^{30} časových jednotiek. Časovou jednotkou v Golićovom útoku rozumieme čas, ktorý je potrebný na vyriešenie sústavy lineárnych rovníc nad telesom $\mathbf{Z}_2$.

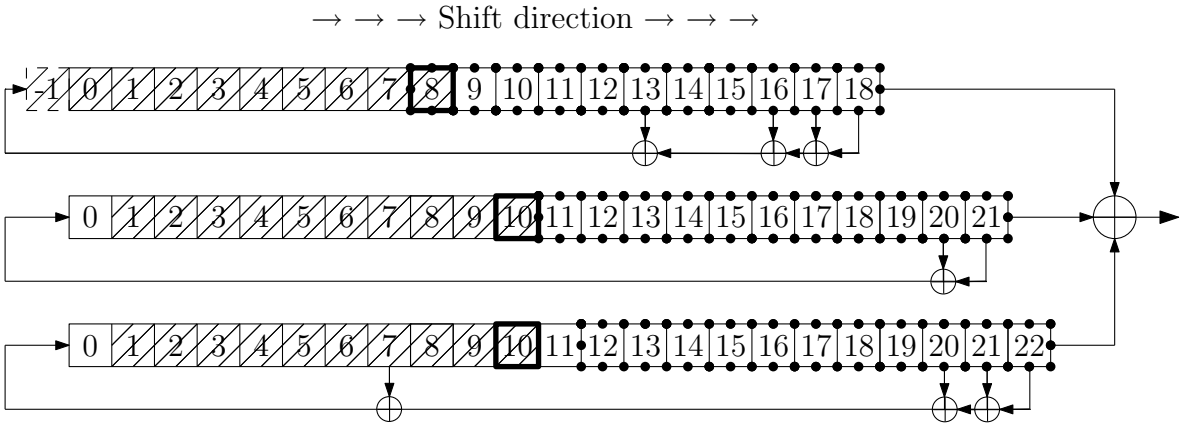
Vnútorný stav $R^{(101)}$ sme dostali tak, že sa najprv vyhodil výstupný bit vyprodukovaný zo stavu $R^{(100)}$ ako posledný krok inicializácie a až potom sa vykonal takt šifry, pomocou ktorého sme sa dostali zo stavu $R^{(100)}$ do stavu $R^{(101)}$. Potom nasleduje výpočet prvého bitu hesla. Preto ďalšiu jednu rovnicu získame jednoducho tak, že spočítame XOR výstupných bitov, ktoré sa majú rovnať prvému bitu hesla. Túto rovnicu nazývame *výstupnou rovnicou*. Po získaní rovnice $k_1 = R_1^{(101)}[18] \oplus R_2^{(101)}[21] \oplus R_3^{(101)}[22]$ sa vykoná takt šifry, čím prejdeme do nasledujúceho stavu $R^{(102)}$.

30 uhádnutých bitov sme zvolili tak, aby sme v najhoršom prípade nasledujúcich 10 taktov šifry vedeli, ktoré registre sa posunú. Najhorší prípad z hľadiska počtu získaných rovníc nastane vtedy, keď sa v každom takte posunú všetky tri registre. Tým by sme získali len ďalších 10 rovníc do našej sústavy. Najlepší prípad nastane vtedy, keď sa vždy posunú len dva registre, pričom všetky dvojice registrov by sa posunuli rovnaký počet. V tomto prípade by sme získali až 15 rovníc, dokým by sme prvýkrát nepoznali všetky taktovacie bity.

Odvodíme si priemerný počet takto získaných rovníc. V jednom takte šifry môžu nastať štyri možnosti posunu registrov. Môžu sa posunúť buď len registre R_1 a R_2, alebo len R_2 a R_3, alebo len R_1 a R_3, alebo sa posunú všetky tri registre. To znamená, že pravdepodobnosť, že sa v jednom takte šifry posunie daný register, je 3/4. Potom počet taktov šifry potrebný na to, aby sa daný register posunul 10-krát, je v priemere $4/3 \cdot 10$, čo je približne 13,33. Z tohto dôvodu budeme

uvažovať, že získame v priemere ďalších 13,33 lineárných a nezávislých rovníc. Pričom nezávislé sú vďaka dĺžke registrov, ktoré sú dostatočne dlhé na to, aby každá rovnica obsahovala novú premennú sústavy. Tieto rovnice sú v rovnakom tvare ako prvá *výstupná rovnica*, ktorú sme získali z výstupných bitov bez dovtedy žiadného vykonania taktu počas útoku. Formálne ich môžeme vyjadriť v tvare $x \oplus y \oplus z = k$, kde x, y, z sú premenné sústavy, ktoré sú na pozíciach výstupných bitov, $k \in \{0, 1\}$ a znak $\oplus$ značí bitovú operáciu XOR.

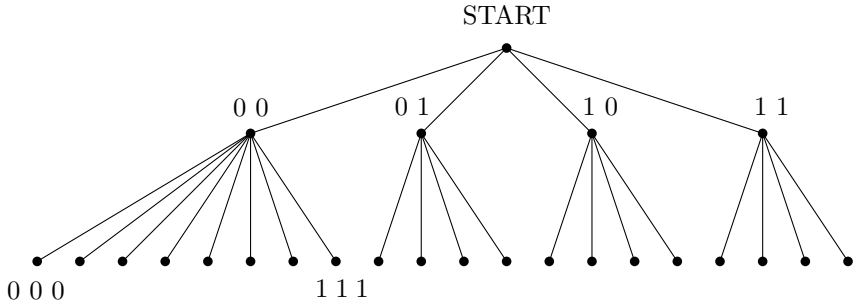
Pomocou 30 *hádaných bitov* získame informáciu o niekoľkých bitoch s najvyššími indexmi v každom registri. Pričom počet týchto bitov, o ktorých získame informáciu, závisí na počte posunutí jednotlivých registrov. Za predpokladu, že sa posunuli všetky tri registre rovnaký počet(10-krát), tak dostaneme určitú informáciu o 11 bitoch s najvyššími indexmi pre každý register. Tieto bity sú na obrázku 3.2 ohraničené bodkami.



Obr. 3.2: Vo vybodkovanej oblasti sú bity, o ktorých sme vďaka *hádaným bitom* získali určitú informáciu.

Všetky doteraz získané lineárne rovnice sú na sebe nezávislé, pretože každá rovnica obsahuje informáciu aspoň o jednom novom bite, ktorý sme nehádali. Celkovo zatiaľ máme v priemere $30 + 1 + 13,33 = 44,33$ lineárnych a navzájom nezávislých rovníc a zostáva nám v priemere získať zvyšných $63,32 - 44,33 = 18,99$ rovníc.

Teda v tomto momente nám chýba približne v priemere 19 rovníc, aby sme mohli získať obsah celého vnútorného stavu $R^{(101)}$. Miesto prehľadávania všetkých 2^{19} možností hrubou silou budeme vytvárať tzv. strom taktovacích bitov, ktorého časť je zobrazená na obrázku 3.3.



Obr. 3.3: Strom taktovacích bitov.

Každému vrcholu stromu(okrem vrcholu *START*) budeme postupne priraďovať nejaké ohodnotenie taktovacích bitov, ktoré sa v najbližšom takte stanú vstupom do funkcie *Maj*. V bode *START* sa ocitneme v čase, keď prvýkrát nebudeme poznať všetky aktuálne taktovacie bity, ktoré sme hádali na začiatku útoku. Na základe znalosti hodnôt taktovacích bitov môžeme vykonať takt šifry a presunúť sa o úroveň nižšie do ďalšieho vrcholu. Ak v danom vrchole po vykonaní taktu nepoznáme n nových taktovacích bitov, tak z vrcholu výjde(smerom dole) práve 2^n hrán. Obrázok 3.3 ukazuje príklad, keď vo vrchole *START* nepoznáme práve dva z troch taktovacích bitov. Preto z tohto vrcholu vychádzajú štyri možné ohodnotenia týchto dvoch bitov. Vidíme, že o úroveň nižšie sa môže vyskytnúť situácia, keď nebudeme poznať všetky tri taktovacie bity a preto sa príslušný vrchol rozvetví až na 8 hrán.

Teraz si spočítame priemerný počet hrán vychádzajúcich z vrcholu. S pravdepodobnosťou 3/4 sa v jednom takte posunú dva registre, čím dostaneme 2 nové bity ako vstup do funkcie *Maj*, čím máme 4 rôzne ohodnotenia dvojice bitov. Na druhej strane, keď sa posunú všetky tri registre, čo nastane s pravdepodobnosťou 1/4, dostaneme tri nové neznáme bity, čím máme 8 možností ohodnotenia troch bitov. Potom priemerný počet hrán vychádzajúcich z každého vrcholu je $3/4 \cdot 4 + 1/4 \cdot 8 = 5$.

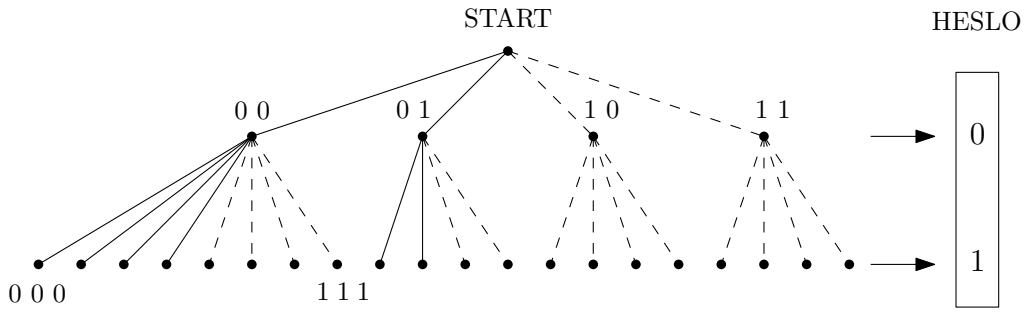
Hĺbka stromu závisí na danom stave registrov. Keby sa vo fáze pred vstupom do stromu všetky registre posúvali rovnomerne, hĺbka stromu by bola najmenšia. Na druhej strane, keby sa vždy posúvali len dva tie isté registre, tak by hĺbka stromu bola najväčšia, pretože by sme museli v stromovej štruktúre získať veľa informácií o bitoch neposúvaného registru. Pravdepodobne preto Golić odvodil hĺbku stromu za zjednodušujúcej podmienky, že v každom registri nepoznáme práve m bitov. Zároveň hĺbka stromu bude závislá na počte chýbajúcich rovníc, ktorých je v tomto momente v priemere 18,99. Aby sme teda získali informáciu o ďalších m neznámych bitov pre každý register, hĺbka stromu by mala byť $4/3 \cdot m$. Potom pre m, ktoré je približne $18,99/3 = 6,33$, je hĺbka stromu približne 8,44.

Počas pohybu po strome zároveň kontrolujeme, či pre dané ohodnotenie taktovacích bitov platí vzťah, že XOR výstupných bitov, ktorých výber je odvodený od daného ohodnotenia taktovacích bitov, je rovný hodnote príslušného bitu hesla. Ak áno, tak ďalej pokračujeme v prechádzaní po strome do najbližšieho vrcholu. V opačnom prípade zahodíme celý podstrom s vrcholom, kde je práve tá kombinácia taktovacích bitov, ktorá zapríčinila uvedenú nerovnosť. Pretože uvažujeme len prípady, pre ktoré XOR výstupných bitov súhlasí s heslom, čo nastane v priemere v polovici prípadov, tak vo výpočte celkovej zložitosti útoku budeme uvažovať, že každý vrchol má v priemere len dva a pol hrán. Obrázok 3.4 na ďalšej strane ilustruje príklad vyrezávania podstromov, ktoré nekorešpondujú s bitmi hesla. Stromy, ktoré sú vyrezávané, sú zobrazené čiarkovane a tie nemusíme prehľadávať.

Počas prechádzania stromu zároveň pridávame nové rovnice do sústavy. V momente, keď získame 64 lineárnych a navzájom nezávislých rovníc, získame vyriešením sústavy kandidáta na hľadaný vnútorný stav $R^{(101)}$.

Pretože každý vrchol obsahuje v priemere dva a pol hrán a hĺbka stromu je v priemere 8,44, tak počet všetkých možností, ktoré potrebujeme prehľadať v strome, je $2,5^{8,44}$, čo je približne $2^{11,16}$.

Postupným prehľadávaním stromu budeme získavať chýbajúce rovnice, až kým



Obr. 3.4: Orezávanie podstromov nesprávnych taktovacích bitov.

dosiahneme sústavu 64 lineárnych a na sebe nezávislých rovníc. Priemerná zložitosť získania takejto sústavy je $2^{30} \cdot 2^{11,16} = 2^{41,16}$. Pretože nám v priemere stačí prehľadať len polovicu všetkých možností, celková priemerná zložitosť tohto útoku je rovná vyriešeniu $2^{40,16}$ sústav lineárnych rovníc.

Zhrnieme si celý Golićov popis útoku do nasledujúceho pseudokódu. V algoritme budeme sústavu rovníc značiť M a ohodnotenie neznámych taktovacích bitov ako *ohodnotenie*.

ALGORITMUS 3.1

VSTUP: prvých 64 bitov hesla: $k_1, k_2, \ldots, k_{64}$
VÝSTUP: $R^{(101)} = (R_1^{(101)}, R_2^{(101)}, R_3^{(101)})$

1. `FOR EACH` *hádané bity* `DO`
   - 1.1 *Pridaj do sústavy 30 rovníc*(M, *hádané bity*),
   - 1.2 *Pridaj do sústavy rovnicu prvého bitu hesla*(M, k_1),
   - 1.3 `WHILE` poznáme všetky taktovacie bity `DO`
     - 1.3.1 *Vykonaj takt šifry*(*hádané bity*),
     - 1.3.2 *Pridaj ďalšiu rovnicu do sústavy*(M, $k_1, k_2, \ldots, k_{64}$),
   - 1.4 `FOR EACH` ohodnotenia neznámych taktovacích bitov `DO`
     - 1.4.1 *Strom taktovacích bitov*(*ohodnotenie*, M, $k_1, k_2, \ldots, k_{64}$),
2. `RETURN` $R_1^{(101)}, R_2^{(101)}, R_3^{(101)}$.

Vonkajším FOR-cyklom z kroku 1. prechádzame všetkých 2^{30} ohodnotení *hádaných bitov* postupne od ohodnotenia $(0, \ldots, 0)$, $(0, \ldots, 0, 1)$, až po $(1, \ldots, 1)$. Tento FOR-cyklus nám zvolí nejaké ohodnotenie 30 *hádaných bitov* a procedúra *Pridaj do sústavy 30 rovníc* z nich vytvorí odpovedajúcich 30 rovníc a doplní ich do sústavy. Potom do sústavy pridáme ešte jednu rovnicu, ktorú ako sme spomínali vyššie, získame prakticky zadarmo bez taktovania pomocou prvého bitu hesla k_1.

V kroku 1.3 nasleduje WHILE-cyklus, ktorým získame maximálny počet rovníc, ktoré nám 30 *hádaných bitov* poskytnú. Ako bolo uvedené, v priemere ich získame 13,33. Najprv vo WHILE-cykle vykonáme takt šifry, čím dostaneme nové výstupné bity a vďaka nim spolu s bitmi hesla vytvoríme ďalšiu rovnicu a pridáme ju do sústavy. V momente, keď prvýkrát nebudeme poznať všetky tri taktovacie bity, vstupujeme do stromovej štruktúry. Budeme sa nachádzať vo vrchole *START* a v kroku 1.4 budeme postupne prechádzať vetvy vychádzajúce z tohto vrcholu. Akým spôsobom si zvolíme poradie prechádzaných vetiev Golić

nepopisoval, preto si ho popíšeme až v kapitole o implementácii tohto útoku. Po vybratí vetvy stromu, ktorá korešponduje s príslušným ohodnotením neznámych taktovacích bitov sa pomocou rekurzívnej procedúry *Strom taktovacích bitov* posunieme hlbšie do stromu. Popis tejto procedúry je nižšie. Výsledkom procedúry je výpis kandidáta na hľadaný vnútorný stav, alebo v prípade, že vonkajší FOR-cyklus z kroku 1. nám zvolil nesprávne ohodnotenie *hádaných bitov* a prešli sme už všetky ohodnotenia neznámych taktovacích bitov z kroku 1.4, tak sa vrátime do kroku 1., kde sa zvolí ďalšie ohodnotenie *hádaných bitov*.

Teraz zhrnieme postup prechádzania stromu taktovacích bitov do pseudokódu s názvom *Strom taktovacích bitov*. Premenné použité v pseudokóde, budú *ohodnotenie*, M a $k_1, k_2, \ldots, k_{64}$ a navyše ešte budeme používať premennú *počet získaných rovníc*, ktorá ako názov naznačuje, bude vyjadrovať aktuálny počet rovníc sústavy v danom kroku výpočtu.

ALGORITMUS 3.2: *Strom taktovacích bitov*

VSTUP: *ohodnotenie*, M, $k_1, k_2, \ldots, k_{64}$
VÝSTUP: $R^{(101)} = (R_1^{(101)}, R_2^{(101)}, R_3^{(101)})$

1. *Vykonaj takt šifry*(*ohodnotenie*),
2. `IF` poznáme výstupné bity všetkých troch registrov `THEN`
   2.1 `IF` XOR výstupných bitov nekorešponduje s heslom `THEN`
      2.1.1 *BREAK*
3. *Pridaj ďalšie rovnice do sústavy*(M),
4. `IF` máme 64 lineárne nezávislých rovníc `THEN`
   4.1 *Spočítaj riešenie sústavy*(M),
   4.2 *Over správnosť riešenia*($k_1, k_2, \ldots, k_{64}$),
   4.3 `IF` riešenie je správne `THEN`
      4.3.1 `RETURN` $R^{(101)}$,
      `ELSE` *BREAK*,
   `ELSE` (nedostatok lineárne nezávislých rovníc)
      `FOR EACH` ohodnotenia neznámych taktovacích bitov `DO`
         Strom taktovacích bitov(*ohodnotenie*, M, $k_1, k_2, \ldots, k_{64}$).

Procedúra *Strom taktovacích bitov* začína vykonaním taktu šifry v kroku 1. To, aké registre sa posunú, závisí na ohodnoteniu neznámych taktovacích bitov. Po vykonaní taktu šifry v kroku 2. zistíme, či poznáme všetky výstupné bity registrov. To zistíme podľa počtu posunutí jednotlivých registrov. Ak sa R_1 posunul aspoň 10-krát, R_2 aspoň 11-krát a R_3 aspoň 12-krát, tak poznáme všetky výstupné bity registrov. Ak ich poznáme, môžeme na základe znalosti hesla porovnať, či XOR výstupných bitov je zhodný s aktuálnym bitom hesla. Ak sa tieto bity nezhodujú, tak sa vykoná príkaz *BREAK*, ktorý znamená, že program ukončí výpočet aktuálnej procedúry *Strom taktovacích bitov*. Príkazom *BREAK* sa zahodí celý podstrom nesprávnych kombinácií taktovacích bitov ako to znázorňuje obrázok 3.4 vyššie.

Zo zvoleného ohodnotenia neznámych taktovacích bitov, ktoré korešponduje s heslom, v kroku 3. odvodíme nové rovnice a pridáme ich do matice M. Golić tvar pridávaných rovníc vo svojom popise nespomína, preto celý mechanizmus pridávania rovníc popíšeme až v kapitole o implementácii. V kroku 4. sa pýtame,

či máme v sústave 64 lineárnych a na sebe nezávislých rovníc. Ak áno, v kroku 4.1 spočítame riešenie sústavy rovníc. Potom v kroku 4.2 overíme vďaka znalosti hesla správnosť tohto riešenia. V kroku 4.3 vypíšeme riešenie do výstupu, ak kandidát na hľadaný vnútorný stav je správny. V opačnom prípade príkazom *BREAK* vyskočíme z aktuálnej procedúry a výpočet pokračuje v úrovni vyššie zvolením ďalšieho ohodnotenia taktovacích bitov. Ak podmienka z kroku 4. neplatí, t.j. ešte nemáme 64 lineárnych a na sebe nezávislých rovníc, zvolíme si ďalšie ohodnotenie neznámych taktovacích bitov a vstúpime o úroveň hlbšie do stromu rekurzívnym zavolaním procedúry *Strom taktovacích bitov*.

V nasledujúcej kapitole si predstavíme rôzne verzie implementácie Golićovho útoku a podrobnejšie vysvetlíme jednotlivé kroky algoritmu. V jednotlivých implementáciach útoku využívame implementáciu šifry *A5/1*, ktorá je inšpirovaná z [2].

3.2 Implementácia útoku na vnútorný stav $R^{(101)}$

V nasledujúcej kapitole si predstavíme implementáciu Golićovho útoku na vnútorný stav $R^{(101)}$. Počas implementácie sme narazili na pár problémov, ktoré sa dajú riešiť rôznym spôsobom. Preto predstavíme viac verzií implementácie, ktoré budú riešiť tieto problémy špecifickým spôsobom. V závislosti na tomto riešení sa môže radikálne zmeniť rýchlosť programu. Základným spomínaným problémom je tvar rovníc, ktoré sa pridávajú do sústavy počas prechádzania stromu. V tejto kapitole si predstavíme niekoľko modifikácií Golićovho útoku a navzájom ich porovnáme. Najprv uvedieme dve verzie, ktoré sa budú líšiť v tvare pridávaných rovníc. Tento tvar rovníc ovplyvní aj štruktúru stromu. Tieto dve verzie sú veľmi pomalé, pretože v 4.kroku algoritmu *Strom taktovacích bitov* sa celá matica kopíruje do pomocnej matice, na ktorej sa zisťuje počet lineárne nezávislých rovníc. Tieto verzie slúžia hlavne na porovnanie počtu nadbytočných lineárne závislých rovníc.

Potom si uvedieme jedno vylepšenie pochádzajúce od autorov Pornina a Sterna a toto vylepšenie použijeme v poslednej verzii implementácie tohto útoku, kde v 4. kroku algoritmu 3.2 už nebude treba kopírovať celú maticu a celkový výpočet bude prebiehať podstatne rýchlejšie.

Pretože algoritmy prvých dvoch verzií implementácie sa líšia len v tvare pridávaných rovníc a v kroku 1.4 Algoritmu 3.1, podrobný popis jednotlivých krokov algoritmu bude predstavený len v podkapitole 1.verzie útoku. V podkapitole pre 2.verziu implementácie preto popis jednotlivých krokov algoritmu vynecháme a spomenieme len dva rozdiely medzi týmito verziami: typ pridávaných rovníc počas prechádzania stromom, krok 1.4 Algoritmu 3.1.

V poslednej verzii implementácie, kde využívame Porninovo a Sternovo vylepšenie, opäť nebudeme opisovať celý popis algoritmu, pretože sa oproti 2.verzii implementácie líši len v úprave rovníc pred pridaním do sústavu.

3.2.1 Implementácia 1.verzie útoku a jej výsledky

Implementácia 1.verzie útoku oproti ostatným verziám sa snaží čo najmenej modifikovať Golićov popis útoku. Pretože podľa Golićovho popisu stromu každý

vrchol(okrem vrcholu $START$) obsahuje nejaké ohodnotenie neznámych taktovacích bitov, tak rovnice, ktoré získavame počas prechádzania stromom, budú mať nasledujúci tvar: $t = k$, kde t je taktovací bit a $k \in \{0, 1\}$ je jeho hodnota. Tento jednoduchý tvar budú mať rovnice len na začiatku útoku, dokým nie je potrebné na vyjadrenie bitov registrov použiť spätnú väzbu. Jednotlivé vetvy vychádzajúce z daného vrcholu smerom hlbšie do stromu, budú spájať vrcholy, ktoré sa budú líšiť iným ohodnotením neznámych taktovacích bitov. Strom taktovacích bitov bude mať potom rovnaký tvar, ako je v predchádzajúcej kapitole na obrázku 3.3.

Základným rozdielom medzi jednotlivými verziami Golićovho útoku je tvar rovníc, ktoré sa pridávali do sústavy počas prechádzania stromovej štruktúry. Golić vo svojom popise tvar týchto rovníc neuvádza. Našim cieľom je odvodiť čo najviac lineárne nezávislých rovníc zo znalosti posunutia registrov v najbližšom takte.

Pretože sa v jednom takte posunú dva alebo tri registre, tak počas prechádzania stromom budeme v každom takte pridávať do sústavy dve, alebo tri rovnice tvaru $t = k$. Pokiaľ nebudeme poznať aspoň jeden výstupný bit, tak do sústavy budeme pridávať aj *výstupnú rovnicu*. V prípade, že budeme poznať všetky výstupné bity, tak *výstupnú rovnicu* už nebudeme do sústavy pridávať, pretože bude lineárne závislá na ostatných rovniciach. V tomto prípade *výstupnú rovnicu* budeme používať len na vyrezávanie nesprávnych podstromov.

Počas prechádzania stromom taktovacích bitov budeme v 1.verzii útoku pridávať rovnice v týchto tvaroch:

$$t = k_1, \tag{3.1}$$

$$x \oplus y \oplus z = k_2, \tag{3.2}$$

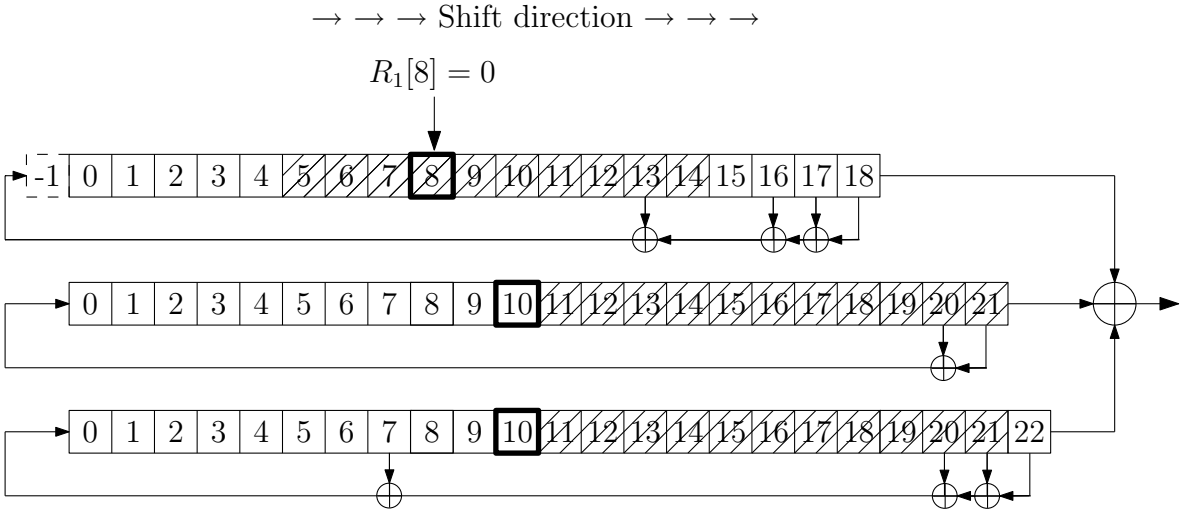
kde t je taktovací bit, x, y, z sú výstupné bity a k_1, k_2 sú konštanty.

Pseudokód 1.verzie útoku bude totožný so pseudokódom Algoritmu 3.1, preto ho nebudeme znovu uvádzať. Podrobnejšie si ale vysvetlíme krok 1.4 Algoritmu 3.1, t.j. akým spôsobom pracujeme s neznámymi taktovacími bitmi. Vysvetlíme si, v akom poradí budeme prechádzať vetvy z aktuálneho vrcholu, resp. ako si zoradíme ohodnotenia neznámych taktovacích bitov, ktoré príslušia daným vetvám.

Pretože chceme začať s vyrezávaním stromu čo najskôr, tak ako prvé budeme od zoradenia ohodnotení požadovať, aby sa počas vykonávania taktu posúvali tie registre, u ktorých nepoznáme ich výstupné bity. Ďalej budeme požadovať, aby sa posunuli registre, ktorým chýba najväčší počet posunutí do toho, aby sme poznali ich výstupné bity.

Uveďme zoradenie ohodnotení neznámych taktovacích bitov na príklade. Nech napríklad nepoznáme taktovacie bity registrov R_2 a R_3 a hodnota taktovacieho bitu registru R_1 je 0. Ďalej nech poznáme výstupný bit registra R_2 a aby sme poznali výstupný bit registra R_1, musel by sa posunúť 4-krát a aby sme poznali výstupný bit registra R_3, musel by sa posunúť raz. Táto situácia je ilustrovaná na obrázku 3.5 na ďalšej strane, kde známe bity sú zvýraznené vyšrafovaním.

Potom zoradenie ohodnotení neznámych taktovacích bitov bude nasledujúce: $(0, 0)$, $(1, 0)$, $(0, 1)$, $(1, 1)$, kde prvá zložka dvojice je taktovací bit registra R_2 a druhá zložka je taktovací bit registra R_3. Ohodnotenie $(0, 0)$ je na prvom mieste preto, lebo vďaka nemu sa posunú všetky registre. Ohodnotenie $(1, 0)$ je na druhom mieste, pretože sa v tomto prípade posunú tie registre, u ktorých nepoznáme



Obr. 3.5: Príklad vnútorného stavu.

ich výstupné bity. Taktovací bit registra R_1 je rovnaký ako taktovací bit registra R_3 a je rôzny od taktovacieho bitu registru R_2. Nakoniec ohodnotenie $(0, 1)$ má prednosť pred ohodnotením $(1, 1)$, pretože register R_1 potrebuje väčší počet posunov k tomu, aby sme poznali jeho výstupný bit.

Pseudokód procedúry *Strom taktovacích bitov* pre 1. verziu útoku bude rovnaký ako pseudokód Algoritmu 3.2, preto ho nebudeme znovu uvádzať. Podrobnejšie si vysvetlíme niektoré kroky tejto procedúry, ktoré súvisia s 1.verziou útoku. V kroku 3. budeme pridávať rovnice, ktoré budú v tvare (3.1) alebo (3.2). Rovnice týchto typov budú po určitom čase meniť svoj tvar. To je spôsobené tým, že s rastúcim počtom vykonaných taktov, rastú aj indexy taktovacích a výstupných bitov a tieto bity prestávajú byť premennými sústavy. Preto po určitom čase musíme rovnice získané v strome prepísať pomocou premenných sústavy. Pomôžu nám k tomu nasledujúce rovnice odvodené od rovnice spätnej väzby:

$$\begin{aligned} R_1^{(101)}[i] &= R_1^{(101)}[14+i] \oplus R_1^{(101)}[17+i] \oplus R_1^{(101)}[18+i] \oplus R_1^{(101)}[19+i], \\ i &= -1, -2, \ldots, -14. \end{aligned} \tag{3.3}$$

$$\begin{aligned} R_2^{(101)}[i] &= R_2^{(101)}[21+i] \oplus R_2^{(101)}[22+i], \\ i &= -1, -2, \ldots, -21. \end{aligned} \tag{3.4}$$

$$\begin{aligned} R_3^{(101)}[i] &= R_3^{(101)}[8+i] \oplus R_3^{(101)}[21+i] \oplus R_3^{(101)}[22+i] \oplus R_3^{(101)}[23+i], \\ i &= -1, -2, \ldots, -8. \end{aligned} \tag{3.5}$$

V prípade potreby sa rovnice (3.3), (3.4) a (3.5) môžu použiť rekurzívne. Napríklad v prípade výpočtu hodnoty $R_1^{(101)}[-15]$ rovnicu (3.3) použijeme dvakrát:

$$R_1^{(101)}[-15] = R_1^{(101)}[-1] \oplus R_1^{(101)}[2] \oplus R_1^{(101)}[3] \oplus R_1^{(101)}[4] =$$

$$= (R_1^{(101)}[13] \oplus R_1^{(101)}[16] \oplus R_1^{(101)}[17] \oplus R_1^{(101)}[18]) \oplus R_1^{(101)}[2] \oplus R_1^{(101)}[3] \oplus R_1^{(101)}[4].$$

Až do vstupu do stromovej štruktúry bola matica rovníc pomerne riedka. Ale počas pridávania rovníc v priebehu stromovej štruktúry sa počet jednotiek zvyšuje.

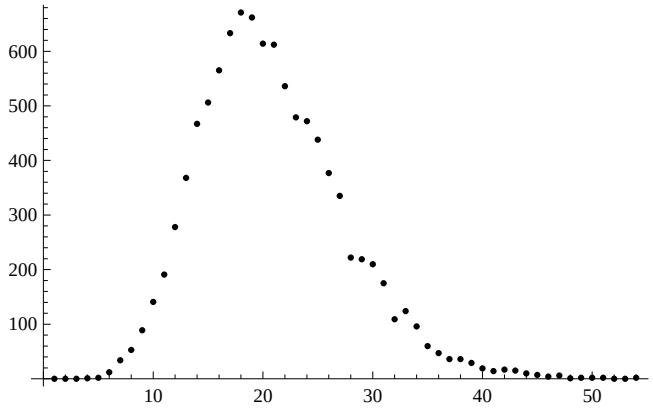
Pred vysvetlením kroku 4. sa pozrime, čo sa deje so sústavou rovníc, keď sa po strome vraciame o hladinu vyššie. V každom volaní procedúry *Strom taktovacích bitov*, ak sa výpočet dostane do kroku 3., pridávame niekoľko rovníc do matice. Tieto rovnice nemusia byť správne, pretože sú odvodené od ohodnotenia neznámych taktovacích bitov a to sa "háda". Keď z akéhokoľvek dôvodu ukončíme aktuálnu procedúru *Strom taktovacích bitov*, tak pred jej ukončením musíme vrátiť maticu do stavu, v akom bola o hladinu vyššie. To znamená, že pri pridávaní rovníc si budeme musieť pamätať indexy riadkov pridávaných rovníc, aby sme mohli rovnice pridané v aktuálnom volaní procedúry vymazať. Na konci procedúry a pred každým výskytom príkazu *BREAK* (okrem toho, ktorý sa nachádza v kroku 2.1.1), keď vymažeme rovnice, dostaneme sústavu do stavu, v akom bola o jednu hladinu stromu vyššie. Pred príkazom *BREAK*, ktorý sa nachádza v kroku 2.1.1, nemusíme mazať žiadne rovnice, lebo sme v tom čase ešte žiadne do sústavy nepridali. Týmto spôsobom sa do procedúry o hladinu vyššie dostane matica, ktorá nebola ovplyvnená nižšími nesprávnymi vetvami stromu.

Po pridaní rovníc sa dostávame do kroku 4., ktorý pri implementácii spôsoboval najväčšie problémy. Počas prechádzania stromu sa do sústavy pridávajú aj rovnice, ktoré sú lineárne závislé na ostatných. Existencia lineárne závislých rovníc spôsobí, že potrebujeme do sústavy pridať viac ako 64 rovníc. Pretože na získanie celého vnútorného stavu $R^{(101)}$ potrebujeme 64 nezávislých rovníc, musíme pokračovať v prechádzaní stromu a pridávať rovnice dovtedy, pokým dosiahneme 64 nezávislých rovníc. Preto ak budeme počas prechádzania stromu pridávať do sústavy rovnice typu (3.1) alebo (3.2), tak aby sme dostali 64 lineárne nezávislých rovníc, hĺbka stromu bude väčšia, ako v priemere 8,44, ako uvádza Golić. Rovnice typu (3.1) a (3.2) je potrebné neskôr počas útoku modifikovať pomocou spätnej väzby. Čím je počet nadbytočných lineárne závislých rovníc väčší, tým hlbšie musíme vojsť do stromu a tým sa nám zvyšuje zložitosť výpočtu. Počet závislých rovníc, ktoré získame do doby, pokým dostaneme 64 nezávislých rovníc, závisí od daného vnútorného stavu.

Pred prezentovaním samotných výsledkov algoritmu si ešte popíšeme kroky 4.1 a 4.2 v *Strome taktovacích bitov*. V kroku 4.1 počítame riešenie sústavy rovníc pomocou známej metódy Gaussovej eliminácie. Táto metóda upraví rovnicu na trojuholníkový tvar, z ktorého ľahko dostaneme riešenie - kandidáta na hľadaný vnútorný stav. V kroku 4.2 overíme správnosť tohto kandidáta tak, že ho postupne necháme taktovať. Po každom takte porovnáme vyprodukovaný bit hesla so správnym bitom *hesla*. Akonáhle nájdeme prvú nezhodu, kandidáta zahodíme a ukončíme aktuálne volanie procedúry *Strom taktovacích bitov*.

Teraz si predstavíme výsledky testovania. Na vzorke 10 000 náhodne vygenerovaných vnútorných stavov sme počítali počet nadbytočných lineárne závislých rovníc, t.j. takých, že po ich odobratí ostane v sústave len 64 navzájom nezávislých rovníc. Výpočet prebiehal tak, že do *hádaných bitov* sme priradili správne hodnoty a počas prechádzania stromu sme išli vždy po správnej vetve. Nasledujúci graf na obrázku 3.6 na ďalšej strane znázorňuje počet vnútorných stavov, ktoré majú príslušný počet nadbytočných lineárne závislých rovníc.

Najmenší počet nadbytočných lineárne závislých rovníc vyšiel 3. Tento počet dosiahol len jeden stav zo vzorky 10 000 vnútorných stavov. Najväčší počet nadbytočných lineárne závislých rovníc je až 53. Tento počet dosiahli 2 stavy z 10 000. Vnútorné stavy oboch extrémov nájdeme v prílohe A. Priemerný počet



Obr. 3.6: Vzťah počtu vnútorných stavov a počtu nadbytočných lineárne závislých rovníc. Na zvislej osi je počet vnútorných stavov a na vodorovnej osi je počet nadbytočných lineárne závislých rovníc.

nadbytočných lineárne závislých rovníc podľa testovania vychádza približne 20,8.

V dôsledku existencie lineárne závislých rovníc, budeme musieť krok 4. v procedúre *Strom taktovacích bitov* modifikovať. Dopredu nevieme povedať, koľko rovníc budeme potrebovať, aby sme dostali 64 lineárne nezávislých rovníc. Jednou možnosťou je po každom pridaní jednej rovnice do sústavy otestovať hodnosť matice. Potom by sme krok 4. ponechali nezmenený. Táto možnosť je ale veľmi nepraktická, pretože zisťovanie hodnosti matice, napr. pomocou Gaussovej eliminácie, je veľmi výpočtovo náročné. Preto radšej využijeme výsledok z testovania: priemerný počet nadbytočných lineárne závislých rovníc je 20,8. Krok 4. potom modifikujeme tak, že sa v ňom budeme pýtať, či máme v sústave 85 rovníc, t.j. 64 lineárne nezávislých a približne 21 nadbytočných lineárne závislých. Podľa výsledkov testovania by sme vo väčšine prípadov mali získať hľadaný vnútorný stav. Nevýhodou tohto postupu je, že nemáme zaručené, že dostaneme výsledok. Na druhej strane, keby sme v kroku 4. miesto 85 rovníc požadovali vyšší počet, tak na jednej strane by sme zväčšili pravdepodobnosť získania hľadaného vnútorného stavu, ale na druhej strane by bol výpočet v strome príliš pomalý.

Na rovnakej vzorke 10 000 náhodne vygenerovaných vnútorných stavov sme rovnakým postupom zisťovali priemernú hĺbku stromu, ktorá vyšla približne až 17, čo je podstatne viac, ako 8,44 z Golićovho popisu. Potom počet všetkých možností v strome, ktoré potrebujeme prehľadať, je $2,5^{17}$ čo je približne $2^{22,47}$ a to je podstatne horší výsledok, ako $2^{11,16}$, ktorý je uvedený v popise útoku.

Nakoniec sme si náhodne vygenerovali 100 vnútorných stavov, na ktorých získanie netreba viac ako 80 rovníc a testovali sme, ako dlho trvá výpočet v strome taktovacích bitov. Pred vstupom do stromu sme opäť do 30-tích *hádaných bitoch* priradili správne hodnoty, aby sme sa dočkali výsledku v podstatne kratšom čase. Priemerný čas výpočtu vychádzal rádovo v minútach. Ak nepoznáme hodnoty *hádaných bitov*, tak čas výpočtu v strome musíme prenásobiť číslom 2^{30}, z čoho vychádza, že celkový výpočtový čas 1.verzie útoku je príliš vysoký a v praxi nepoužiteľný.

3.2.2 Implementácia 2.verzie útoku a jej výsledky

Teraz si predstavíme 2.verziu útoku na vnútorný stav $R^{(101)}$, ktorá využíva iný typ rovníc, ktorý uviedli Pornin a Stern v článku [8]. Nasledujúca verzia sa oproti 1.verzii líši len v inom tvare rovníc, ktoré sa pridávajú počas prechádzania stromom v kroku 3. procedúry *Strom taktovacích bitov*.

Na to, aby sme vedeli, ktoré registre sa v danom takte posunú, nie je potrebné poznať ich konkrétne hodnoty, ako bolo uvažované v 1.verzii. Stačí poznať len "XOR-y" taktovacích bitov. Vysvetlime si to podrobnejšie na príklade. Nech sa počas taktu majú napríklad posunúť prvé dva registre a tretí sa nemá posunúť. Túto situáciu jednoznačne popisujú nasledujúce dve rovnice:

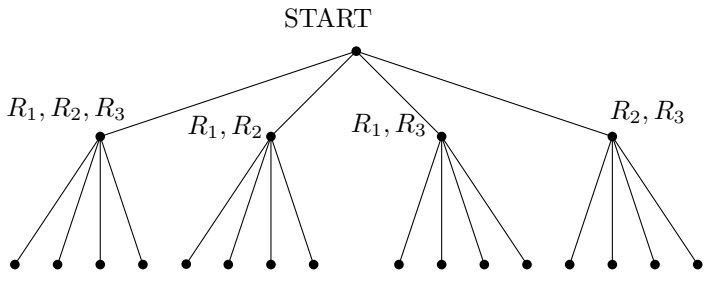
$$\begin{aligned} t_1 \oplus t_2 &= 0 \\ t_1 \oplus t_3 &= 1, \end{aligned}$$

kde t_1, t_2 a t_3 sú po rade taktovacie bity registrov R_1, R_2 a R_3.

Je triviálne odvodiť tvary rovníc pre všetky možnosti posunov registrov. Sú uvedené v nasledujúcej tabuľke.

posunú sa registre:	tvar rovníc:
R_1, R_2	$t_1 \oplus t_2 = 0,\ t_1 \oplus t_3 = 1$
R_1, R_3	$t_1 \oplus t_2 = 1,\ t_1 \oplus t_3 = 0$
R_2, R_3	$t_1 \oplus t_2 = 1,\ t_1 \oplus t_3 = 1$
R_1, R_2, R_3	$t_1 \oplus t_2 = 0,\ t_1 \oplus t_3 = 0$

Pretože nebudeme hádať jednotlivé hodnoty neznámych taktovacích bitov ako v 1.verzii, ale len ich "XOR-y," zmení sa štruktúra celého stromu. Z každého vrcholu stromu budú vychádzať(smerom hlbšie do stromu) vždy 4 vetvy, pretože sú 4 možnosti posunu registrov: buď sa posunú všetky tri registre, alebo sa neposunie len register R_1, alebo len R_2, alebo len R_3. Vo vrchole $START$ sa budeme nachádzať v čase, keď prvýkrat nebudeme poznať všetky taktovacie bity. Príklad takéhoto stromu je na obrázku 3.7, kde pri vrcholoch sú označené registre, ktoré sa budú v nasledujúcom takte posúvať.



Obr. 3.7: Strom "XOR-ov" taktovacích bitov.

S pridávaním *výstupnej rovnice* budeme pracovať rovnako, ako v 1. verzii. Pokiaľ nebudeme poznať aspoň jeden výstupný bit, tak do sústavy budeme pridávať aj *výstupnú rovnicu*. V prípade, že budeme poznať všetky výstupné bity,

tak *výstupnú rovnicu* nebudeme pridávať, pretože bude lineárne závislá na ostatných. V tomto prípade *výstupnú rovnicu* budeme používať len na vyrezávanie nesprávnych podstromov.

Počas prechádzania stromom taktovacích bitov budeme v 2.verzii útoku pridávať nasledovné rovnice:

$$x \oplus y = k_1, \tag{3.6}$$

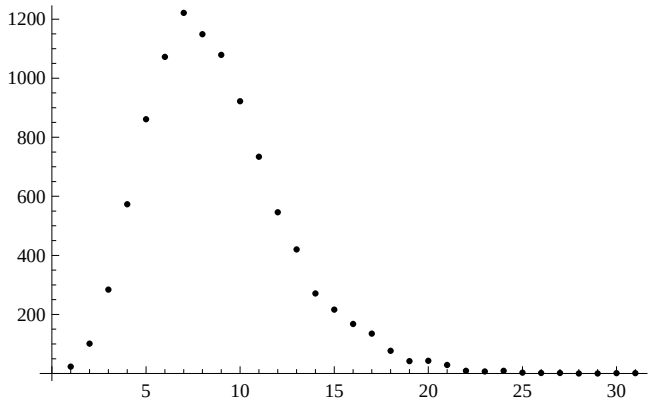
$$x \oplus z = k_2, \tag{3.7}$$

$$a \oplus b \oplus c = k_3, \tag{3.8}$$

kde x, y, z sú taktovacie bity, a, b, c sú výstupné bity a k_1, k_2, k_3 sú konštanty. Tieto rovnice bude opäť počas útoku potrebné modifikovať pomocou spätnej väzby.

Pseudokódy Algoritmov 3.1 a 3.2 budú mať pre túto verziu rovnaký tvar. Rovnako ako v 1.verzii si v kroku 1.4 Algoritmu 3.1 usporiadáme vetvy vychádzajúce z vrcholu *START* tak, aby sme mohli začať čo najskôr s vyrezávaním stromov. Aj v tomto prípade budeme FOR-cyklom prechádzať všetky vetvy vychádzajúce z vrcholu *START*. Jediný rozdiel je v tom, že v 2. verzii tieto vetvy nebudú korešpondovať s konkrétnymi hodnotami neznámych taktovacích bitov, ale budú korešpondovať s jednou zo 4 možností posunutia registrov. Potom na základe výberu konkrétnej možnosti posunutia registrov vykonáme v Algoritme 3.2(ktorý v tejto je verzii vhodnejšie pomenovať *Strom XOR-ov taktovacích bitov*) takt šifry a výpočet pokračuje rovnako, ako v 1.verzii.

Teraz si predstavíme výsledky testovania. V tejto verzii implementácie, rovnako ako v prvej, sa vyskytli nadbytočné lineárne závislé rovnice. Náhodne sme si vygenerovali 10 000 vnútorných stavov a zisťovali sme počet týchto nadbytočných lineárne závislých rovníc. Testovanie prebiehalo tak, že sme si do *hádaných bitov* dosadili ich správne hodnoty a počas prechádzania stromom sme išli len po správnej vetve. Graf na obrázku 3.8 znázorňuje počet vnútorných stavov, ktoré majú príslušný počet nadbytočných lineárne závislých rovníc.



Obr. 3.8: Vzťah počtu vnútorných stavov a počtu nadbytočných lineárne závislých rovníc. Na zvislej osi je počet vnútorných stavov a na vodorovnej osi je počet nadbytočných lineárne závislých rovníc.

Priemerný počet nadbytočných lineárne závislých rovníc pre túto verziu vyšiel približne 8,73, čo je výrazné zlepšenie oproti minulej verzii. Najmenší počet nadbytočných lineárne závislých rovníc na vzorke 10 000 vnútorných stavov bol

0 a dosiahlo ho 23 stavov. Najväčší počet nadbytočných lineárne závislých rovníc vyšiel 30 a dosiahol ho len jeden vnútorný stav. Príklad oboch extrémnych prípadov nájdeme v prílohe A.

Keďže aj v tejto verzii sa vyskytujú nadbytočné lineárne závislé rovnice, budeme musieť, podobne ako v 1.verzii, upraviť krok 4. Algoritmu 3.2. V kroku 4. sa budeme pýtať, či máme v sústave aspoň 73 rovníc. Číslo 73 dostaneme sčítaním 64 nezávislých rovníc s 8,44 nadbytočnými lineárne závislými rovnícami po zaokrúhlení nahor. Vo väčšine prípadov, t.j. keď počet nadbytočných lineárne závislých rovníc je najviac 9, dostaneme správny výsledok. Opäť môžeme zväčšiť hodnotu 73 z kroku 4., čím na jednej strane zvýšime pravdepodobnosť nájdenia hľadaného vnútorného stavu, ale na druhej strane bude výpočet v strome pomalší.

Na rovnakej vzorke 10 000 vnútorných stavov sme spočítali priemerný počet taktov vykonaných v strome a ten vyšiel 12. Je to síce viac, ako 8,44 ako predpokladal Golić, ale je to omnoho lepší výsledok oproti 1.verzii.

Ďalej sme na vzorke 100 náhodne generovaných vnútorných stavov počítali priemerný čas behu programu, ktorý vyšiel rádovo v sekundách. Pri testovaní sme opäť do *hádaných bitov* dosadili správne hodnoty, aby sme sa dočkali výsledku v krátkom čase.

Výsledky testovania tejto verzie sú podstatne lepšie, ako v 1.verzii, ničmenej, ešte stále nie sú dostatočne dobré na to, aby sme mohli pustiť kompletný program, t.j. bez predpokladu znalosti 30 *hádaných bitov* a len s predpokladom znalosti bitov *hesla* $k_1, \ldots, k_{64}$.

3.2.3 Porninovo a Sternovo urýchlenie algoritmu

V tejto podkapitole si predstavíme urýchlenie algoritmu, ktoré pochádza od autorov: Thomas Pornin a Jacques Stern. Tí ho predstavili v článku [8], v ktorom tiež uviedli typ rovníc, ktorý je základom 2.verzie útoku. Pripomeňme, že sa jedná o rovnice (3.6), (3.7) a (3.8), kde prvé dve rovnice popisujú posun registrov a tretia je *výstupná rovnica*. Porninovo a Sternovo vylepšenie podstatne urýchľuje výpočet a šikovne obchádza problém, ktorý spomaluje predchádzajúce verzie útoku. Spomenutým problémom je existencia lineárne závislých rovníc, ktoré sa vyskytujú počas prechádzania stromom. V celej tejto podkapitole vychádzame z [8] a budeme uvažovať 2.verziu útoku, na ktorej popíšeme aplikovanie urýchlenia algoritmu.

Počas celého behu algoritmu, t.j. od voľby 30 *hádaných bitov* až po prechádzanie stromu "XORov" taktovacích bitov, pridávame do sústavy rovnice, ktoré keď upravíme určitým spôsobom, tak budú vytvárať maticu, ktorej homogénna časť bude mať nasledujúcu vlastnosť (V): pre každú rovnicu n, existuje taká premenná, ktorej koeficient v rovnici n je 1 a v každej z nasledujúcich rovníc(t.j. rovníc $n+1, n+2, ...$) je jej koeficient 0.

Spôsob, akým budeme upravovať každú rovnicu, si prezradíme neskôr. Najskôr si uveďme príklad matice, ktorá má vlastnosť (V):

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Označme si premenné tejto sústavy ako $x_1, x_2, \ldots, x_6$, kde premenná x_i prísluší i-tému stĺpci matice. Prvá rovnica spĺňa podmienku uvedenú vo vlastnosti (V) vďaka premennej x_1. Vidíme, že pod touto premennou v stĺpci sú samé nuly. Rovnako aj druhá rovnica spĺňa podmienku uvedenú vo vlastnosti (V) vďaka premennej x_6. Opäť pod touto premennou v stĺpci sú samé nuly. Analogickým postupom overíme zvyšné rovnice.

Využitie vlastnosti (V) je spomínaným trikom, ktorý má urýchliť celý program. Keď budeme mať v sústave 64 rovníc a vlastnosť (V) bude splnená, tak máme zaručené, že v matici sa nachádzajú len lineárne nezávislé rovnice.

Teraz si predstavíme postup, ktorým budeme upravovať pridávané rovnice. Úprava spočíva vo vykonaní Gaussovej eliminácie na pridávanej rovnici vzhľadom k predchádzajúcim rovniciam. Označme si u_i premennú, ktorej koeficient je 1 v rovnici i a 0 vo všetkých rovniciach j, kde $j > i$. Potom na pridávanú rovnicu n aplikujeme nasledujúci algoritmus:

Uprav rovnicu

VSTUP: matica M
VÝSTUP: matica M s novou upravenou rovnicou

1. označme si pridávanú rovnicu ako X,
2. `FOR` $i = 1$ `TO` $n - 1$ `DO`
   2.1 `IF` u_i má koeficient 1 v rovnici X `THEN`
      2.1.1 pričítaj rovnicu i k rovnici X
3. pridaj rovnicu X do sústavy
4. nájdi prvý nenulový koeficient v rovnici X a označ u_n príslušnú premennú.

Vďaka existencii lineárne závislých rovníc, ktorú sme testovaním overili v predchádzajúcej kapitole, vieme, že v poslednom kroku sa nám nie vždy podarí nájsť nenulový koeficient. Keď sa rovnica X v kroku 2. celá vynuluje, tak to znamená, že je lineárne závislá na ostatných a nebudeme ju pridávať do sústavy. V takom prípade pokračujeme vo výpočte. Môže však nastať prípad, že rovnica X bude v homogénnej časti matice nulová a v pravej strane matice bude jednotka. V takomto prípade dostávame spor a sústava nemôže mať riešenie. Inými slovami, dostali sme sa v strome do nesprávnej vetvy, ktorú sme detekovali pomocou bitov *hesla* a musíme z tejto vetvy preč. To vykonáme vymazaním rovníc, ktoré sme v tejto vetve pridali do sústavy a prechodom v strome o hladinu vyššie, kde si zvolíme ďalšiu vetvu. Po prechode o hladinu vyššie má aktuálna matica opäť vlastnosť (V), čo znamená, že na ňu nemusíme aplikovať Gaussovu elimináciu, v čom spočíva hlavná výhoda tohto vylepšenia.

Keď obdržíme 64 lineárne nezávislých rovníc, môžeme vyriešiť sústavu rovníc

a získať kandidáta na hľadaný vnútorný stav šifry *A5/1*. Pri výpočte používame Gaussovu metódu riešenia sústavy lineárnych rovníc. Túto metódu používame podobne, ako pri riešení matice rovníc v trojuholníkovom tvare. Posledná rovnica musí mať tvar

$$x = k_1, \tag{3.9}$$

kde x je premenná sústavy a k_1 je konštanta z $\{0, 1\}$. Keby sa v rovnici objavili dve premenné, tak homogénna časť matice by nesplňovala vlastnosť (V). Predposledná rovnica bude v tvare

$$y + k_2 x = k_3, \tag{3.10}$$

kde y je ďalšia premenná sústavy a k_2 a k_3 sú konštanty. Ak už budeme poznať hodnotu x, budeme poznať aj hodnotu y. Opäť predposledná rovnica nemôže obsahovať viac ako dve premenné, inak by bola porušená vlastnosť (V). Týmto spôsobom pokračujeme až po prvú rovnicu a tak obrdžíme 64 bitov hľadaného vnútorného stavu.

Po získaní kandidáta je potrebné vykonať overovaciu fázu: kandidát na hľadaný vnútorný stav bude postupne produkovať výstupné bity a budeme ich porovnávať so správnymi bitmi $k_1, \ldots, k_{64}$ hesla. Pornin a Stern v svojom článku navrhujú overiť správnosť kandidáta iným spôsobom a tvrdia, že je efektívnejší. Tento spôsob overovania kandidáta sme neimplementovali, ale pre zaujímavosť si ho popíšeme.

Po dosiahnutí kompletnej matice navrhujú pokračovať ďalej v eliminácii pridávaných rovníc. Každá pridávaná rovnica buď bude vynulovaná, alebo bude v spore s *heslom*. Len jedna zo štyroch možných vetiev stromu poskytne dve rovnice, ktoré sa vynulujú. Je tomu tak preto, že keď už budeme mať 64 rovníc, tak budeme mať informáciu o celom vnútornom stave a teda aj o posunoch registrov. Tretia rovnica v uvažovanej vetve bude *výstupná rovnica* a tá bude závisieť na *výstupnom bite*. Preto po úprave sa s pravdepodobnosťou 1/2 vynuluje a s rovnakou pravdepodobnosťou bude v spore s *heslom*. Takže v priemere musíme pokračovať ešte dva kroky v prechádzaní stromu, t.j. musíme upraviť 6 ďalších rovníc, aby sme overili správnosť kandidáta na hľadaný vnútorný stav.

Pornin a Stern vo svojom článku [8] uvádzajú, že celková zložitosť útoku je v priemere $2^{44,3}$. Pričom výpočtová jednotka je Gaussova eliminácia jednej rovnice podľa v priemere 64 predošlých lineárnych rovniciach. Ďalej tvrdia, že zložitosť ich verzie útoku je rovnaká, ako uvádza Golić, t.j. málo nad 2^{40}, pretože ich výpočtová jednotka je realistickejšia. Ich implementácii útoku trvalo 200 dní, aby získali hľadaný vnútorný stav, pričom používali špeciálny hardware Compaq-XP1000.

3.2.4 Implementácia Porninovho a Sternovho vylepšenia

Pseudokód verzie s využitím Porninovho a Sternovho vylepšenia je prakticky rovnaký ako u 2.verzie útoku. Existujú len dva rozdiely, na ktoré je potrebné upozorniť. Prvý rozdiel je v tom, že pred každým pridaním rovnice do matice sa vykoná procedúra *Uprav rovnicu*. Druhý rozdiel je v tom, že v tejto verzii presne vieme, kedy má začať overovacia fáza kandidáta na hľadaný vnútorný

stav. V predošlej verzii sme nevedeli, kedy bude mať matica hodnosť 64. Mimo tieto dva rozdiely je postup analogický s 2.verziou, preto pseudokód algoritmu vynechávame.

Algoritmus sme testovali na 10 000 náhodne generovaných vnútorných stavoch. Opäť sme za *hádané bity* dosadili ich správne hodnoty a počas prechádzania stromu sme išli len po správnej vetve. Čas výpočtu sme začali merať akonáhle sme vstúpili do stromovej štruktúry. Priemerný čas, kým sme dostali hľadaný vnútorný stav, vyšiel v priemere 66 milisekúnd. Tento čas je podstatne lepší, ako v 2.verzii bez využitia Porninovho a Sternovho vylepšenia. Je tomu tak preto, že sa zbytočne viackrát nevykonáva Gaussova eliminácia na správnych rovniciach.

Ak by sme pustili kompletnú kryptoanalýzu šifry *A5/1* s cieľom získať $R^{(101)}$ len zo znalosti 64 bitov *hesla*, museli by sme 66 milisekúnd vynásobiť hodnotou 2^{30}, aby sme dostali celkový výpočetný čas, ktorý by vyšiel približne 800 dní. Algoritmus sa vďaka FOR-cyklu z kroku 1, kde za *hádané bity* dosadzujeme postupne všetky možné hodnoty 30 bitov, dá rozdeliť na ľubovoľný počet podúloh. Keby sme tieto podúlohy riešili paralélne na viacerých počítačoch mohli by sme dosiahnúť výsledky aj v praktickejšom čase.

3.3 Problém s existenciou lineárne závislých rovníc

Počas prechádzania stromom sa do matice pridávajú aj lineárne závislé rovnice, či už má strom štruktúru z 1.verzie alebo, z 2. verzii útoku. V 1.verzii podľa testovania vyšiel priemerný počet nadbytočných lineárne závislých rovníc 20,8. V 2. verzii ich vyšlo v priemere 8,73. Existencia lineárne závislých rovníc neovplyvní hĺbku stromu, ale ovplyvní počet riadkov matice, ak tieto rovnice nebudeme eliminovať. Pornin-Sternovo vylepšenie nám ukázalo postup, ako lineárne závislé rovnice eliminovať, ale za cenu Gaussovej eliminácie na každom jednom riadku. Mechanizmus vzniku lineárne závislých rovníc je nám neznámy. Ukážeme si na príklade vznik lineárne závislých rovníc.

Uvažujme štruktúru stromu a tvar rovníc popísané v 1. verzii implementácie. Majme nasledujúci vnútorný stav $R^{(101)}$.

R_1	0	1	0	1	0	1	0	0	1	1	1	0	1	1	1	0	0	1	0
R_2	0	1	1	1	0	0	0	0	0	1	1	0	0	1	0	0	1	0	1	1	1	0
R_3	1	1	1	0	0	0	0	1	0	1	1	0	0	0	1	0	0	1	0	1	0	1	0

Obr. 3.9: Bity vnútorného stavu $R^{(101)}$.

Na tomto vnútornom stave sme pustili algoritmus z 1.verzie útoku. Do *hádaných bitov* sme dosadili ich správne hodnoty a počas prechádzania stromom sme išli vždy po správnej vetve. To znamená, že všetky pridávané rovnice sú správne. Algoritmus sme zastavili v momente, keď sa do sústavy pridala prvá nadbytočná lineárne závislá rovnica. Táto rovnica po prepísaní do premenných sústavy má tvar:

$$R_2[17] \oplus R_2[16] = 1$$

Uvedená prvá nadbytočná lineárne závislá rovnica sa uložila do 58. riadku, takže prvých 57 rovníc bolo na sebe navzájom nezávislých. Nasleduje minimálna podmnožina rovníc sústavy, z ktorých sa dá odvodiť uvažovaná nadbytočná lineárne závislá rovnica:

$$\begin{aligned}
R_3[4] &= 0 \\
R_3[6] &= 0 \\
R_1[18] \oplus R_2[20] \oplus R_3[21] &= 0 \\
R_1[18] \oplus R_2[19] \oplus R_3[20] &= 1 \\
R_1[16] \oplus R_2[17] \oplus R_3[18] &= 0 \\
R_1[16] \oplus R_2[16] \oplus R_3[17] &= 0 \\
R_2[1] \oplus R_2[19] &= 0 \\
R_3[21] \oplus R_3[20] \oplus R_3[19] \oplus R_3[6] &= 0 \\
R_3[19] \oplus R_3[18] \oplus R_3[17] \oplus R_3[4] &= 0
\end{aligned}$$

Matica uvažovaného vnútorného stavu po dokončení algoritmu spolu obsahovala 4 nadbytočné lineárne závislé rovnice. Zvyšné tri nadbytočné lineárne závislé rovnice spolu s príslušnými minimálnymi podmnožinami rovníc sústavy sú v prílohe B reprezentované v bitovom zápise.

3.4 Poznámky k útoku na vnútorný stav $R^{(101)}$

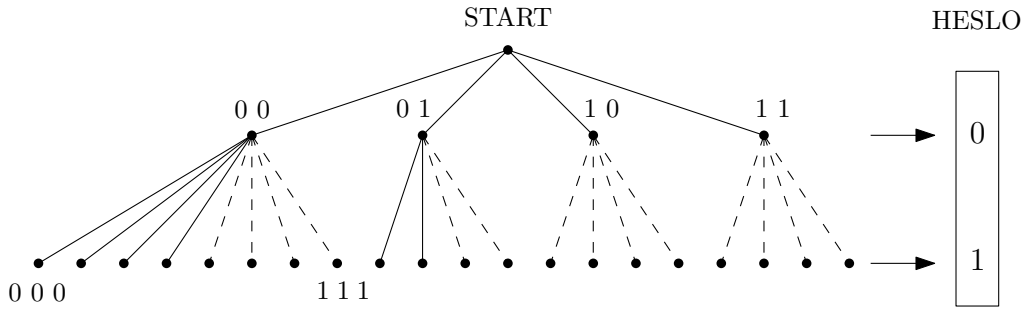
V tejto podkapitole sa najprv pokúsime obhájiť použitie rekurzie počas prechadzánia stromu. Potom si uvedieme jedno pozorovanie, ktoré nebolo započítané do celkovej priemernej zložitosti $2^{40,16}$, ktorú uvádza Golić.

Procedúra *Strom taktovacích bitov* je rekurzívna. V kryptoanalýze sa všeobecne použitie rekurzie príliš nedoporučuje, pretože môže viesť k pomalším výpočtom. Tieto pomalšie výpočty prevažne vznikajú z dôvodu, že pri každom volaní procedúry sa musia znovu vytvoriť všetky parametre procedúry. Napríklad ak jedným z parametrov je matica typu 100x100, tak ak sa procedúra volá často, viditeľne to môže zvýšiť časovú aj pamäťovú zložitosť algoritmu. Ale v našom prípade potrebujeme poznať nastavenie parametrov z predchádzajúcej iterácie, pretože sa často stáva, že zvolený parameter *ohodnotenie* je nesprávny a musíme sa vrátiť o hladinu vyššie.

Druhým, pomerne častým dôvodom, prečo nepoužívať rekurziu je ten, že sa v nej môžu niekoľkokrát vykonávať rovnaké výpočty. Tento problém v procedúre *Strom taktovacích bitov* taktiež nehrozí, pretože každá vetva stromu je unikátna, obsahuje vždy inú informáciu o posune registrov a na základe tejto informácie sa menia všetky výpočtu v procedúre.

Teraz uvedieme menšiu úpravu v zložitosti v Golićovom popise útoku. Poznamenajme, že vyrezávanie podstromov môžeme vykonávať iba za podmienky, že poznáme všetky výstupné bity v danom čase. Ak aspoň jeden z nich nepoznáme, nemôžeme vylúčiť žiadnu kombináciu taktovacích bitov a musíme ich otestovať

všetky. Ako vidieť na obrázku 3.2, aj keby sa tretí register prvých 10 taktov vždy posunul, nemali by sme žiadnu informáciu o bite $R_3^{(101)}[11]$. Tento bit sa stane výstupným až po najbližšom takte, ktorý bude vykonaný počas prechádzania stromu. A pretože ho nepoznáme, nemôžeme vylúčiť žiadnu kombináciu a musíme počítať so všetkými. Obrázok 3.10 ilustruje situáciu, že s vyrezávaním nesprávnych podstromom môžeme začať až v druhej hladine.



Obr. 3.10: Orezávanie podstromov nesprávnych taktovacích bitov.

Potom sa zložitosť upraví nasledujúco. Priemerný počet hrán vychádzajúcich z vrcholu *START* je 5 a priemerný počet hrán vychádzajúcich z ostaných vrcholov už bude 2,5, pretože už môžeme poznať všetky výstupné bity. Z tohto dôvodu počet všetkých možností, ktoré budeme prehľadávať v strome je $5 \cdot 2,5^{7,44}$, čo je približne $2^{12,16}$. Čím sa celková priemerná zložitosť útoku upraví na $2^{41,16}$.

3.5 Útok na počiatočný vnútorný stav $R^{(0)}$

V predchádzajucej podkapitole sme si ukázali útok na vnútorný stav šifry $R^{(101)}$, t.j. stav, ktorý vyprodukuje prvý bit hesla. V tejto časti si ukážeme, ako získať *počiatočný vnútorný stav* $R^{(0)}$ zo stavu $R^{(101)}$. Pripomeňme ešte, že od času $t = 0$ sa vykonáva takt šifry už s využitím *Majoritnej funkcie*.

Nech $c_i = \{c_i(t)\}_{t=1}^{\infty}$ označuje postupnosť bitov, ktoré charakterizujú posunutie registra $R_i, i = 1, 2, 3$, v čase t. Pričom $c_i(t) = 1$, ak sa R_i v čase t posunul a $c_i(t) = 0$, ak sa R_i v čase t neposunul. Čo sa myslí tým, že R_i sa v čase t posunul? Nech vnútorný stav je v čase $t - 1$. Vykoná sa takt, v ktorom sa napr. posunú prvé dva registre a tretí sa neposunie. Po vykonaní tohto taktu sa vnútorný stav bude nachádzať v čase t a bude platiť $c_1(t) = 1, c_2(t) = 1$, a $c_3(t) = 0$.

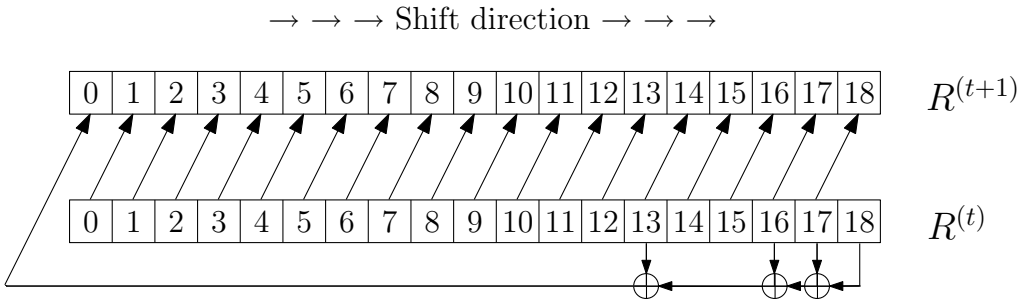
Zaujíma nás hodnota výrazu $\sum_{j=1}^{t} c_i(j)$, ktorý udáva počet posunutí registra R_i od času 0 do času t. K tomu, aby sme získali *počiatočný vnútorný stav* $R^{(0)}$ z vnútorného stavu $R^{(101)}$, nám stačí poznať pre každý register R_i hodnotu práve spomínaného výrazu $\sum_{j=1}^{t} c_i(j)$. Pretože pravdepodobnosť posunutia registra je 3/4, tak v 101 prípadoch sa register posunie v priemere $3/4 \cdot 101 \approx 76$-krát . Uvažované pravdepodobnostné rozdelenie je binomické, preto pre každý register R_i má štandardná odchylka hodnotu $\sqrt{101 \cdot 3/4 \cdot (1 - 3/4)} \approx 4,35$.

Takže každý register R_i sa do času $t = 101$ posunul s najväčšou pravdepodobnosťou 76-krát. S druhou najväčšou pravdepodobnosťou sa do času $t = 101$ posunul 75-krát. Potom s treťou najväčšou pravdepodobnosťou sa posunul 77-krát atď. Keď budeme v hľadaní $R^{(0)}$ postupovať podľa tejto zostupnej postupnosti

pravdepodobností odvodenej od binomického rozdelenia pre každý register nezávisle, tak [5] uvádza, že nájdeme správne počty posunutí R_i do času $t = 101$ s vysokou pravdepodobnosťou do 10^4 pokusov a v najhoršom prípade do 10^6 pokusov.

Pre každý takýto odhad počtu posunutí registrov $R_i, i = 1, 2, 3$, do času $t = 101$ získame spätnou lineárnou rekurziou z vnútorného stavu $R^{(101)}$ *počiatočný vnútorný stav* $R^{(0)}$. Správnosť nášho kandidáta na $R^{(0)}$ potom otestujeme tak, že 101-krát necháme vykonať takt šifry a potom výsledný vnútorný stav porovnáme s $R^{(101)}$.

Najprv sa pozrieme, ako pomocou lineárnej spätnej rekurzie môžme získať *počiatočný vnútorný stav*. Konkrétne sa pozrieme na prvý register R_1, u ostatných registrov bude postup analogický. Majme register $R_1^{(t+1)}$ v nejakom čase $t + 1$ a chceme najprv získať obsah registra $R_1^{(t)}$, pričom predpokladáme, že $c_1(t+1) = 1$. Obrázok 3.11 ilustruje, ako prebieha posunutie registra R_1.



Obr. 3.11: Posunutie registra R_1.

Z obrázku 3.11 môžeme odvodiť, že pre prvých 18 zložiek registra $R_1^{(t)}$ platí

$$R_1^{(t)}[i] = R_1^{(t+1)}[i + 1], i = 0, \ldots, 17. \tag{3.11}$$

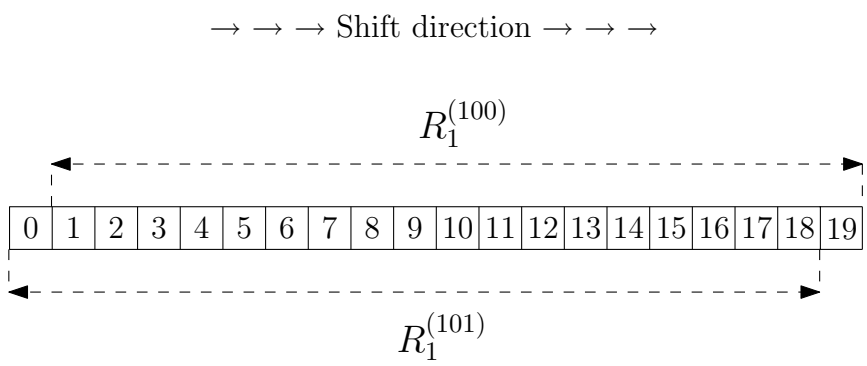
Z rovnice spätnej väzby $R_1^{(t)}[18] \oplus R_1^{(t)}[17] \oplus R_1^{(t)}[16] \oplus R_1^{(t)}[13] = R_1^{(t+1)}[0]$ získame posledný neznámy člen $R_1^{(t)}[18]$ a s použitím (3.11) si ho vieme vyjadriť pomocou zložiek registra $R_1^{(t+1)}$:

$$R_1^{(t)}[18] = R_1^{(t+1)}[0] \oplus R_1^{(t+1)}[14] \oplus R_1^{(t+1)}[17] \oplus R_1^{(t+1)}[18]. \tag{3.12}$$

Podobne, ako sme každú zložku $R_1^{(t)}[i]$ dokázali popísať zložkami $R_1^{(t+1)}[j]$, tak aj každú zložku $R_1^{(t-1)}[i]$ sme schopní popísať pomocou $R_1^{(t)}[j]$ a teda aj zložkami $R_1^{(t+1)}[j]$. Pri výpočte $R_1^{(0)}$ môžeme využívať rovnice (3.11) a (3.12) rekurzívnym spôsobom, t.j. budú sa rekurzívne volať až dovtedy, pokým budeme schopní nejakú zložku registra $R_1^{(0)}[i]$ vyjadriť zložkami registra $R_1^{(101)}$, ktorého obsah poznáme.

Takýto spôsob získania *počiatočného vnútorného stavu* $R^{(0)}$ však nie je príliš praktický, pretože by sa vykonalo zbytočne veľa rovnakých operácií. Pri implementácii útoku na $R^{(0)}$ sme postupovali nasledujúco. Uvažujme prvý register R_1, u ostatných registrov bude postup analogický. Uvažujme pásku P_1 s políčkami, v ktorých budú uložené bity. Políčka sú indexované zľava od nuly postupne prirodzenými číslami. Na prvých 19-tích políčkach si uložíme bity prvého registra

$R_1^{(101)}$ v čase $t = 101$. Na 20-tom políčku si uložíme hodnotu $R_1^{(100)}[18]$, ktorú dostaneme zo vzorca (3.12). Potom bity s indexami 1 až 20 budú bitmi registru $R_1^{(100)}$ v čase $t = 100$, viď obrázok 3.12.



Obr. 3.12: Páska pre register R_1.

Podobne pokračujeme ďalej a pomocou nasledujúcej rovnice, ktorá je odvodená od rovnice spätnej väzby, si vypočítame hodnoty políčok na páske s indexmi 20 a vyššími:

$$P_1[i] = P_1[i-19] \oplus P_1[i-1] \oplus P_1[i-2] \oplus P_1[i-5], i = 20, 21, \ldots \tag{3.13}$$

Potom register $R_1^{(99)}$ bude v páske uložený na políčkach s indexmi 2 až 20, register $R_1^{(98)}$ bude v páske uložený na políčkach s indexmi 3 až 21 atď. Keď budeme potrebovať tvar registra v nejakom čase $t \leq 101$, jednoducho si nájdeme v páske index prvého bitu hľadaného registru a nasledujúcich 18 bitov budú zvyšné bity hľadaného registru.

Rovnakým spôsobom budeme postupovať pre registre R_2 a R_3. V prípade registra R_2 bude na prvých 22 políčok pásky P_2 uložený register $R_2^{(101)}$ v čase $t = 101$. Zvyšné hodnoty políčok si spočítame pomocou rovnice:

$$P_2[i] = P_2[i-22] \oplus P_2[i-1], i = 22, 23, \ldots \tag{3.14}$$

Konečne v prípade registra R_3 si na prvých 23 políčkach pásky P_3 uložíme bity registra $R_3^{(101)}$ v čase $t = 101$. Zvyšné hodnoty políčok si spočítame pomocou rovnice:

$$P_3[i] = P_3[i-23] \oplus P_3[i-1] \oplus P_3[i-2] \oplus P_3[i-15], i = 23, 24, \ldots \tag{3.15}$$

Predtým, ako si predstavíme samotný pseudokód algoritmu na získanie *počiatočného vnútorného stavu*, ukážeme si, ako dostaneme zostupnú postupnost pravdepodobností počtov posunutí registrov od času $t = 0$ do času $t = 101$. Chceme si vytvoriť tabuľku(alebo súbor), označme si ho pod pojmom *tabuľka*, kde budú zoradené posuny registrov podľa pravdepodobnosti. Prvá zložka poľa, nech má index 1, bude obsahovať najpravdepodobnejšiu trojicu počtu posunutí (76, 76, 76). V druhej zložke bude uložená druhá najpravdepodobnejšia trojica počtu posunutí (75,76,76), atď.

Nech X, Y, Z sú náhodné veličiny s binomickým rozdelením, ktoré nadobúdajú hodnoty $k = 0, 1, \ldots, 101$ s pravdepodobnosťami $\binom{101}{k}(\frac{3}{4})^k(\frac{1}{4})^{101-k}$. Náhodná veličina X udáva počet posunutí registra R_1, náhodná veličina Y udáva počet posunutí registra R_2 a konečne náhodná veličina Z udáva počet posunutí registra R_3. Pravdepodobnosť, že sa register R_1 od času $t = 0$ do času $t = 101$ posunie k-krát, si označme p_k^1. Analogicky si pre register R_2, resp. R_3 označme pravdepodobnosti p_l^2, resp. p_m^3, že sa príslušný register od času $t = 0$ do času $t = 101$ posunie l-krát, resp. m-krát. Pretože náhodné veličiny X, Y a Z sú nezávislé, tak pravdepodobnosť, že sa register R_1 posunie k-krát a zároveň register R_2 l-krát a zároveň register R_3 m-krát, je rovná hodnote $P(X = k) \cdot P(Y = l) \cdot P(Z = m) = p_k^1 \cdot p_l^2 \cdot p_m^3$.

Tabuľka s trojicami počtov posunutí nezávisí na zadanom vnútornom stave $R^{(101)}$, preto si ju môžeme dopredu spočítať, pred samotným útokom na *počiatočný vnútorný stav*. Môžeme to vykonať jednoducho tak, že cez všetky hodnoty $k, l, m \in \{0, \ldots, 101\}$ si spočítame hodnotu $p_k^1 \cdot p_l^2 \cdot p_m^3$. Pre každú túto hodnotu pravdepodobnosti si zapamätáme trojicu počtu posunutí (k, l, m) a nakoniec tieto pravdepodobnosti zoradíme od najväčšej. Vo väčšine prípadov netreba spočítať hodnotu $p_k^1 \cdot p_l^2 \cdot p_m^3$ pre všetky možné počty posunov $0, \ldots, 101$, pretože veľmi nízke počty posunov a zároveň veľmi vysoké počty posunov, sú veľmi nepravdepodobné.

Ak už máme dopredu pripravenú tabuľku, kde sú zoradené trojice počtu posunutí podľa ich pravdepodobnosti, môžeme začať s kryptoanalýzou *počiatočného vnútorného stavu* $R^{(0)}$. Algoritmus bude mať dva vstupy. Prvým vstupom je register $R^{(101)}$ a druhým vstupom je *tabuľka*. Trojicu počtov posunutí z tabuľky budeme označovať ako (q_i^1, q_i^2, q_i^3), kde q_i^j udáva počet posunutí registra R_j, $j = 1, 2, 3$, v i-tej najpravdepodobnejšej trojici počtov posunutí. Nasleduje pseudokód algoritmu.

ALGORITMUS 3.3

VSTUP: register $R^{(101)}$, *tabuľka*
VÝSTUP: stav $R^{(0)}$

1. *pokračuj* = TRUE, i=1,
2. *Vytvoríme pásky* P_1, P_2, P_3,$(R^{(101)})$,
3. `WHILE` (*pokračuj*) `DO`
   - 3.1 do R_1 priraď kandidáta na $R_1^{(0)}(P_1\ q_i^1)$,
   - 3.2 do R_2 priraď kandidáta na $R_2^{(0)}(P_2\ q_i^2)$,
   - 3.3 do R_3 priraď kandidáta na $R_3^{(0)}(P_3\ q_i^3)$,
   - 3.4 `FOR` $j = 1$ `TO` 101 `DO`
     - 3.4.1 $Clock(R_1, R_2, R_3)$,
   - 3.5 `IF` $(R_1, R_2, R_3) = R^{(101)}$ `THEN`
     - 3.5.1 *pokračuj* = FALSE,

   `ELSE` $i = i + 1$,
4. `RETURN` $R^{(0)}$.

V kroku 1. si inicializujeme pomocné premenné: *pokračuj* dáva informáciu, že sme ešte nenašli *počiatočný vnútorný stav*; premenná i udáva index riadku *tabuľky*. V druhom kroku si pomocou vzorcov (3.13), (3.14), (3.15) vytvoríme pásky P_1, P_2, P_3, z ktorých ľahko dostaneme požadovaný tvar registrov . Hlavným pilierom Algoritmu 3.3 je WHILE-cyklus z kroku 3., ktorý sa opakuje dovtedy, pokým

nenájdeme *počiatočný vnútorný stav*. V krokoch 3.1, 3.2 a 3.3 si z i-tého riadku *tabuľky* vyberieme uvažovanú trojicu počtu posunutí (q_i^1, q_i^2, q_i^3) a na základe tejto trojice si nájdeme na jednotlivých páskach hľadaný tvar príslušného registra. Potom tento kandidát vykoná 101 taktov a overí sa jeho správnosť porovnaním so stavom $R^{(101)}$. V prípade, že je kandidát správny, WHILE-cyklus sa ukončí priradením hodnoty FALSE do premennej *pokračuj*. V opačnom prípade zvýšime index riadku v *tabuľke* a na základe novej trojice posunutí získame nového kandidáta a overujeme jeho správnosť.

Algoritmus 3.3 sme otestovali na náhodne vygenerovaných 100 vnútorných stavov. Predpokladali sme, že tieto stavy sú v čase $t = 101$ a chceli sme získať ich *počiatočný vnútorný stav*. Dopredu sme si vygenerovali *tabuľku* trojíc posunutí. Nebolo potrebné ju generovať celú, pretože načastejší počet posunutí bol v rozmedzí 71 až 81, čo súhlasí s hodnotou smerodatnej odchylky 4.35. Po vygenerovaní *tabuľky* sme spustili samotný výpočet rekonštrukcie *počiatočného vnútorného stavu* $R^{(0)}$. Priemerný čas výčtu vyšiel rádovo v sekundách. Nakoniec poznamenajme, že rekonštrukcia $R^{(0)}$ je omnoho rýchlejšia ako získanie $R^{(101)}$.

4. Korelačný útok od Ekdahla a Johanssona

Korelačný útok patrí medzi najdôležitejšie útoky na prúdové šifry. Okrem kryptoanalýzy šifry *A5/1* sa úspešne používa aj v mnohých ďalších prúdových šifrách. Predstavíme si korelačný útok pochádzajúci od dvoch švédskych autorov Patrika Ekdahla a Thomasa Johanssona, ktorí ho prezentovali v článku [9] v roku 2001. V tejto kapitole budeme vychádzať z tohto článku a uvedieme si v nej podrobný popis a poznámky k našej implementácii.

Pretože sa opäť jedná o útok typu *known-plaintext attack*, budeme predpokladať znalosť *hesla*. V tomto útoku budeme potrebovať toľko bitov *hesla* a to pre viacero rámcov, aby bol zabezpečený približne 5 minutový hovor. Prvým cieľom tohto útoku je získať *inicializačný vnútorný stav* $R^{(0)}$. Z neho nakoniec odvodíme *tajný kľúč*, ktorý sa používa v inicializačnej fáze šifry.

4.1 Idea útoku

Uvedieme si jednoduchú ideu korelačného útoku, ktorá sa stane podkladom konečného útoku na šifru *A5/1*. Najprv si zadefinujeme pár základných pojmov. *Tajný kľúč* a n-té *číslo rámca*, obe známe z inicializačnej fázy, si po rade označme $K = (k_1, \ldots, k_{68})$ a $F_n = (f_1, f_2, \ldots, f_{22})$, kde $k_i, f_i \in \mathbb{Z}_2$.

Ďalej nech $r_1(t)$, $t = 0, 1, \ldots$, označuje postupnosť výstupných bitov registru R_1 vzniknutých pri pravidelnom posúvaní, t.j. bez použitia *Majoritnej funkcie*. Prvý člen $r_1(0)$ vyprodukuje register R_1, keď sa nachádza v *inicializačnom vnútornom stave*. Podobne nech $r_2(t)$, $t = 0, 1, \ldots$, označuje postupnosť výstupných bitov druhého registra R_2 vzniknutú pravidelným posúvaním. A nakoniec nech $r_3(t)$, $t = 0, 1, \ldots$, označuje postupnosť výstupných bitov tretieho registra R_3 vzniknutú pravidelným posúvaním. Potom výraz $(r_1(0), r_1(1), \ldots, r_1(18))$ popisuje register $R_1^{(0)}$, t.j. prvý register *inicializačného vnútorného stavu*. Podobne výraz $(r_2(0), r_2(1), \ldots, r_2(21))$ popisuje register $R_2^{(0)}$ a výraz $(r_3(0), r_3(1), \ldots, r_3(22))$ popisuje register $R_3^{(0)}$.

Uvedomme si, že z algoritmu inicializačnej fázy *hesla* vyplýva, že *inicializačný vnútorný stav* môže byť vyjadrený lineárnou funkciou veličín K a F_n. Presnejšie povedané, každý výstupný bit $r_1(t), t \geq 0$, registra R_1 si môžeme vyjadriť v tvare

$$r_1(t) = \bigoplus_{i=1}^{64} a_{it}^1 k_i \oplus \bigoplus_{i=1}^{22} b_{it}^1 f_i, \tag{4.1}$$

každý výstupný bit $r_2(t), t \geq 0$, registra R_2 si môžeme vyjadriť ako

$$r_2(t) = \bigoplus_{i=1}^{64} a_{it}^2 k_i \oplus \bigoplus_{i=1}^{22} b_{it}^2 f_i, \tag{4.2}$$

nakoniec každý výstupný bit $r_3(t), t \geq 0$, registra R_3 si môžeme vyjadriť v tvare

$$r_3(t) = \bigoplus_{i=1}^{64} a_{it}^3 k_i \oplus \bigoplus_{i=1}^{22} b_{it}^3 f_i, \tag{4.3}$$

kde $a_{it}^1, a_{it}^2, a_{it}^3 \in \mathbb{Z}_2, i = 1, \ldots, 64$ a $b_{it}^1, b_{it}^2, b_{it}^3 \in \mathbb{Z}_2, i = 1, \ldots, 22$, sú známe konštanty.

Všimnime si, že prvá suma vo vzorcoch (4.1),(4.2) a (4.3) je lineárnou kombináciou bitov *tajného kľúča* a druhá suma je lineárnou kombináciou bitov *čísla rámca*. Aby sme tieto dve sumy od seba oddelili a zjednodušili zápis, zavedieme si pre každý register nasledujúce označenie:

$$\sigma_1(t) = \bigoplus_{i=1}^{64} a_{it}^1 k_i \quad \text{a} \quad \phi_1(t) = \bigoplus_{i=1}^{22} b_{it}^1 f_i, \ t \geq 0,$$
$$\sigma_2(t) = \bigoplus_{i=1}^{64} a_{it}^2 k_i \quad \text{a} \quad \phi_2(t) = \bigoplus_{i=1}^{22} b_{it}^2 f_i, \ t \geq 0,$$
$$\sigma_3(t) = \bigoplus_{i=1}^{64} a_{it}^3 k_i \quad \text{a} \quad \phi_3(t) = \bigoplus_{i=1}^{22} b_{it}^3 f_i, \ t \geq 0,$$

Tým pádom si môžeme vzorce (4.1), (4.2) a (4.3) prepísať do nasledujúcich prehľadnejších vzťahov:

$$r_1(t) = \sigma_1(t) \oplus \phi_1(t), \ t \geq 0. \tag{4.4}$$
$$r_2(t) = \sigma_2(t) \oplus \phi_2(t), \ t \geq 0, \tag{4.5}$$
$$r_3(t) = \sigma_3(t) \oplus \phi_3(t), \ t \geq 0. \tag{4.6}$$

Detailnejšie sa pozrime na postupnosti $\sigma_i(t)$ a $\phi_i(t)$, $i = 1, 2, 3$. Pretože *číslo rámca* je verejné, môžeme predpokladať, že poznáme všetky prvky postupností $\phi_i(t)$, $i = 1, 2, 3$. Na druhej strane bity *tajného kľúča* nepoznáme, preto nepoznáme ani prvky postupností $\sigma_i(t)$, $i = 1, 2, 3$. Ďalej si uvedomme, že postupnosti $\phi_i(t)$, $i = 1, 2, 3$, závisia na danom rámci. Ale postupnosti $\sigma_i(t)$, $i = 1, 2, 3$ sú rovnaké pre všetky rámce. Toto posledné pozorovanie budeme využívať v korelačnom útoku.

Základnú ideu korelačného útoku na *A5/1* si najlepšie ukážeme na nasledujúcom príklade. Uvažujme *inicializačný vnútorný stav* $R^{(0)}$. Musí sa vykonať 101 taktov[1], aby sme získali prvý bit *hesla*. Označme si ho h_1 a pripomeňme, že predpokladáme jeho znalosť.

Predpokladajme, že počas 101 taktov sa každý register posunul napríklad 79-krát. Tento predpoklad si označme ako A. Dostávame *výstupnú rovnicu*

$$r_1(79) \oplus r_2(79) \oplus r_3(79) = h_1. \tag{4.7}$$

Výstupnú rovnicu (4.7) si prepíšeme podľa (4.4), (4.5) a (4.6) do nasledujúceho vzťahu

$$(\sigma_1(79) \oplus \sigma_2(79) \oplus \sigma_3(79)) \oplus (\phi_1(79) \oplus \phi_2(79) \oplus \phi_3(79)) = h_1. \tag{4.8}$$

[1] vykoná sa 100 taktov ako záver inicializačnej fázy a potom sa vykoná ešte jeden takt, ktorý vyprodukuje prvý bit hesla

Separovaním známych a neznámych hodnôt získame takýto vzťah:

$$\sigma_1(79) \oplus \sigma_2(79) \oplus \sigma_3(79) = \phi_1(79) \oplus \phi_2(79) \oplus \phi_3(79) \oplus h_1. \tag{4.9}$$

Pravú stranu rovnice, ktorej hodnoty poznáme, si označme ako $Z^n_{(79,79,79,1)}$, kde n je index rámca. Za predpokladu A dostávame rovnosť

$$\sigma_1(79) \oplus \sigma_2(79) \oplus \sigma_3(79) = Z^n_{(79,79,79,1)}. \tag{4.10}$$

Môžu nastať dva prípady, kedy je rovnica (4.10) splnená:

- Predpoklad A platí, t.j. počas 101 taktov sa každý register posunul presne 79-krát. Pravdepodobnosť, že predpoklad A platí si označme $P(A)$. Výraz (4.10) potom platí s pravdepodobnosťou 1.
- Predpoklad A neplatí. Potom ale výraz (4.10) platí len s pravdepodobnosťou 1/2.

Potom dostávame základný vzťah korelačného útoku

$$P(\sigma_1(79) \oplus \sigma_2(79) \oplus \sigma_3(79) = Z^n_{(79,79,79,1)}) = P(A \text{ platí}) \cdot 1 + P(A \text{ neplatí}) \cdot 1/2. \tag{4.11}$$

Do vzorca (4.11) dosadíme konkrétne hodnoty. Pravdepodobnosť $P(A)$ je približne 10^{-4}. Potom platí

$$P(\sigma_1(79)\oplus\sigma_2(79)\oplus\sigma_3(79) = Z^n_{(79,79,79,1)}) = 10^{-4}+(1-10^{-4})\cdot 1/2 = 1/2+1/2\cdot 10^{-4}. \tag{4.12}$$

Pripomeňme, že výraz $\sigma_1(79)\oplus\sigma_2(79)\oplus\sigma_3(79)$ je fixný pre každý rámec. Potom za predpokladu, že máme k dispozícii hodnoty $Z^n_{(79,79,79,1)}$ pre n rádovo miliónov rámcov, môžeme hodnotu výrazu $\sigma_1(79)\oplus\sigma_2(79)\oplus\sigma_3(79)$ správne určiť s vysokou pravdepodobnosťou. Túto hodnotu by sme určili jednoducho tak, že pre každý rámec i by sme si spočítali a zapamatali hodnotu $Z^i_{(79,79,79,1)}$. Potom výsledná hodnota výrazu $\sigma_1(79) \oplus \sigma_2(79) \oplus \sigma_3(79)$ by bol bit, ktorý by sa v postupnosti $Z^i_{(79,79,79,1)}$ nachádzal najčastejšie.

Takýmto spôsobom získame jeden bit informácie o *tajnom kľúči* K. Uvažovaním ďalších trojíc posunutí registrov počas prvých 101 taktov od *inicializačného vnútorného stavu* môžeme získať viac informácie o *tajnom kľúči*. Nevýhodou je však vysoký počet rámcov.

4.2 Popis útoku

V popise útoku budeme vychádzať z idei z predošlej podkapitoly, ktorú zovšeobecníme. Základnym vzťahom idei bol vzorec (4.11), ktorý v sebe obsahoval výraz $P(A)$, t.j. pravdepodobnosť, že sa počas 101 taktov každý register posunul práve 79-krát. Inými slovami, ak sa každý register od *inicializačného vnútorného stavu* posunul práve 79-krát, tak sa aktuálne vyprodukoval bit h_1 *hesla*. Avšak mohli sa s určitými pravdepodobnosťami vyprodukovať aj bity *hesla* $h_2, h_3, \ldots$.

Hlavným vylepšením bude zakomponovanie týchto pravdepodobnosti do vzorca (4.11), čím sa zlepší jeho kvalita.

Pred samotným popisom útoku si definujme si pravdepodobnosť

$$P((t_1, t_2, t_3) \text{ po } i \text{ taktoch}) \tag{4.13}$$

ako pravdepodobnosť, že od *inicializačného vnútorného stavu* sa registre R_1, R_2 a R_3 po i taktoch posunuli po rade t_1, t_2 a t_3-krát.

Veta 1. *Predpokladajme, že postupnosti taktovacích bitov sú pre každý register náhodné a navzájom nezávislé. Potom platí*

$$P((t_1, t_2, t_3) \textit{ po } i \textit{ taktoch}) = \frac{\binom{i}{i-t_1}\binom{t_1}{i-t_2}\binom{t_1+t_2-i}{i-t_3}}{4^i}. \tag{4.14}$$

Dôkaz. Počas jedného taktu môžu nastať nasledujúce štyri situácie:

- situácia (123) - posunú sa všetky tri registre,
- situácia (12) - posunú sa len registre R_1 a R_2,
- situácia (13) - posunú sa len registre R_1 a R_3,
- situácia (23) - posunú sa len registre R_2 a R_3.

Pretože sa vykoná i taktov, tak počet všetkých rôznych postupností situácii po i taktoch je 4^i. Aby sa register R_1 posunul t_1-krát, tak počas i taktov musela situácia (23) nastať $(i - t_1)$-krát. Takže počet možností, keď sa počas i taktov register R_1 posunul t_1-krát je $\binom{i}{i-t_1}$.

Predpokladajme, že sa register R_1 posunul t_1-krát, t.j. že situácia (23) nastala $(i - t_1)$-krát počas i taktov. V týchto $(i - t_1)$ prípadoch sa posunul register R_2. Vo zvyšných t_1 taktoch, keď nenastala situácia (23), potrebujeme, aby sa register R_2 posunul $(t_2 - (i - t_1))$-krát. Takže počet možností, keď sa register R_2 posunie t_2-krát je potom $\binom{t_1}{i-t_2}$, pretože platí $\binom{t_1}{t_2-(i-t_1)} = \binom{t_1}{i-t_2}$.

Nakoniec predpokladajme, že sa register R_1 posunul t_1-krát a register R_2 t_2-krát. Situácia (23) musela počas i taktov nastať $(i - t_1)$-krát a toľkokrát sa počas tejto situácie posunul register R_3. V prípadoch, keď nenastala situácia (23), tak sa $(t_2 - (i - t_1))$-krát posunul R_2, t.j. $t_1 - (t_2 - (i - t_1)) = (i - t_2)$-krát nastala situácia (13). Takže potrebujeme, aby sa v $t_2 - (i - t_1)$ taktoch register R_3 posunul $(t_3 - (i - t_1) - (i - t_2))$-krát, čím dostávame $\binom{t_2-(i-t_1)}{t_3-(i-t_1)-(i-t_2)} = \binom{t_1+t_2-i}{i-t_3}$ možností. □

Pozrime sa do akej miery pravdepodobnosti podľa vzorca (4.14) odpovedajú hodnotám, ktoré sme získali simuláciou. Uvažujme *inicializačný vnútorný stav*, t.j. máme nastavenie $(t_1, t_2, t_3) = (0, 0, 0)$ a $i = 0$. Necháme vykonať 101 taktov a budeme si zapisovať hodnoty (t_1, t_2, t_3) pre $i = 101$. Celkovo takúto simuláciu vykonáme sto miliónkrát a vytvoríme si frekvenčnú tabuľku najčastejších trojíc (t_1, t_2, t_3). Takúto simuláciu ďalej vykonáme pre $i = 103, 105$ a 107 taktov.

V nasledujúcich štyroch tabuľkách sú vždy v prvom stĺpci uvedené parametre pravdepodobnosti (4.14). Len v tabuľke si ju označme skrátene $P(t_1, t_2, t_3, i)$. V druhom stĺpci je hodnota pravdepodobnosti z prvého stĺpca vynásobená sto miliónmi. V treťom stĺpci je počet výskytov uvažovanej trojice (t_1, t_2, t_3) po i taktoch v sto miliónkrát opakovanej simulácii.

$P(t_1,t_2,t_3,i)$	vz. (4.14)	simul.	$P(t_1,t_2,t_3,i)$	vz. (4.14)	simul.
P(76, 76, 75, 101)	97434	97251	P(77, 78, 77, 103)	94641	94546
P(76, 76, 76, 101)	97434	97146	P(78, 77, 77, 103)	94641	94280
P(75, 76, 76, 101)	97434	97051	P(77, 77, 77, 103)	94641	94204
P(76, 75, 76, 101)	97434	96683	P(77, 77, 78, 103)	94641	93962
P(75, 77, 76, 101)	93686	94345	P(77, 78, 78, 103)	91136	91714
P(76, 77, 75, 101)	93686	94036	P(76, 78, 78, 103)	91136	91714
P(75, 76, 77, 101)	93686	93928	P(78, 77, 78, 103)	91136	91329
P(77, 75, 76, 101)	93686	93783	P(78, 78, 77, 103)	91136	91020
P(77, 76, 75, 101)	93686	93638	P(78, 78, 76, 103)	91136	90884
P(75, 77, 75, 101)	93686	93425	P(77, 78, 76, 103)	91136	90751

$P(t_1,t_2,t_3,i)$	vz. (4.14)	simul.	$P(t_1,t_2,t_3,i)$	vz. (4.14)	simul.
P(79, 79, 79, 105)	92012	92091	P(80 81 80, 107)	89472	89745
P(78, 79, 79, 105)	92012	91670	P(81 80 80, 107)	89472	89314
P(79, 79, 78, 105)	92012	91554	P(80 80 81, 107)	89472	89081
P(79, 78, 79, 105)	92012	91108	P(80 80 80, 107)	89472	88587
P(78, 80, 79, 105)	88605	89292	P(81 81 80, 107)	86277	87352
P(79, 80, 78, 105)	88605	88837	P(81 80 81, 107)	86277	86977
P(78, 80, 78, 105)	88605	88680	P(80 81 81, 107)	86277	86901
P(78, 79, 80, 105)	88605	88662	P(79 81 81, 107)	86277	86293
P(80, 78, 79, 105)	88605	88507	P(81 81 79, 107)	86277	86273
P(80, 78, 78, 105)	88605	88004	P(80 81 79, 107)	86277	86149

Poznamenajme, že autori tohto útoku uvádzajú v [9] iný vzorec, o ktorom tvrdia, že dáva rovnaké hodnoty ako (4.14). Samozrejme len pre *platné* trojice (t_1,t_2,cl_3) v i-tej pozícii, napr. trojica $(10,10,10)$ pre $i=100$-ú pozíciu nie je „platná“, pretože takáto situácia nemôže nastať. Uvádzajú nasledujúcu rekurzívnu funkciu:

$$P((t_1,t_2,t_3) \text{ po } i \text{ taktoch}) = \mathcal{F}(t_1,t_2,t_3,i), \tag{4.15}$$

kde

$$\mathcal{F}(t_1,t_2,t_3,0) = 1, \text{ ak } t_1=0, t_2=0, t_3=0,$$

$$\mathcal{F}(t_1,t_2,t_3,i) = \begin{cases} 0 & \text{ak } t_1<0 \text{ alebo } t_2<0 \text{ alebo } t_3<0, \\ 0 & \text{ak } t_1>i \text{ alebo } t_2>i \text{ alebo } t_3>i, \end{cases}$$

$$\mathcal{F}(t_1,t_2,t_3,i) = 0.25\mathcal{F}(t_1-1,t_2-1,t_3-1,i-1)+0.25\mathcal{F}(t_1,t_2-1,t_3-1,i-1)+$$

$$0.25\mathcal{F}(t_1-1,t_2,t_3-1,i-1)+0.25\mathcal{F}(t_1-1,t_2-1,t_3,i-1).$$

Po vysvetlení pravdepodobnosti (4.13) prejdime k popisu útoku. Uvažujme j-tý rámec a nejakú trojicu posunutí registrov (t_1,t_2,t_3). Definujme si interval $\mathcal{I}$ pre túto trojicu, ktorý bude obsahovať len tie hodnoty i, pre ktoré je pravdepodobnosť $P((t_1,t_2,t_3)$ po i taktoch) nezanedbateľná. Ďalej si označme

$p^j_{(t_1,t_2,t_3)} = P(\sigma_1(t_1) \oplus \sigma_2(t_2) \oplus \sigma_3(t_3) = 0)$. Vzorec (4.11), v ktorom sme uvažovali len jeden výskyt trojice (t_1, t_2, t_3), zovšeobecníme tak, že budeme uvažovať interval $\mathcal{I}$. Prepis vzorca (4.11) bude mať potom nasledujúci tvar:

$$p^j_{(t_1,t_2,t_3)} = \sum_{i \in \mathcal{I}} P((t_1, t_2, t_3) \text{ po } i \text{ taktoch}) \cdot \mathcal{X}(Z^j_{(t_1,t_2,t_3,i)} = 0) + 1/2 \cdot (1 - \sum_{i \in \mathcal{I}} P((t_1, t_2, t_3) \text{ po } i \text{ taktoch})), \tag{4.16}$$

kde $\mathcal{X}$ je obdobou charakteristickej funkcie, t.j.

$$\mathcal{X}(Z = 0) = \begin{cases} 1 & \text{ak } Z = 0, \\ 0 & \text{ak } Z \neq 0. \end{cases}$$

Hodnota $Z^j_{(t_1,t_2,t_3,i)}$ pre j- tý rámec je definovaná ako

$$Z^j_{(t_1,t_2,t_3,i)} = \phi_1(t_1) \oplus \phi_2(t_2) \oplus \phi_3(t_3) \oplus h_{i-100},$$

kde h_k je k-tý bit *hesla*, ktorý predpokladáme, že poznáme. Hodnoty $\phi_1(t_1), \phi_2(t_2)$ a $\phi_3(t_3)$ sú odvodené od *čísla rámca* a tiež predpokladáme, že ich poznáme. V intervale $\mathcal{I}$ sa nachádzajú všetky $i \in \mathcal{I}$, pre ktoré je pre danú trojicu (t_1, t_2, t_3) pravdepodobnosť $P((t_1, t_2, t_3)$ po i taktoch) nezanedbateľná.

Počet prístupných rámcov si označme ako m. Zadefinujme si $\mathcal{P}_{(t_1,t_2,t_3)} = P(\sigma_1(t_1) \oplus \sigma_2(t_2) \oplus \sigma_3(t_3) = 0)$ ako pravdepodobnosť, že $\sigma_1(t_1) \oplus \sigma_2(t_2) \oplus \sigma_3(t_3) = 0$ uvažovanú cez všetkých m rámcov. Uvedomme si rozdiel medzi pravdepodobnosťami $p^j_{(t_1,t_2,t_3)}$ a $\mathcal{P}_{(t_1,t_2,t_3)}$. Prvý výraz nám hovorí, s akou pravdepodobnosťou platí $\sigma_1(t_1) \oplus \sigma_2(t_2) \oplus \sigma_3(t_3) = 0$ pre daný j-tý rámec. Na rozdiel od toho výraz $\mathcal{P}_{(t_1,t_2,t_3)}$ využíva informáciu zo všetkých m prístupných rámcov.

Ďalej si definujme tzv. *log-pravdepodobnostný podiel*[2] $\Lambda_{(t_1,t_2,t_3)}$ pravdepodobnosti $\mathcal{P}_{(t_1,t_2,t_3)}$ ako

$$\Lambda_{(t_1,t_2,t_3)} = \ln \frac{\mathcal{P}_{(t_1,t_2,t_3)}}{1 - \mathcal{P}_{(t_1,t_2,t_3)}}, \tag{4.17}$$

kde *ln* označuje prirodzený logaritmus. Viac o tomto pojme sa môžeme dozvedieť v [10], nám stačí vedieť, že platia nasledujúce vzťahy:

$$\Lambda_{(t_1,t_2,t_3)} = 0 \text{ ak } P(\sigma_1(t_1) \oplus \sigma_2(t_2) \oplus \sigma_3(t_3) = 0) = 1/2,$$
$$\Lambda_{(t_1,t_2,t_3)} > 0 \text{ ak } P(\sigma_1(t_1) \oplus \sigma_2(t_2) \oplus \sigma_3(t_3) = 0) > 1/2,$$
$$\Lambda_{(t_1,t_2,t_3)} < 0 \text{ ak } P(\sigma_1(t_1) \oplus \sigma_2(t_2) \oplus \sigma_3(t_3) = 0) < 1/2.$$

Hodnotu *log-pravdepodobnostného podielu* odhadneme podľa nasledujúceho vzťahu:

$$\Lambda_{(t_1,t_2,t_3)} = \sum_{j=1}^{m} \ln \frac{p^j_{(t_1,t_2,t_3)}}{1 - p^j_{(t_1,t_2,t_3)}}. \tag{4.18}$$

Zrekapitulujme si doterajší útok. Uvažujme nejakú trojicu (t_1, t_2, t_3) počtov posunutí registrov R_1, R_2 a R_3. Pre všetkých m známych rámcov si spočítame

[2]pozn. anglický názov je log-likelihood ratio

pravdepodobnosti $p^j_{(t_1,t_2,t_3)}$, $j = 1, \ldots, m$ pomocou vzorca (4.16). Z nich pomocou vzorca (4.18) spočítame hodnotu $\Lambda_{(t_1,t_2,t_3)}$ a na základe nej vykonáme odhad výrazu $\sigma_1(t_1) \oplus \sigma_2(t_2) \oplus \sigma_3(t_3)$. Ak napr. $\Lambda_{(t_1,t_2,t_3)} = 1$, tak odhadneme, že $\sigma_1(t_1) \oplus \sigma_2(t_2) \oplus \sigma_3(t_3) = 0$. Ak napr. $\Lambda_{(t_1,t_2,t_3)} = -2$, tak odhadneme, že $\sigma_1(t_1) \oplus \sigma_2(t_2) \oplus \sigma_3(t_3) = 1$. Tento postup budeme aplikovať na viacero trojíc (t_1, t_2, t_3).

Definujme si napr. tri intervaly dĺžky 8: $\mathcal{J}_1 = [79, 86]$, $\mathcal{J}_2 = [87, 94]$ a $\mathcal{J}_3 = [95, 102]$. Na začiatok nech $t_1, t_2, t_3 \in \mathcal{J}_1$. Pretože interval $\mathcal{J}_1$ má 8 prvkov, dostaneme 512 rôznych trojíc (t_1, t_2, t_3). Pre každú takúto trojicu pomocou hore uvedeného postupu odhadneme výraz $\sigma_1(t_1) \oplus \sigma_2(t_2) \oplus \sigma_3(t_3)$. Dostaneme 512 rovníc s 24 neznámymi

$$\sigma_1(79), \ldots, \sigma_1(86), \sigma_2(79), \ldots, \sigma_2(86), \sigma_3(79), \ldots, \sigma_3(86).$$

Sústava rovníc bude v tvare

$$\begin{aligned}
\sigma_1(79) \oplus \sigma_2(79) \oplus \sigma_3(79) &= \mathcal{H}(\Lambda_{(79,79,79)}),\\
\sigma_1(79) \oplus \sigma_2(79) \oplus \sigma_3(80) &= \mathcal{H}(\Lambda_{(79,79,80)}),\\
&\vdots\\
\sigma_1(85) \oplus \sigma_2(86) \oplus \sigma_3(86) &= \mathcal{H}(\Lambda_{(86,86,86)}),\\
\sigma_1(86) \oplus \sigma_2(86) \oplus \sigma_3(86) &= \mathcal{H}(\Lambda_{(86,86,86)}),
\end{aligned} \tag{4.19}$$

kde funkcia $\mathcal{H}$ je definovaná takto

$$\mathcal{H}(x) = \begin{cases} 0 & \text{ak } x \geq 0,\\ 1 & \text{ak } x < 0. \end{cases}$$

Hodnoty uvedených 24 neznámych získame hrubou silou nasledujúcim postupom. Každú z 2^{24} rôznych možností dosadíme za neznáme v sústave (4.19), čím na ľavej strane sústavy dostaneme vektor $v \in \mathbb{Z}_2^{24}$. Pre všetkých 2^{24} takto získaných vektorov v spočítame *Hammingovú vzdialenosť* [3] medzi v a vektorom reprezentujúcim pravú stranu sústavy (4.19). Za konečné riešenie sústavy budeme brať taký vektor v, ktorý bude „najbližie“ k vektoru pravej strany v zmysle *Hammingovej vzdialenosti*.

Rovnakým spôsobom budeme postupovať pre intervaly $\mathcal{J}_2$ a $\mathcal{J}_3$. Tak získame celkovo 24+24+24 = 72 bitov informácie o *tajnom kľúči*, pričom na jeho odhalenie nám stačí poznať len 64 bitov. Získame nasledujúce bity:

$$\begin{aligned}
R_1 &: \sigma_1(79), \ldots, \sigma_1(102),\\
R_2 &: \sigma_2(79), \ldots, \sigma_2(102),\\
R_3 &: \sigma_3(79), \ldots, \sigma_3(102).
\end{aligned} \tag{4.20}$$

Záverečný krok útoku je overovacia fáza. Uvedené bity (4.20) si uložíme do príslušných registrov. Potom každý register posunieme $79 + 22 = 101$-krát v opačnom smere ako pri bežnom taktovaní. Po 79 posunutiach sa dostaneme do času $t = 0$ príslušný pre *inicializačný vnútorný stav* a po ďalších 22 posunutiach sa

[3] pre $u, v \in \mathbb{Z}_2^{24}$ ju definujeme ako $d(u, v) = |\{i; u_i \neq v_i\}|$, t.j. počet súradníc, v ktorých sa vektory u a v líšia

dostaneme na začiatok 3. kroku *inicializačného algoritmu*, v ktorom sa vykonáva načítane *čísla rámca*. Ďalej budeme pokračovať v *inicializačnom algoritme* bežným spôsobom, t.j. načítame 22 bitové *číslo rámca* a 100-krát vykonáme takt šifry, pričom výstupné bity vždy vyhodíme. Pretože *tajný kľúč* má 64 bitov, vykonáme 64 taktov a príslušné výstupné bity porovnáme s príslušnými bitmi *hesla*. Nakoniec poznamenajme, že o pár rokov spoluautor tohto článku T. Johansson s ďalšími kolegami rozšírili uvedený útok o nové pozorovania, čím zlepšili jeho výsledky. Ich výsledky nájdeme v [11].

4.3 Implementácia útoku

V tejto podkapitole predstavíme detaily implementácie útoku. Uvedieme si parametre implementácie a potom samotný pseudokód algoritmu útoku. Vstupom do algoritmu budú *heslá* vyprodukované z $m = 70000$ rámcov, označme si ich ako $heslo_1, \ldots, heslo_m$. Pretože všetky *čísla rámcov* sú verejne známe, predpokladáme ich znalosť. *Čísla rámcov* príslušné k *heslám* $heslo_i$, $i = 1, \ldots, m$, budú ďalším vstupom algoritmu a označme si ich ako F_i, $i = 1, \ldots, m$.

Pravdepodobnosti $P((t_1, t_2, t_3)$ po i taktoch$)$ nezávisia na vstupoch algoritmu, preto si ich môžeme spočítať dopredu. V našej implementácii sme uvažovali tieto štyri intervaly

$$\mathcal{J}_1 = \{79, \ldots, 86\}, \quad \mathcal{J}_2 = \{84, \ldots, 91\}, \quad \mathcal{J}_3 = \{89, \ldots, 96\}, \quad \mathcal{J}_4 = \{94, \ldots, 101\}.$$

Pre každý interval $\mathcal{J}_i$, $i = 1, \ldots, 4$, resp. pre každú jeho trojicu prvkov $t_1, t_2, t_3 \in \mathcal{J}_i$ je potrebné si definovať interval $\mathcal{I}$, pre ktorý je pravdepodobnosť $P((t_1, t_2, t_3)$ po i taktoch$)$, kde $i \in \mathcal{I}$, nezanedbateľná. Pre každú trojicu sme interval $\mathcal{I}$ získali tak, že sme spočítali všetky hodnoty pravdepodobnosti $P((t_1, t_2, t_3)$ po i taktoch$)$ pre $i = 101, \ldots, 150$. Interval $\mathcal{I}$ sme definovali tak, aby obsahoval všetky i také, že $P((t_1, t_2, t_3)$ po i taktoch$) \geq 10^{-5}$. Najväčšie takto získané i je 140, čo znamená, že z každého *hesla* $heslo_i$, $i = 1, \ldots, m$, budeme potrebovať len prvých 40 bitov.

V nasledujúcich tabuľkách uvádzame pre štyri konkrétne trojice (t_1, t_2, t_3), ako rýchlo sa mení pravdepodobnosť v závislosti na i. Len v tabuľke si ju označme skrátene $P(t_1, t_2, t_3, i)$. Maximálna hodnota je zvýraznená tučným písmom.

i	$P(76, 76, 75, i) \cdot 10^6$	$P(77, 78, 77, i) \cdot 10^6$
101	**974.34**	555.52
102	816.75	821.62
103	521.35	**946.41**
104	250.40	843.85
105	89.100	577.64
106	23.019	300.32
107	4.2050	116.97
108	0.5240	33.519
109	0.0420	6.9060
110	0.0020	0.9920

i	$P(100,96,102,i)\cdot 10^6$	$P(99,95,103,i)\cdot 10^6$
125	2.3640	3.3460
126	8.9750	11.838
127	28.189	34.607
128	73.261	83.595
129	157.456	166.718
130	279.517	274.107
131	409.051	370.702
132	**492.139**	**411.124**
133	485.060	372.43
134	389.887	274.206
135	254.147	163.078
136	133.427	77.752
137	55.945	29.445
138	18.543	8.7570
139	4.7980	2.0170

Teraz si uvedieme posledný parameter implementácie - počet najlepších riešení sústavy (4.19). Najlepším riešením sa myslí riešenie, ktoré je najbližšie k pravej strane sústavy (4.19) v zmysle *Hammingovej vzdialenosti*. Takéto riešenie je však málokedy správnym riešením sústavy. Preto si pre každý interval $\mathcal{J}_i, i = 1,\ldots,4$ spočítame a uložíme 1000 najlepších riešení.

Konečne si predstavme pseudokód algoritmu, podľa ktorého sa vznikla implementácia útoku.

KORELAČNÝ ÚTOK
VSTUP: *heslá:* $heslo_i$, $i = 1,\ldots,m$, a *čísla rámcov:* F_i, $i = 1,\ldots,m$
VÝSTUP: *tajný kľúč*

1. `FOR EACH` interval $\mathcal{J}_1,\ldots,\mathcal{J}_4$ `DO`
   1.1 `FOR EACH` (t_1,t_2,t_3) z vybraného intervalu `DO`
      1.1.1 Spočítaj $p^j_{(t_1,t_2,t_3)}$ podľa (4.16),
      1.1.2 Spočítaj $\Lambda_{(t_1,t_2,t_3)}$ podľa (4.18),
      1.1.3 Spočítaj $\mathcal{H}(\Lambda_{(t_1,t_2,t_3)})$,
   1.2. Získame a uložíme 1000 najlepších riešení sústavy (4.19),
2. *Kombinácia riešení zo všetkých intervalov a overenie správnosti*,
3. `IF` *riešenie správne*,
   3.1 *Odvodenie tajného kľúča*,
   3.2 `RETURN` *tajný kľúč*,

`ELSE` *Vypíš: útok zlyhal.*

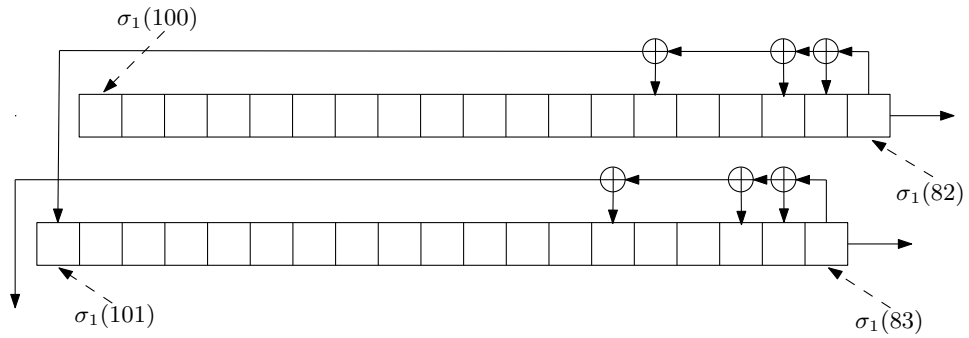
Detailnejšie sa pozrime na 2.krok *Kombinácia riešení zo všetkých intervalov a overenie správnosti*. Všimnime si, že intervaly $\mathcal{J}_i$, $i = 1,\ldots,4$, sú prekrývajúce: $|\mathcal{J}_1 \cap \mathcal{J}_2| = |\mathcal{J}_2 \cap \mathcal{J}_3| = |\mathcal{J}_3 \cap \mathcal{J}_4| = 3$. Je to z toho dôvodu, že keby sa neprekrývali, tak by sme museli skontrolovať až 1000^4 všetkých možných prípadov. V prípade prekrývajúcich intervalov môžeme využiť fakt, že ak v každom intervale máme správne riešenie sústavy, tak nasledujúcich deväť prekrývajúcich bitov sa musí zhodovať:

$$\mathcal{J}_1 \cap \mathcal{J}_2 : \sigma_1(84), \sigma_1(85), \sigma_1(86), \sigma_2(84), \sigma_2(85), \sigma_2(86), \sigma_3(84), \sigma_3(85), \sigma_3(86)$$

$$\mathcal{J}_2 \cap \mathcal{J}_3 : \sigma_1(89), \sigma_1(90), \sigma_1(91), \sigma_2(89), \sigma_2(90), \sigma_2(91), \sigma_3(89), \sigma_3(90), \sigma_3(91)$$

$$\mathcal{J}_3 \cap \mathcal{J}_4 : \sigma_1(94), \sigma_1(95), \sigma_1(96), \sigma_2(94), \sigma_2(95), \sigma_2(96), \sigma_3(94), \sigma_3(95), \sigma_3(96),$$

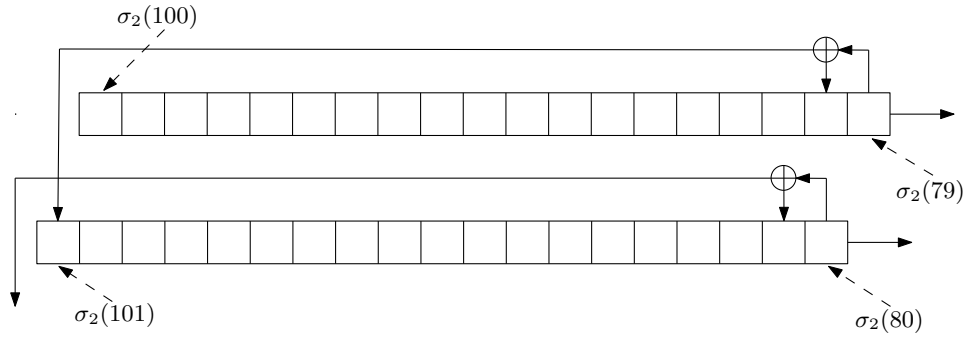
Ďalší fakt, ktorý môžeme využiť v overovacej fáze je ten, že postupnosť $\sigma_1(79), \ldots, \sigma_1(101)$, ktorá má 23 prvkov, je o 4 bity dlhšia, ako dĺžka registra R_1. Tieto 4 bity musia spĺňať rovnicu spätnej väzby, ktorú využijeme pri overovaní. Situáciu si znázornime na obrázku 4.1.



Obr. 4.1: Ilustrácia spätnej väzby pre register R_1.

Z obrázku jasne vidno rovnicu spätnej väzby $\sigma_1(101) = \sigma_1(82) \oplus \sigma_1(83) \oplus \sigma_1(84) \oplus \sigma_1(87)$. Prepíšeme si ju do tvaru $\sigma_1(82) = \sigma_1(83) \oplus \sigma_1(84) \oplus \sigma_1(87) \oplus \sigma_1(101)$, čo bude prvá kontrolná rovnica. Pretože ešte máme k dispozícii bity $\sigma_1(81), \ldots, \sigma_1(79)$, analogicky vytvoríme príslušné rovnice pomocou spätnej väzby, ktoré použijeme v overovacej fáze.

Ďalej aj postupnosť $\sigma_2(79), \ldots, \sigma_2(101)$ je o 1 bit dlhšia, ako dĺžka registra R_2. Situácia je znázornená na obrázku 4.2.



Obr. 4.2: Ilustrácia spätnej väzby pre register R_2.

Z obrázku opäť vidno, že rovnica spätnej väzby je $\sigma_2(101) = \sigma_1(79) \oplus \sigma_1(80)$. Po úprave má tvar $\sigma_2(79) = \sigma_1(80) \oplus \sigma_1(101)$. Pretože register R_3 je rovnako dlhý, ako postupnosť $\sigma_3(79), \ldots, \sigma_3(101)$, nezískame z neho žiadnu kontrolnú rovnicu.

Zrekapitulujme si všetky rovnice, ktorých platnosť budeme kontrolovať pri

overovacej fáze:

$$\begin{aligned}
\sigma_1(82) &= \sigma_1(83) \oplus \sigma_1(84) \oplus \sigma_1(87) \oplus \sigma_1(101), \\
\sigma_1(81) &= \sigma_1(82) \oplus \sigma_1(83) \oplus \sigma_1(86) \oplus \sigma_1(100), \\
\sigma_1(80) &= \sigma_1(81) \oplus \sigma_1(82) \oplus \sigma_1(85) \oplus \sigma_1(99), \\
\sigma_1(79) &= \sigma_1(80) \oplus \sigma_1(81) \oplus \sigma_1(84) \oplus \sigma_1(98), \\
\sigma_2(79) &= \sigma_2(80) \oplus \sigma_2(101).
\end{aligned} \tag{4.21}$$

Kontrolu zhody prekrývajúcich sa bitov a využitie vzorcov (4.20) vykonávame hneď na začiatku overovacej fázy, pretože sú výpočetne menej náročné ako 101 násobné taktovanie v opačnom smere.

Dôvod zlyhania útoku je taký, že aspoň v jednom zozname 1000 najlepších riešení sústavy (4.19) sa nenachádza správne riešenie. V nasledujúcej tabuľke ilustrujeme niekoľko takých prípadov. V jednotlivých riadkoch sú uvedené poradia riešení sústavy (4.19) z tabuľky najlepších kandidátov pre jednotlivé intervaly $\mathcal{J}_i, i = 1, \ldots, 4$. Hviezdičkou je označené poradie, ktoré je vyššie ako 1000.

$\mathcal{J}_1$	$\mathcal{J}_2$	$\mathcal{J}_3$	$\mathcal{J}_4$
1	2	6	*
2	*	651	4
24	6	*	*
5	1	*	434
*	*	2	50

Posledný krok algoritmu je *Odvodenie tajného kľúča*. Naším cieľom je zo znalosti hodnôt $\sigma_1(t), \sigma_2(t)$ a$\sigma_3(t)$, $t = 79, \ldots, 101$, získať bity *tajného kľúča* $K = (k_1, \ldots, k_{68})$. Prvým krokom bude získať hodnoty vnútorného stavu v čase bezprostredne po 2.kroku *inicializačného algoritmu*, t.j. po načítaní *tajného kľúča*.

Z obrázkov 4.1. a 4.2. a z rovníc (4.21) jednoducho zovšeobecníme vzorce pre výpočet prvkov $\sigma_1(t), \sigma_2(t)$, $t < 79$. V prípade registra R_3 je postup analogický. Vykonáme to pomocou nasledujúcich vzťahov:

$$\begin{aligned}
\sigma_1(t) &= \sigma_1(t+1) \oplus \sigma_1(t+2) \oplus \sigma_1(t+5) \oplus \sigma_1(t+19), t < 79, \\
\sigma_2(t) &= \sigma_t(t+1) \oplus \sigma_2(t+22), t < 79, \\
\sigma_3(t) &= \sigma_3(t+1) \oplus \sigma_3(t+2) \oplus \sigma_3(t+15) \oplus \sigma_3(t+23), t < 79.
\end{aligned} \tag{4.22}$$

Každý zo vzorcov (4.22) aplikujeme pre $t = 78, \ldots, -22$, t.j.vrátime sa o 79+22=101 taktov naspäť až do času ihneď po načítaní *tajného kľúča*. Bity vnútorného stavu v tomto čase potom budú nasledujúce:

- $R_1 : \sigma_1(0), \sigma_1(-1), \ldots, \sigma_1(-22)$,
- $R_2 : \sigma_2(-1), \sigma_2(-2), \ldots, \sigma_2(-22)$,
- $R_3 : \sigma_3(-4), \sigma_3(-5), \ldots, \sigma_3(-22)$.

Každý bit uvedeného vnútorného stavu je nejakou lineárnou kombináciou bitov $k_1, \ldots, k_{68}$. Vytvoríme si sústavu 64 rovníc so 64 neznámymi. Neznáme sústavy budú práve bity $k_1, \ldots, k_{68}$. Pravá strana matice bude v tvare

$$(\sigma_1(0), \ldots, \sigma_1(-22), \sigma_2(-1), \ldots, \sigma_2(-22), \sigma_3(-4), \ldots, \sigma_3(-22))^T.$$

Homogénnu maticu sústavy uvádzame v prílohe C. Jej konštrukciu vynechávame kvôli obšírnym technickým a nie príliš zaujímavým algoritmom. V prípade záujmu, čitateľa odkazujeme na adresár *Correl* na priloženom CD, kde nájde triedu *Attack.java* a v jej metóde *vypisTajnyKluc* môže nájsť kompletný postup.

Na záver implementácie uveďme, že kompletný útok sme pustili 100-krát a priemerný výpočtový čas vyšiel približne šesť a pol minúty. Úspešnosť útoku, t.j. získanie správneho *tajného kľúča* vyšla mierne nad 50%. V útoku sa používali predpočítané tabuľky s pravdepodobnosťami $P((t_1, t_2, t_3)$ po i taktoch), ktoré zaberajú menej ako 0.5 MB.

5. Poznámky k implementácii

V tejto kapitole si predstavíme jednotlivé programy, ktoré boli použité v práci. Vysvetlíme si, aké majú vstupné parametre a ako sa spúšťajú. Táto kapitola môže byť chápaná ako užívateľská dokumentácia. Programy sú naprogramované v jazyku Java a aby ich bolo možné spustiť, je potrebné mať nainštalovanú Javu JRE 6. Zdrojové kódy sú uložené na priloženom CD.

Implementácia šifry A5/1:

Implementácia šifry *A5/1* sa nachádza v adresári s názvom *a51*. V tomto adresári sa nachádza trieda *A51*, ktorá obsahuje procedúry používané pri výpočte šifry. Pred každou procedúrou je komentár, v ktorom je popísaný účel, vstupy a výstupy procedúry. Táto trieda si náhodne vygeneruje tajný kľúč a číslo rámca, potrebné pri inicializácii a vypíše ich na konzolu. Po inicializačnej fáze sa vypíše vnútorný stav šifry v čase $t = 101$. Potom sa vyprodukuje *heslo*, ktoré sa následne vypíše na konzolu. Trieda *A51* nemá žiadny vstup. Program je potrebné spustiť z adresára *a51* nasledujúcim príkazom:
java -cp "dist/a51.jar" a51.A51

Implementácia útoku od Bihama a Dunkelmana::

Implementácia jednotlivých tried sa nachádza v adresári s názvom *bihDun* a obsahuje nasledujúce časti:

1.trieda *bihdun.Bihdun* : ide o implementáciu útoku od Bihama a Dunkelmana. Vygeneruje sa 220 bitov *hesla* a pustí sa na ne algoritmus. Ak tieto bity *hesla* poskytnú špeciálny prípad: register R_3 sa 10-krát po sebe neposunie; tak sa spočíta hľadaný vnútorný stav a vypíše sa výsledok. Ak nenastane špeciálny prípad, program vypíše, že sa vygeneroval nedostatok bitov *hesla* a treba vygenerovať ďalšie bity. Výstup je na konzolu. Poznámka: ide o kompletný útok bez zjednodušujúcich predpokladov, preto dĺžka výpočtu na jednom PC je príliš vysoká. Program je potrebné pustiť z adresára *bihDun* nasledujúcim príkazom:
java -cp "dist/bihDun.jar" bihdun.Bihdun

2.trieda *bihdun.test* : je zjednodušená verzia útoku od Bihama a Dunkelmana. Predpokladá sa, že nastal spomínaný špeciálny prípad. Program prejde ohodnotenia *hádaných bitov* až pokým nenájde ich správne hodnoty. Potom sa dopočítajú zvyšné neznáme bity a na konzolu sa vypíše hľadaný vnútorný stav. Program je potrebné pustiť z adresára *bihDun* nasledujúcim príkazom:
java -cp "dist/bihDun.jar" bihdun.test

Implementácia útoku od Golića na vnútorný stav $R^{(101)}$:

Implementácia jednotlivých tried sa nachádza v adresári s názvom *finalGolic* a obsahuje nasledujúce časti:

1. trieda *finalgolic.Verzia1*: ide o implementáciu 1.verzie Golićovho útoku. Trieda číta vstup zo súboru files/inputVerzia1.txt a výstup píše na konzolu. Na vstupe sa očakávajú tri riadky, ktoré obsahujú znaky '0' a '1' oddelené medzerami. Jednotlivé riadky vstupu reprezentujú obsahy registrov. V prvom riadku sa preto očakáva 19 bitov, v druhom riadku sa očakáva 22 bitov a v treťom riadku sa očakáva 23 bitov. Príklad vstupu:

0 1 0 1 0 1 0 0 1 1 1 0 1 1 1 0 0 1 0
0 1 1 1 0 0 0 0 0 1 1 0 0 1 0 0 1 0 1 1 1 0
1 1 1 0 0 0 0 1 0 1 1 0 0 0 1 0 0 1 0 1 0 1 0

Program vykoná kryptoanalýzu šifry *A5/1* za zjednodušujúceho predpokladu, že poznáme *hádané bity*. Výstupom programu bude hľadaný vnútorný stav. Program je potrebné spustiť z adresára *finalGolic* nasledujúcim príkazom:
java -cp "dist/finalGolic.jar" finalgolic.Verzia1

2. trieda *finalgolic.Verzia1r*: určená na zistenie priemerného počtu nadbytočných lineárne zívislých rovníc pre 1. verziu útoku od Golića. Trieda neobsahuje žiadny vstup. Vygeneruje si 10 000 náhodných vnútorných stavov a pre každý stav spočíta počet nadbytočných lineárne závislých rovníc a počet vykonaných taktov v strome. Výstup bude na konzolu, kde sa vypíše stav s najmenším a s najväčším počtom nadbytočných lineárne závislých rovníc. Ďalej sa vypíše frekvenčná tabuľka počtu vnútorných stavov, ktoré vyprodukovali daný počet lineárne závislých rovníc. Nakoniec sa vypíše priemerná hĺbka stromu. Program je potrebné spustiť z adresára *finalGolic* nasledujúcim príkazom:
java -cp "dist/finalGolic.jar" finalgolic.Verzia1r

3. trieda: *finalgolic.Verzia2*: ide o implementáciu 2.verzie Golićovho útoku. Trieda číta vstup zo súboru files/inputVerzia2.txt a výstup píše na konzolu. Vstup je v rovnakom formáte ako v prípade triedy *finalgolic.Verzia1*. Program vykoná kryptoanalýzu šifry *A5/1* za zjednodušujúceho predpokladu, že poznáme *hádané bity*. Výstupom programu bude hľadaný vnútorný stav. Program je potrebné spustiť z adresára *finalGolic* nasledujúcim príkazom:
java -cp "dist/finalGolic.jar" finalgolic.Verzia2

4. trieda *finalgolic.Verzia2r*: určená na zistenie priemerného počtu nadbytočných lineárne závislých rovníc pre 2. verziu útoku. Trieda neobsahuje žiadny vstup. Funkciu a výstup má táto trieda analogický, ako trieda *finalgolic.Verzia1r*. Program je potrebné spustiť z adresára *finalGolic* nasledujúcim príkazom:
java -cp "dist/finalGolic.jar" finalgolic.Verzia2r

5. trieda *finalgolic.PorninStern*: ide o implementáciu 2.verzie Golićovho útoku s využitím Porninovho a Sternovho triku. Vstupný súbor je files/PorninStern.txt a výstup sa píše na konzolu. Vstup je v rovnakom formáte ako v prípade triedy *finalgolic.Verzia1*. Program vykoná kryptoanalýzu šifry *A5/1* za zjednodušujúceho predpokladu, že poznáme *hádané bity*. Výstup programu bude na konzolu, kde sa vypíše hľadaný vnútorný stav. Výstup ďalej obsahuje ďalšie informácie o výslednom vnútornom stave: počet vykonaných taktov a indexy výstupných bi-

tov. Program môžeme pustiť z adresára *finalGolic* nasledujúcim príkazom:
java -cp "dist/finalGolic.jar" finalgolic.PorninStern

6.trieda *finalgolic.JustTest*: ide o implementáciu 2.verzie Golićovho útoku s využitím Porninovho a Sternovho triku. Vygeneruje sa 10 000 náhodných vnútorných stavov a na každého z nich sa vykoná kryptoanalýza za zjednodušujúcej podmienky znalosti *hádaných bitov*. Táto trieda neobsahuje žiadny vstup. Výstup je na konzolu, kde sa vypíše maximálny, priemerný a minimálny čas kryptoanalýzy jedného vnútorného stavu. Program je potrebné spustiť z adresára *finalGolic* nasledujúcim príkazom:
java -cp "dist/finalGolic.jar" finalgolic.JustTest

7.trieda *finalgolic.MinSust*: táto trieda vypočíta pre každú lineárne závislú rovnicu jej príslušnú minimálnu sústavu rovníc, ktoré generujú uvažovanú rovnicu. Trieda číta vstup zo súboru files/minSust.txt a výstup píše na konzolu. Na výstupe pri každej lineárne závislej rovnici bude uvedený jej index v riadku matice a pod ňou bude príslušná minimálna sústava rovníc, ktorá ju generujú. Tento program slúži na štúdium vzniku lineárne závislých rovníc. Program je potrebné spustiť z adresára *finalGolic* nasledujúcim príkazom:
java -cp "dist/finalGolic.jar" finalgolic.MinSust

Implementácia útoku od Golića na počiatočný vnútorný stav $R^{(0)}$:

Implementácia útoku na *počiatočný vnútorný stav* sa nachádza v adresári s názvom *R0*. V tomto adresári sa nachádza trieda *pocStav*, ktorá číta vstup zo súboru files/pocStav.txt a výstup píše na konzolu. Vstup je v rovnakom formáte, ako v popise triedy *finalgolic.Verzia1*. Na vstupe sa nachádza *počiatočný vnútorný stav*, na ktorom budeme testovať útok. Vykoná sa 101 taktov šifry, čím dostaneme vnútorný stav $R^{(101)}$ a na ňom aplikujeme spomínaný útok. Na výstupe bude v prípade úspechu znovu *počiatočný vnútorný stav* alebo sa zobrazí informácia, že sme si na začiatku vygenerovali malý interval trojíc posunov registrov. Program je potrebné spustiť z adresára *R0* nasledujúcim príkazom:
java -cp "dist/R0.jar" r0.pocStav

Implementácia korelačného útoku:

Implementácia jednotlivých tried sa nachádza v adresári s názvom *Correl* a obsahuje nasledujúce časti:

1.trieda *correl.Attack*: obsahuje kompletnú implementáciu korelačného útoku od Ekdahla a Johanssona. Trieda načíta vstupné súbory: files/frameKey.dat, files/tab1_formated.dat, files/tab2_formated.dat, files/tab3_formated.dat a files/tab4_formated.dat, ktoré obsahujú informácie o bitoch *hesla* zo 70000 rámcov a predpočítané tabuľky pravdepodobností používané počas útoku. Postupne sa pre všetky intervaly $\mathcal{J}_i, i = 1, \ldots, 4$ získa tabuľka s 1000 najlepšími riešeniami, prebehne overovacia fáza a v prípade úspechu sa spočíta *tajný kľúč*. Program je potrebné pustiť z adresára *correl* nasledujúcim príkazom:
java -cp "dist/Correl.jar" correl.Attack

2.trieda *correl.Precomput*: implementuje postup na získanie predpočítaných tabuliek, t.j. súborov: tab1_formated.dat, tab2_formated.dat, tab3_formated.dat a tab4_formated.dat, ktoré sú vstupom do samotného korelačného útoku. Tieto súbory sa po vytvorení uložia do podadresára *files*. Program je potrebné pustiť z adresára *correl* nasledujúcim príkazom:
java -cp "dist/Correl.jar" correl.Precomput

Na záver uvádzame parametre počítača, na ktorom boli testované všetky uvedené programy:
Intel Core 2 Duo E6750 @ 2666 MHz
2048 MB (2 x 1024 DDR2-SDRAM)

Závěr

V práci sme sa venovali kryptoanalýze prúdovej šifry *A5/1*. Predstavili sme si a implementovali tri útoky, ktoré predpokladajú znalosť otvoreného textu a príslušného šifrového textu. Pretože šifrovanie prebieha pomocou operácie *XOR*, predpokladáme znalosť *hesla*. Každý z troch útokov predpokladá iné množstvo *hesla*, pričom tieto veľkosti sa výrazne líšia.

Prvý útok, ktorého autormi sú Biham a Dunkelman, predpokladá v priemere 2^{20} bitov *hesla*, kým v druhom útoku nám stačí len 64 bitov *hesla*. Nevýhodou druhého útoku je pomerne zložitá výpočtová jednotka, ktorá je rovná vyriešeniu sústavy lineárnych rovníc nad telesom $\mathbf{Z}_2$.

V práci sme sa venovali hlavne druhému útoku, ktorého autorom je Golić. Počas implementácie sme narazili na problém existencie lineárnych rovníc, ktoré vznikajú počas prechodu stromom. Tento problém sme sa snažili vyriešiť použitím inej štruktúry stromu. Na jednej strane sa znížil počet nadbytočných lineárne závislých rovníc, na druhej strane problém existencie závislých rovníc pretrval.

Použitím Porninovho a Sternovho triku sa tento problém určitým spôsobom obišiel, ale za cenu čiastočnej Gaussovej eliminácie na každej pridávanej rovnice do sústavy. Po aplikácii spomínaného triku sa podarilo znížiť výpočetný čas na približne 800 dní pri použití bežného počítaču. Pretože štruktúra programu nám dovoluje rozložiť úlohu na viac podúloh, ktoré by sa mohli riešiť paralélne, môže výpočet programu prebehnúť v reálnejšom čase.

Na záver sme sa venovali korelačnému útoku od Ekdahla a Johanssona. Ich útok sme úspešne implementovali, pričom výpočtový čas trval len o niečo viac, ako 6 minút. Úspešnosť útoku je o niečo vyššia ako 50% a predpokladá sa znalosť takého počtu bitov *hesla*, ktoré by zabezpečilo viac ako 5 minutovú komunikáciu.

Zoznam použitej literatúry

[1] A. Menezes, P. van Oorschot, S. Vanstone, Handbook of Applied Cryptography, CRC Press, 1997.

[2] Steve Babbage, Some thoughts on Trivium, 2007.

[3] M.Bricero, I.Goldberg, D.Wagner, A pedagogical implementation of A5/1, `http://www.scard.org/gsm/a51.html`, 1999.

[4] E.Biham, O.Dunkelman, Cryptanalysis of the A5/1 GSM stream Cipher, *Lecture notes in Computer Science*, vol-1977, 2000, pp. 43-51, (Indocrypt 2000).

[5] J. Golic, Cryptanalysis of Alleged A5 Stream Cipher, EUROCRYPT'97, LNCS 1233, pp.239.

[6] K.B.Athreya, P.E.Ney, Branching Processes, Berlin: Springer-Verlag, 1972.

[7] T.H.Harris, The Theory of Branching Processes. Berlin: Springer-Verlag, 1963.

[8] T.Pornin, J. Stern, Software-Hardware Trade-Offs: Application to A5/1 Cryptanalysis, CHES 2000, LNCS 1965, pp.318-327

[9] P. Ekdahl and T. Johansson. Another attack on A5/1. *In Proceedings of International Symposium on Information Theory*, page 160. IEEE, 2001.

[10] Cover, T.M., Thomas, J.A.: Elements of Information Theory. 2nd edn. Wiley Series in Telecommunications and Signal Processing. Wiley-Interscience (2006)

[11] A. Maximov, T. Johansson, and S. Babbage. *An Improved Correlation Attack on A5/1.* In Proc. of SAC'04, volume 3357 of LNCS, pages 239–255. Springer-Verlag, 2005.

6. Prílohy

6.1 Príloha A

V nasledujúcich tabuľkách prvý riadok určuje bity registra R_1, druhý riadok určuje bity R_2 a tretí riadok určuje bity R_3.

Príklad stavu s najmenším počtom nadbytočných lineárne závislých rovnic pre 1.verziu implementácie:

0	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0				
1	1	0	1	1	0	0	1	0	1	0	1	0	0	0	1	1	1	1	0	1	0	
1	1	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	1	0	1	1	1

Príklad stavu s najväčším počtom nadbytočných lineárne závislých rovnic pre 1.verziu implementácie:

1	1	0	0	1	0	0	0	0	1	1	0	1	0	0	1	1	0	0				
0	0	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1	1	0	
0	1	0	1	1	0	1	0	1	0	1	0	0	1	0	0	0	0	1	0	0	0	1

Príklad stavu s najmenším počtom nadbytočných lineárne závislých rovnic pre 2.verziu implementácie:

1	0	1	1	0	0	0	0	1	1	0	1	0	0	0	1	1	1	0				
1	0	0	1	0	1	1	0	1	0	1	0	1	0	1	0	0	0	1	1	0	1	
1	1	1	1	0	0	0	1	0	1	0	1	0	0	1	1	0	1	1	0	1	1	0

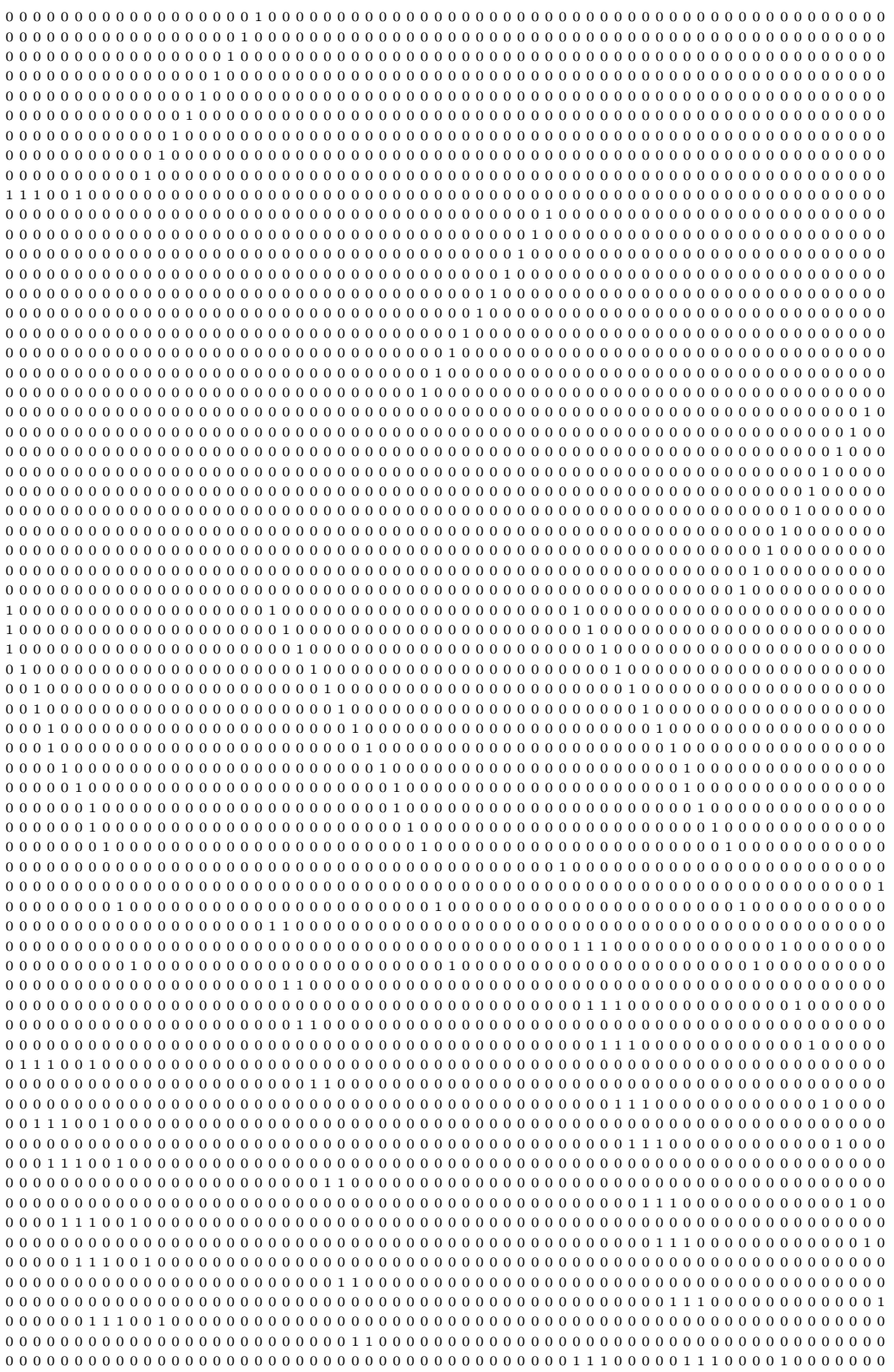
Príklad stavu s najväčším počtom nadbytočných lineárne závislých rovnic pre 2.verziu implementácie:

1	0	1	0	1	0	1	0	0	0	1	1	1	0	1	1	1	0	0				
1	1	1	0	1	1	1	1	1	1	1	0	0	1	0	0	1	1	0	1	0	1	
1	0	1	1	0	0	0	0	0	0	0	1	1	0	1	1	1	1	1	0	1	1	0

Na druhej strane sa nachádza kompletná matica nasledujúceho vnútorného stavu:

0	1	0	0	1	1	1	0	1	1	1	0	0	1	0	1	0	1	0				
0	1	1	1	0	1	0	0	1	0	0	1	1	0	0	0	0	0	1	1	1	0	
0	1	0	1	0	1	0	0	1	0	0	0	1	1	0	1	0	0	0	0	1	1	1

Matica vznikla v situácii, keď sme poznali *hádané bity* a počas prechádzania stromom sme išli vždy po správnej vetve. Poradie stĺpcov matice je $R_1^{(101)}[18], \ldots, R_1^{(101)}[0]$, $R_2^{(101)}[21], \ldots, R_2^{(101)}[0]$, $R_3^{(101)}[22], \ldots, R_3^{(101)}[0]$.



6.2 Príloha B

V dodatku B nájdeme bitovú reprezentáciu zvyšných troch nadbytočných lineárne závislých rovníc z podkapitoly s názvom Problém s existenciou lineárne závislých rovníc, ktorá sa nachádza v kapitole Útok na šifru A5/1 od Golića. Poradie stĺpcov matice je $R_1^{(101)}[18], \ldots, R_1^{(101)}[0]$, $R_2^{(101)}[21], \ldots, R_2^{(101)}[0]$, $R_3^{(101)}[22], \ldots, R_3^{(101)}[0]$.

Uvažovaný vnútorný stav $R^{(101)}$:

R_1	0	1	0	1	0	1	0	0	1	1	1	0	1	1	1	0	0	1	0				
R_2	0	1	1	1	0	0	0	0	0	1	1	0	0	1	0	0	1	0	1	1	1	0	
R_3	1	1	1	0	0	0	0	1	0	1	1	0	0	0	1	0	0	1	0	1	0	1	0

Obr. 6.1: Bity vnútorného stavu $R^{(101)}$.

Bitová reprezentácia druhej nadbytočnej lineárne závislej rovnice, ktorá sa vyskytla v 62.riadku matice:

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1

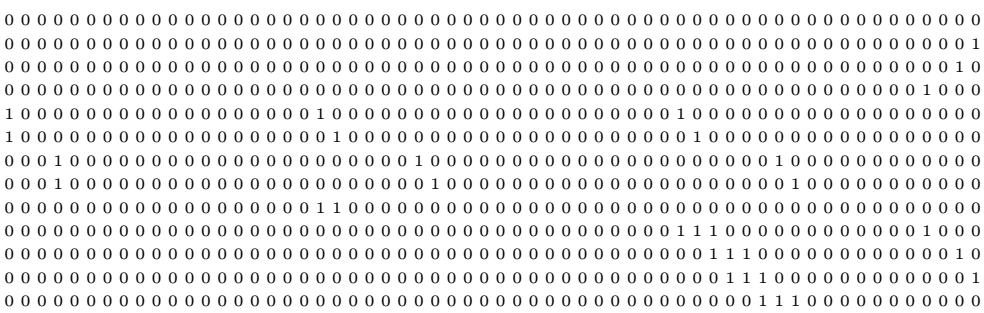
Bitová reprezentácia minimálnej podmnožiny rovníc sústavy, z ktorých sa dá odvodiť uvažovaná nadbytočná lineárne závislá rovnica:

1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Bitová reprezentácia tretej nadbytočnej lineárne závislej rovnice, ktorá sa vyskytla v 63.riadku matice:

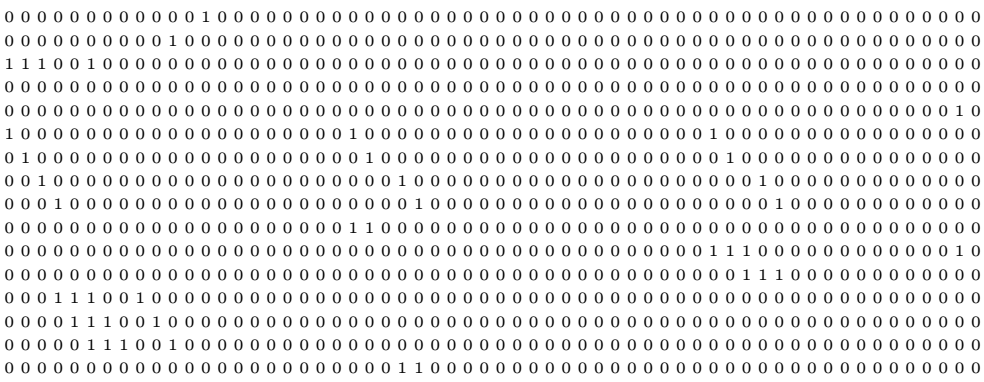
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Bitová reprezentácia minimálnej podmnožiny rovníc sústavy, z ktorých sa dá odvodiť uvažovaná nadbytočná lineárne závislá rovnica je na ďalšej strane:



Bitová reprezentácia poslednej nadbytočnej lineárne závislej rovnice, ktorá sa vyskytla v 64.riadku matice:

Bitová reprezentácia minimálnej podmnožiny rovníc sústavy, z ktorých sa dá odvodiť uvažovaná nadbytočná lineárne závislá rovnica:



6.3 Príloha C

Homogénna časť matice typu 64x64, ktorá sa používa v závere korelačného útoku na získanie *tajného kľúča* $K = (k_1, \ldots, k_{68})$.