УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ

Јована Г. Рибаћ

НАПАД НА ГЕНЕРАТОРЕ СА КОНТРОЛОМ ТАКТОВАЊА

мастер рад

Београд, 2023.

Ментор:

др Миодраг Живковић, редовни професор
Универзитет у Београду, Математички факултет

Чланови комисије:

др Саша Малков, ванредни професор
Универзитет у Београду, Математички факултет

др Стефан Мишковић, доцент
Универзитет у Београду, Математички факултет

Датум одбране: септембар 2023.

Мојим родитељима

Наслов мастер рада: Напад на генераторе са контролом тактовања

Резиме: Криптографија, проточна шифра, као и генератори са контролом тактовања су значајни фактори савремених система за заштиту података. Међутим, ови системи нису потпуно безбедни, нису непробојни, па самим тим јавља се потреба за проучавањем различитих врста напада, у циљу унапређења и јачања безбедности ових система. Циљ овог рада јесте анализа и разумевање неких напада на генератор $P1 \rightarrow P2$ (енг. Shrinking generator, генератор SG). Приказују се два напада, са различитим приступима. Један се заснива на упрошћеној верзији Левенштајновог растојања. Други напад је алгебарски и заснива се на масовном решавању система линеарних једначина по модулу 2.

Кључне речи: криптографија, генератор псеудослучајног низа, проточна шифра, криптоаналитички напад, померачки регистар са линеарном повратном спрегом (LFSR), генератор $P1 \rightarrow P2$, Левенштајново растојање.

Садржај

1 Увод **1**
1.1 Криптографија 1
1.2 Проточна шифра 2
1.3 Генератор низа кључа 3
1.4 Померачки регистар са линеарном повратном спрегом 4
1.5 Генератори са контролом тактовања 5
1.6 Генератор *P1* $\rightarrow$ *P2* 7
1.7 Криптоанализа 9

2 Напади на генератор са контролом тактовања **11**
2.1 Напад на регистар који тактује *P1* 11
2.2 Напад на тактовани регистар *P2* 22

3 Имплементације напада **25**
3.1 Имплементација напада на тактујући регистар *P1* 25
3.2 Имплементација напада на тактовани регистар P2 30

4 Закључак **36**

Библиографија **38**

Прилог **40**
Прилог 1 40
Прилог 2 51

Глава 1

Увод

Криптографија и криптоанализа су две важне дисциплине које чине основу за сигурност информација у дигиталном свету.

Криптографија је област чији је главни задатак да осигура приватност и интегритет поверљивих порука. Кроз векове, од појаве писама, јавила се потреба да се поруке заштите од погледа других људи. Данас, у доба дигиталне ере, криптографија је постала значајна област која омогућава сигурну комуникацију и заштиту података у свим сферама живота, укључујући банкарство, електронску трговину, војну комуникацију, свако слање и примање порука преко разних комуникационих апликација, и многе друге примене.

Са друге стране, криптоанализа је област чији је главни циљ откривање скривених порука без познавања тајних података, који су обично потребни да би се приступило скривеним порукама. На тај начин проналази слабости у комуникацији, па самим тим доприноси и унапређењу сигурности комуникације.

1.1 Криптографија

Криптографија се бави трансформацијом (шифровањем) података које треба послати у нечитљив облик, који називамо шифрат. Податке које треба послати називамо отворени текст. Кључна компонента криптографије је да се шифра може шифровати у оригиналну поруку само уз помоћ одговарајућег кључа.

Криптографски систем је пар чији су елементи алгоритам шифровања и алгоритам дешифровања. Ови алгоритми зависе већином од кључа којим се

бира конкретна шифарска трансформација у оквиру система. Криптографски системи у зависности од врсте трансформација и врсте кључа могу бити симетрични и асиметрични.

Асиметрични криптографски системи користе пар кључева, јавни кључ за шифровање поруке и приватни кључ за дешифровање. Њихова мана је не толико добра брзина, која је у комуникацијама јако битна.

Симетрични криптографски системи користе исти тајни кључ за шифровање и дешифровање порука. То је ефикасан и брз начин, међутим неопходно је да се обезбеди сигуран пренос кључа између пошиљаоца и примаоца.

Симетрични криптографски системи се могу поделити, на основу тога како трансформишу отворени текст, на проточне шифре и блоковске шифре. Проточна шифра трансформише отворени текст симбол по симбол, односно најчешће бит по бит. Блоковска шифра примењује се на симболе отвореног текста груписане у блокове.

1.2 Проточна шифра

Проточна шифра је криптографски алгоритам, где две комуникационе стране најпре бирају неки генератор псеудослучајних бројева, као и његово почетно стање, односно кључ. Изабраним генератором на основу кључа обе стране изгенеришу исти псеудослучајни низ битова, низ кључа. Сваки бит шифрата добија се применом операције ексклузивне дисјункције (XOR), на бит отвореног текста и одговарајући бит низа кључа. На пријему се примљени бит шифрата, применом операције XOR са истим битом низа кључа, трансформише у оригинални бит отвореног текста.

Проточна шифра може бити синхрона и самосинхронишућа проточна шифра. Синхрона проточна шифра сабира бит по бит отвореног текста операцијом XOR са низом кључа да би се добио шифрат. Самосинхронишућа проточна шифра у процесу шифровања користи и неке претходне битове отвореног текста. Предност оваквог система јесте да један бит шифрата не зависи само од једног бита отвореног текста, па је самим тим отпорност оваквог система на напад већа [1].

Главни елемент у дизајну проточне шифре је генератор низа кључа.

1.3 Генератор низа кључа

Генератор низа кључа је коначни аутомат чије почетно стање је одређено кључем.

Циљ генератора јесте да генерише низ битова који изгледају као да су случајни, иако се генеришу коришћењем детерминистичког алгоритма. Сваки генератор низа кључа треба да буде довољно сигуран, што значи да генерисани битови треба да буду тешки за предвидети, чак и ако је познато како генератор функционише.

Важно је користити поуздане и сигурне генераторе низа кључа у криптографским системима, како би се онемогућио или бар битно отежао напад на систем. Сваки добар генератор низа кључа има за циљ да обезбеди добре статистичке особине низа кључа као што су:

- дуг период,
- добра расподела битова,
- балансираност,
- случајност битова,
- велика линеарна сложеност.

Балансираност низа битова значи да има приближно исти број појављивања јединица и нула, док се случајност односи на то да низ изгледа случајно, чак и када се анализира велика количина генерисаних битова. Линеарна сложеност је често коришћена као мера сигурности излазне секвенце. Односи се на меру колико је тешко предвидети излазни низ битова који генерише неки генератор, користећи линеарну методу. У суштини то је најмањи број излазних битова који су потребни да би се могли линеарно предвидети будући битови.

Постоје многи генератори базирани на различитом приступу генерисања псеудослучајних низова битова. У наставку се разматрају генератори засновани на примени померачких регистара са линеарном повратном спрегом.

1.4 Померачки регистар са линеарном повратном спрегом

Померачки регистар са линеарном повратном спрегом (енг. linear feedback shift register, LFSR) [3], у даљем тексту регистар, је један од најчешћих типова регистара.

Познат је по својој брзини и ефикасности, што га чини погодним за апликације које захтевају брзо генерисање битова. Генерише излазне низове битова са великим периодом, добром расподелом битова и балансираношћу.

Битови регистра се на одређеним позицијама (у зависности од коефицијената карактеристичног полинома) комбинују користећи операцију XOR и померају кроз регистар при сваком такту, померајући се само у једном смеру, улево или удесно. Повратна спрега је кључна компонента регистра. То је механизам који комбинује одређене битове регистра и враћа резултат на улаз регистра. Без смањења општости, претпоставићемо да се битови у регистру померају улево. Резултат комбиновања изабраних битова уписује се на најдешњу, управо ослобођену позицију. Такт је сигнал који покреће извршавање описане операције.

Детаљније, после сваког такта извршавају се следеће операције:

- *Излаз*: најстарији бит, бит на левом крају регистра се узима као излаз.
- *Израчунавање новог бита*: нови бит се рачуна операцијом ексклузивне дисјункције (XOR) са битовима одређених повратном спрегом.
- *Померање*: битови у регистру се померају за једно место улево. Ово померање ослобађа место на десном крају регистра за нови бит.
- *Унос новог бита:* нови бит се уноси на место које је ослобођено померањем битова.

Посматрајмо регистар дужине N, са почетним стањем $a_0, a_1, a_2, \ldots, a_{n-1}$. Ако је $p(x) = c_0 + c_1 \cdot x + c_2 \cdot x^2 + \cdots + c_n \cdot x^n$ карактеристични полином који одговара повратној спрези, тада су $c_0, c_1, c_2, \ldots, c_{n-1}$ коефицијенти повратне спреге (0 тамо где нема повратне спреге, 1 на позицијама са којих се узима повратна спрега). Такт покреће израчунавање бита a_n по формули $a_n = a_0 \cdot c_0 + a_1 \cdot c_1 + \cdots + a_{n-1} \cdot c_{n-1}$, бит a_0 се узима као излаз из регистра и сви остали битови се померају за једно место улево. Нови бит a_n се смешта на позицију

где се налазио бит a_{n-1} пре померања. Пре следећег такта, стање регистра је $a_1, \ldots, a_n$. Такт покреће израчунавање $a_{n+1} = a_1 \cdot c_0 + a_2 \cdot c_1 + \cdots + a_n \cdot c_{n-1}$. Бит a_1 се узима као излаз из регистра. Затим се сви битови померају улево, а нови бит се смешта на ослобођену позицију, где је био a_n пре померања. Сваки наредни такт покреће извршавање описане операције, после чега се добија нови излазни бит.

Дужина регистра утиче на период генерисане излазне секвенце. Период је истовремено број тактних циклуса потребних да регистар понови почетно стање и да поново генерише исту излазну секвенцу.

Линеарна сложеност излазног низа је дужина одсечка излазног низа довољна да се предвиди следећи бит линеарном методом. Регистар дужине L, има линеарну сложеност L. Сам регистар никада не би требало да се користи као генератор низа кључа, због лоше линеарне сложености генерисаног низа, која је једнака степену карактеристичног полинома.

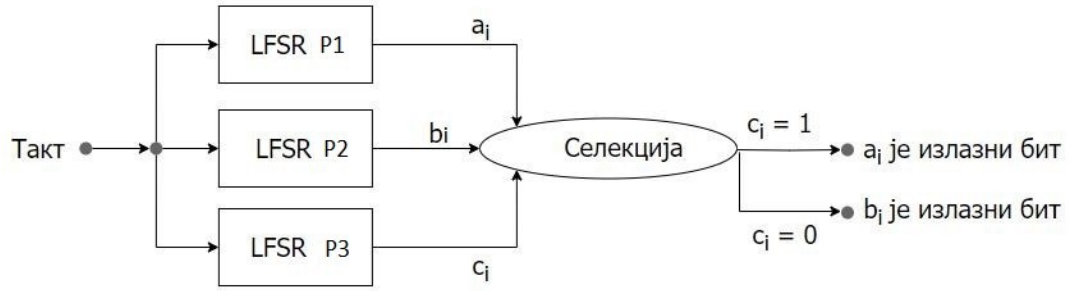
У наставку се претпоставља да излазни низ регистра дужине L има период највеће могуће дужине, $2^L - 1$. Ова претпоставка је еквивалентна са претпоставком да је карактеристични полином регистра примитиван, односно да је полином x генератор мултипликативне групе коначног поља одређеног карактеристичним полиномом регистра.

Да би се смањила слабост која је повезана са линеарношћу регистра, у генератор се мора додати нелинеарна компонента, као што је нерегуларно тактовање регистра, видети одељак 1.5.

1.5 Генератори са контролом тактовања

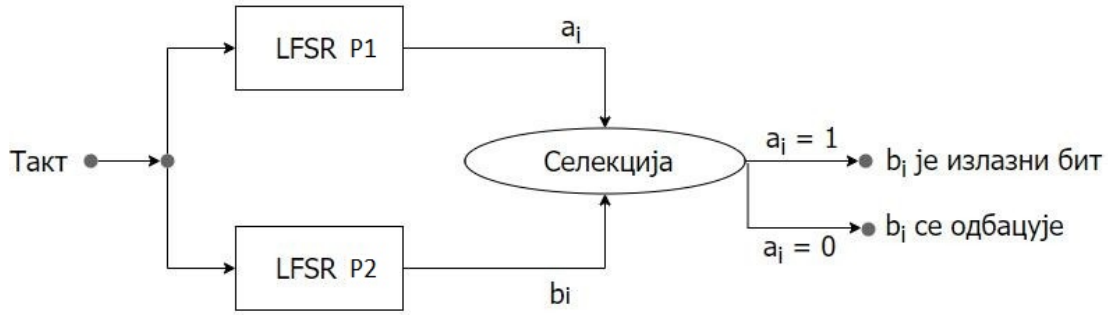
Постоји више врста генератора низа кључа базираних на контроли тактовања регистра.

Генератор ASG (*Alternating Step* Generator) је генератор који садржи три регистра, означимо их са *P1*, *P2*, *P3*, видети слику 1.1 [6]. Регистар *P3* је тактујући регистар, који сваким излазним битом врши селекцију излазних битова преостала два регистра, одређујући који ће бит бити излазни бит генератора. На пример, ако тактујући регистар генерише бит 1, тада излазни бит који генерише регистар *P1* постаје излазни бит генератора. У супротном, ако генерише 0, тада излазни бит који генерише регистар *P2* постаје излазни бит генератора.



Слика 1.1: Генератор ASG

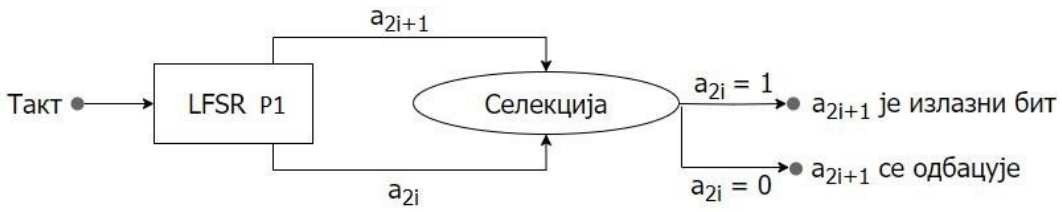
Генератор SG (*Shrinking* Generator) састоји се од два регистра, видети слику 1.2. Детаљније је објашњен у одељку 1.6.



Слика 1.2: Генератор SG

Генератор SSG (*Self Shrinking* Generator) има нешто једноставнију структуру, видети слику 1.3. Састоји се само од једног регистра *P1*, који и производи битове и врши селекцију тих битова, посматрајући парове битова које генерише. На пример, ако регистар генерише бит 1, тада ће наредни изгенерисани бит бити излазни бит генератора. У супротном, ако генерише 0, наредни бит се одбацује. Ако је пар изгенерисаних битова 10 или 11, тада ће излазни бит генератора бити 0 или 1 [10].

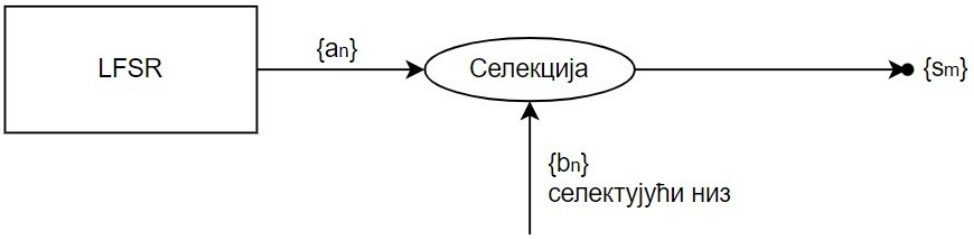
Сва три споменута генератора имају тактовани регистар, а разликују се по извору низа који га тактује. Основна идеја је да се обезбеди нелинеарност у генератору контролисањем излаза једног регистра, што обезбеђује већу сигурност. Сва три генератора производе низ кључа са високом линеарном сложеношћу, дугим периодом и добрим статистичким својствима.



Слика 1.3: Генератор SSG

Упркос својој једноставности и многобројним објављеним нападима, остају отпорни на криптоанализу, јер још увек не постоје познати напади који су довољно ефикасни када регистар генерише дуге низове са великим периодом понављања.

У овом раду разматра се други од споменута три генератора, који је у складу са теоријским моделом приказаним на слици 1.4.



Слика 1.4: Теоријски модел генератора са селекцијом

Ради једноставности, за SG генератор уводи се ознака генератор $P1 \rightarrow P2$.

1.6 Генератор $P1 \rightarrow P2$

Генератор $P1 \rightarrow P2$ је генератор са контролом тактовања заснованом на регистру, тачније генератор са нерегуларним тактовањем регистра [2].

Садржи два регистра који се регуларно тактују у сваком кораку, видети слику 1.2. Регистар $P1$ називамо регистар који тактује, док регистар $P2$ називамо тактовани регистар. Овај генератор функционише тако што се излазни бит регистра $P1$ користи да одреди да ли излазни бит другог регистра $P2$, треба копирати или не, у излазни низ генератора. Ову трансформацију излазног низа регистра $P2$, називамо селекција. Прецизније, селекција делује

тако да ако $P1$ генерише бит 1, тада ће бит генерисан регистром $P2$ бити излазни бит генератора. У противном, ако $P1$ генерише бит 0, тада се бит генерисан регистром $P2$ одбацује, тј. не налази се у низу кључа који генерише овај генератор.

Пример 1. *Нека су* $p_1(x) = 1 + x + x^2$ *и* $p_2(x) = 1 + x + x^3$ *карактеристични полиноми регистара P1 и P2. Нека су почетна стања* $\{10\}$ *и* $\{100\}$*. Тада је излазни низ битова регистра P1* $\{a_i\} = \{1, 0, 1 \dots\}$*, док је за P2 излазни низ битова* $\{b_i\} = \{1, 0, 0, 1, 0, 1, 1, \dots\}$*. Они се периодично поновљају на три, односно на седам позиција. Генерисање низа кључа* $\{s_j\}$ *приказано је на слици 1.5 [9].*

$\{a_i\}$: 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 ...
$\{b_i\}$: 1 ~~0~~ 0 1 ~~0~~ 1 1 ~~1~~ 0 0 ~~1~~ 0 1 ~~1~~ 1 0 ~~0~~ 1 0 ~~1~~ 1 ...
$\{s_j\}$: **1 01 11 00 01 10 10 1 ...**

Слика 1.5: Генерисање низа кључа

Ознаке које се користе у наставку:

- L_1, L_2 – дужина регистра $P1$, дужина регистра $P2$,
- $p_1(x)$, $p_2(x)$ – карактеристични полиноми регистара $P1$ и $P2$,
- $A = \{a_n\}$, $B = \{b_n\}$ – низови битова максималног периода које производе $P1$ и $P2$,
- $S = \{s_m\}$ – излазни низ битова из генератора $P1 \to P2$, низ кључа.

Полиноми $p_1(x), p_2(x) \in F_2[x]$ су примитивни, па самим тим се добија излазни низ битова регистара са највећим могућим периодом. Периоди излазних низова регистара су $T_1 = 2^{L_1} - 1$ и $T_2 = 2^{L_2} - 1$. Када је $\text{NZD}(L_1, L_2) = 1$ (највећи заједнички делилац), тада је период излазног низа из генератора $T = 2^{L_1 - 1} \cdot (2^{L_2} - 1)$ [2].

1.7 Криптоанализа

Криптоанализа је област која се бави проучавањем и анализом криптографског система са циљем откривања њихове слабости и проналажења начина да се шифра разбије. То је процес где се покушава трансформација шифрата у одговарајући отворени текст, при чему је кључ непознат. Ова област је веома битна у криптографији, јер доприноси унапређењу сигурности криптографских система.

Према *Kerhofsovom* принципу криптографски систем треба да буде безбедан, чак и ако је све у вези са системом јавно познато, осим кључа. Према овом правилу, сигурност система у потпуности се заснива на кључу.

Криптоаналитички напади су методе које криптоаналитичари примењују како би разбили криптографске шифре и открили кључеве криптографских система. Постоје различите врсте криптоаналитичких напада, а у зависности од тога којим информацијама нападач располаже, могу се поделити на [1]:

1. *Напад са познавањем само шифрата.* У овом типу напада, нападач нема приступ оригиналном отвореном тексту или било каквим другим информацијама, поседује само шифрат. Ово је најтежи тип напада, јер нападач нема никакве информације које би му помогле у разбијању шифре.

2. *Напад са познатим отвореним текстом.* У овом нападу, нападач поседује неке одговарајуће парове шифрата и отвореног текста које су шифроване истим кључем. Поред тога може имати још шифрата.

3. *Напад са познавањем изабраног отвореног текста.* Претпоставка у овом нападу јесте да нападач може по жељи одабрати одређени отворени текст и шифровати га. На тај начин, нападач има одговарајуће парове отвореног текста и шифрата.

Напад на проточне шифре је посебна врста напада који се фокусира да нападне слабост у генераторима низа кључа, да би се открио кључ који се користи у шифровању. Већина напада на проточне шифре се изводи под претпоставком да се познаје отворени текст и одговарајући шифрат. То је претпоставка да нападач има директан приступ излазу генератора. Рачунска сложеност напада се увек пореди са сложеношћу напада са потпуном претра-

гом. Ако је сложеност напада мања од сложености са потпуном претрагом, онда се напад сматра успешним [8].

У наставку овог рада, разматрају се неки од интересантних напада на проточне шифре засноване на генератору $P1 \rightarrow P2$. Разматрају се напади са познавањем отвореног текста и одговарајућег шифрата, што је еквивалентно одређивању почетног стања генератора на основу дела низа кључа. Циљ је истражити да ли се број почетних стања које треба анализирати може смањити испод броја различитих кључева.

Глава 2

Напади на генератор са контролом тактовања

У одељку 2.1 приказан је напад на регистар $P1$, са идејом да се за $P2$ провере сва могућа почетна стања. Овај напад се заснива на поједностављеној верзији Левенштајновог растојања између два низа, при чему се први трансформише у други само избацивањем неких подниза битова. У одељку 2.2 приказан је други напад, где је фокус на регистру $P2$. Напад се заснива на решавању система линеарних једначина за свако претпостављено почетно стање регистра $P1$.

2.1 Напад на регистар који тактује *P1*

Примена Левенштајновог напада на генератор са контролом тактовања, има за циљ да одреди могућа почетна стања тактујућег регистра, независно од тога шта је извор тактујућег низа.

Левенштајново растојање представља минималан број елементарних операција (уметање, брисање и замена) које су неопходне да се низ A дужине N трансформише у низ B дужине M. Левенштајново растојање има бројне примене, као што је детекција плагијата, провера правописних грешака, упоређивање фајлова, итд. У овом контексту, из скупа елементарних операција, искључене су уметање и замена.

Означимо са $d(A, B)$ Левештајново растојање између низова A и B. Нека су дати низови A=$a_1 \dots a_n$ и $B = b_1 \dots b_m$. A_i је префикс низа A дужине i, B_j је префикс низа B дужине j. Проблем проналаска $d(A_n, B_m)$ решава се

индукцијом. За $n = 0$, потребно је празан префикс A_0 трансформисати у B. Очигледно је за то потребно m операција уметања у низ A, према томе је $d(A_0, B_m) = m$. Слично, $d(A_n, B_0) = n$ јер је потребно n операција брисања у низу A. Да би се решио проблем индукцијом, потребно је да се израчунавање растојања префикса сведе на израчунавање растојања између краћих префикса. Операције које могу бити последње у низу операција да би се A_i трансформисао у B_j су:

- замена знака a_i знаком b_j, ако су различити, пошто је претходно префикс A_{i-1} трансформисан у B_{j-1},
- уметање знака b_j, пошто је претходно префикс A_i трансформисан у B_{j-1},
- брисање знака a_i, пошто је претходно префикс A_{i-1} трансформисан у B_j.

Дакле, најмањи број операција да се A_i трансформише у B_j рачуна се по формули:

$$d(A_i, B_j) = \min\{d(A_{i-1}, B_{j-1}) + c(i, j), d(A_i, B_{j-1}) + 1, d(A_{i-1}, B_j) + 1\},$$

где $c(i, j)$ представља број операција да се a_i трансформише у b_j, једнака је 1 ако је $a_i \neq b_j$, у супротном је 0. Матрица растојања за Левенштајново растојање је облика $M = (m_{i,j})$, где $m_{i,j} = d(A_i, B_j)$, $i = 0, 1, \ldots, n, j = 0, 1, \ldots, m$ [5].

Претпоставка за напад који се приказује, јесте да су познати карактеристични полиноми тактујућег регистра $P1$ и тактованог регистра $P2$, $p_1(x)$ и $p_2(x)$, као и да је познат део низа кључа $\{s_m\}$.

Напад се заснива на рачунању Левенштајновог растојања излазног низа битова регистра *P2*, $B = \{b_n\}$ и низа кључа $S = \{s_m\}$, где су искључене елементарне операције уметања и замене. Циљ је наћи све излазне низове из регистра *P1* који низ B, или неки његов префикс B_i, своде на низ S.

Битно је напоменути да у селектујућем низу регистра $P1$ може да буде највише $L_1 - 1$ узастопних нула. Ово произилази из чињенице да почетно стање које се састоји само од низа нула се не разматра. Са тим почетним стањем регистар даје на излазу тривијални низ нула. На основу ове рестрикције, рачунање Левенштајновог растојања се ограничава на $L_1 - 1$ узастопних брисања.

Нека су B_i и S_j префикси низова $B = \{b_1, \ldots, b_n\}$, односно $S = \{s_1, \ldots, s_m\}$, дужине редом i, односно j. Дужине низова B и S су редом N и M. Прилагођена рекурента релација израчунавања матрице M, без уметања и са ценом замене $c(i,j) = 0$ ако је $b_i = s_j$, $c(i,j) = \infty$ у противном:

$$d(B_i, S_j) = \min\{d(B_{i-1}, S_{j-1}) + c(i,j), d(B_{i-1}, S_j) + 1\},$$

где је $0 \leq i \leq N$, $0 \leq j \leq M$.

С обзиром да се напад заснива на поједностављеној верзији Левенштајновог растојања, уместо опште матрице растојања M користи се упрошћена матрица растојања $W = (w_{i,j})$, где је $w_{i,j}$ Левенштајново растојање префикса B_{i+j} и S_j, $i = 0, 1, \ldots, N - M$, $j = 1, 2, \ldots, M$. Елементи матрице W су $w_{i,j} = m_{i+j,j}$, односно $m_{i,j} = w_{i-j,j}$.

Главне рестрикције при рачунању Левенштајновог растојања низова $\{b_n\}$ и $\{s_m\}$, дужина N и M:

- $0 < M \leq N$,
- дозвољено је само брисање у низу $\{b_n\}$, замена и уметање нису дозвољене,
- максималан број узастопних брисања у низу $\{b_n\}$ је $L_1 - 1$,
- није дозвољено брисање знака b_i, пошто је претходно префикс B_{i-1} трансформисан у S_j. Уместо тога проверава се једнакост b_i и s_j, а брисање се врши у префиксу B_{i-1}. Дакле, директна грана брисања не постоји, већ је она замењена брисањем и ценом замене $c(i,j) = 0$ ако је $b_i = s_j$, иначе $c(i,j) = \infty$.

Основна разлика матрица M и W је у томе што се елемент $m_{i,j}$ матрице M рачуна на основу елемената $m_{i-1,j}$, $m_{i-1,j-1}$ и односа битова a_i и b_j, док се елемент $w_{i,j}$ матрице W рачуна на основу узастопних брисања битова у префиксу B_{i+j-1} и односа битова b_{i+j} и s_j.

Анализирајмо матрицу M, узимајући у обзир рестрикције за рачунање упрошћеног Левенштајновог растојања.

- Зато што је дозвољено само брисање, елементи $m_{i,j}$ где је $i < j$ имају вредност ∞.
- Зато што је $0 < M$, елементи $m_{i,0}$ где је $0 \leq i \leq N$ имају вредност ∞.

- Како би се одредиле димензије матрице W, полази се од дефиниције $m_{ij} = d(B_i, S_j)$, $0 \leq i \leq N$, $0 \leq j \leq M$. Као што је речено, елементи матрице W су $w_{i,j} = m_{i+j,j}$, при чему индекси задовољавају ограничења $1 \leq i + j \leq N$, $1 \leq j \leq M$. Тада се добија ограничење $0 \leq i \leq N - M$, према томе, димензије матрице W су $(N - M + 1) \times M$.

- Елемент $m_{1,1} = \min\{m_{0,0} + c(1,1), m_{0,1} + 1\})$ представља растојање префикса B_1 и S_1. Због забране уметања, игноришу се сви елементи $m_{i,j}$ где је $i < j$, па је растојање префикса B_1 и S_1 само резултат провере првих битова, $m_{1,1} = c(1,1)$. Ако су битови једнаки, $c(1,1) = 0$, иначе је $c(1,1) = \infty$ због услова да замена није дозвољена. Стога је $w_{0,1} = c(1,1)$.

- Елемент $m_{2,2} = \min\{m_{1,1} + c(2,2), m_{1,2} + 1\}$ представља растојање префикса B_2 и S_2. На исти начин растојање B_2 и S_2 зависи од растојања префикса B_1, S_1 и једнакости битова $c(2,2)$, па је $m_{2,2} = m_{1,1} + c(2,2)$. Стога је $w_{0,2} = w_{0,1} + c(2,2)$.

- Елемент $m_{3,3} = \min\{m_{2,2} + c(3,3), m_{2,3} + 1\}$ представља растојање префикса B_3 и S_3. Укључујући услове, растојање B_3 и S_3 је $m_{3,3} = m_{2,2} + c(3,3)$. Стога је $w_{0,3} = w_{0,2} + c(3,3)$.

- Елемент $m_{2,1} = \min\{m_{1,0} + c(2,1), m_{1,1} + 1\}$ представља растојање префикса B_2 и S_1. Како је $m_{1,0} = \infty$, тада је $m_{2,1} = m_{1,1} + 1$. Односно $w_{1,1} = w_{0,1} + 1 = c(1,1) + 1$. Дакле растојање префикса B_2 и S_1 се добија на основу једнакости првих битова и брисањем последњег бита у B_2. С обзиром на забрану брисања последњег бита, треба проверити $c(2,1)$ и брисати бит b_1. Ако означимо $P_k(b_{i+j}, s_j) = k + c(i + j, j)$ као k брисања пре бита b_{i+j}, тада је $w_{1,1} = P_1(2,1)$.

- Елемент $m_{3,1} = \min\{m_{2,0} + c(3,1), m_{2,1} + 1\}$ представља растојање префикса B_3 и S_1. Како је $m_{2,0} = \infty$, тада је $m_{3,1} = m_{2,1} + 1$, односно $w_{2,1} = w_{1,1} + 1 = c(1,1) + 1 + 1$. На основу претходног објашњења и услова да брисање мора да буде пре последњег бита, растојање за B_3 и S_1 се добија на основу два брисања у префиксу B_2 и једнакости битова $c(3,1)$. Дакле, $w_{2,1} = P_2(b_3, s_1)$.

- Елемент $m_{3,2} = \min\{m_{2,1} + c(3,2), m_{2,2} + 1\}$ представља растојање префикса B_3 и S_2. Тада је $w_{1,2} = \min\{w_{1,1} + c(3,2), w_{0,2} + 1\}$, односно

$w_{1,2} = \min\{w_{1,1} + c(3,2), w_{0,1} + c(2,2) + 1\}$. Анализирајмо $w_{0,1} + c(2,2) + 1$. Ако већ постоји растојање префикса B_1 и S_1, да би се добило растојање префикса B_3 и S_2 потребно је проверити битове $c(2,2)$ и брисати последњи бит b_3. На исти начин као и претходно, то се замењује једним брисањем бита из префикса B_2 и једнакости $c(3,2)$, дакле $w_{1,2} = \min\{w_{1,1} + P_0(b_3, s_2), w_{0,1} + P_1(b_3, s_2)\}$.

Уопштавајући наведене специјалне случајеве, закључује се да се сваки елемент матрице W може рекурзивно израчунати на основу формула:

$$
\begin{aligned}
&w_{i,1} = P_i(b_{i+1}, s_1), i = 0, \ldots, L_1 - 1 \\
&w_{i,1} = \infty, i = L_1, \ldots, N - M \\
&w_{0,j} = w_{0,j-1} + P_0(b_j, s_j), j = 2, \ldots, M \\
&w_{i,j} = \min_{k=0,\ldots,L_1-1} \{w_{i-k,j-1} + P_k(b_{i+j}, s_j)\}, i = 1, \ldots, N - M, j = 2, \ldots, M \\
&P_k(b_{i+j}, s_j) = \left\{ \begin{array}{ll} k + c(i+j, j) & , b_{i+j} = s_j \\ \infty & , b_{i+j} \neq s_j \end{array} \right\}
\end{aligned}
\tag{2.1}
$$

Израз $P_k(b_{i+j}, s_j)$ је цена брисања k битова који се налазе пре бита b_{i+j}, на коју се додаје цена замене b_{i+j} са s_j.

Карактеристике везане за матрицу W:

- последња колона даје растојања између S и префикса B_{i+M} где је $i = 0, \ldots, N - M$,
- последњи елемент последње колоне представља растојање између S и B,
- сваки елемент матрице који се не налази у последњој врсти и последњој колони одговара растојању секвенци $b_1, b_2, \ldots, b_{i+j}$ и $s_1, s_2, \ldots, s_j$, за $i = 0, \ldots, N - M$ и $j = 1, \ldots, M - 1$.

Матрица омогућава одређивање растојања, али је компликовано кроз матрицу одређивање свих скупова операција за растојање низова B и S. Стога, уместо рачунања матрице растојања излазног низа бита из *P2* и низа кључа, проблем се своди на тражење најкраћег растојања између два чвора у специјалном графу одређеном матрицом W [6]. Овај специјални граф је усмерен тежински граф, који има чворова колико матрица W има елемената и додата још два чвора. Та два чвора ћемо означити као $Pocetak$ и $Kraj$.

Карактеристике графа:

- сваки чвор, осим $Pocetak$ и $Kraj$, $(i+j, j)$, где је $i = 0, 1, \ldots, N-M$ и $j = 1, 2, \ldots, M$, представља елемент $w_{i,j}$ матрице W,
- чвор $Pocetak$ је повезан са чворовима који одговарају елементима прве колоне матрице W, који нису ∞,
- чвор $Kraj$ је повезан са чворовима који одговарају елементима последње колоне матрице W, који нису ∞,
- сваки чвор може да има више излазних грана у зависности од рестрикције броја узастопних брисања. Чвор $w_{i,j}$ може имати излазне гране само ка чворовима $w_{i+k,j+1}$, $k \in \{0, \ldots, L_1 - 1\}$. Ово се не односи на елементе из последње колоне матрице W, који имају само једну излазну грану ка чвору $Kraj$,
- тежине грана између два чвора представљају број узастопних брисања и директно долазе од израза P_k. Ако елемент $w_{i,j}$ матрице W није ∞, свака грана између $w_{i-k,j-1}$ и $w_{i,j}$ има тежину k, где је $k \in \{0, \ldots, L_1 - 1\}$.

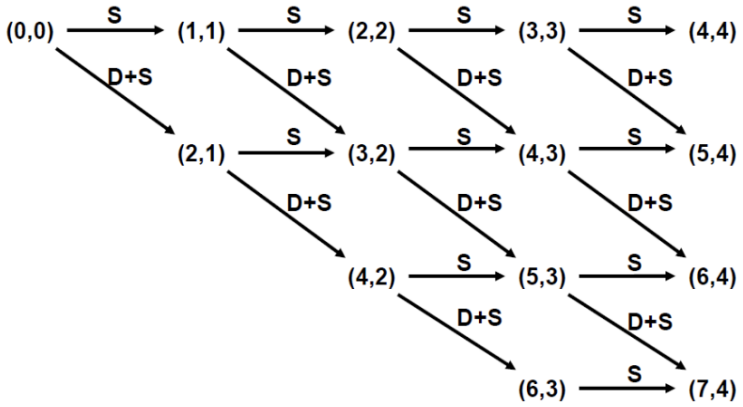
С обзиром на овако одређене тежине у графу, закључује се да све путање од чвора $Pocetak$ до чвора $Kraj$, заправо представљају оптималне путеве, односно путеве са најмањом ценом.

На основу овако дефинисаних тежина грана, проналазак оптималних путева у графу директно утиче на креирање селектујућих низова. Креирање селектујућег низа на основу оптималног пута се изводи на следећи начин:

- ако грана на путу има тежину 0 – у селектујући низ се уписује бит 1,
- ако грана на путу има тежину k – у селектујући низ се уписује k нула и једна јединица на крај,
- ако је грана на путу последња грана – у селектујући низ се не уписује ни један бит. Зато што је у том случају цео низ B или неки префикс B_i сведен на S.

Пример 2. *Пример графа где је дозвољено само једно узастопно брисање, приказан је на слици 2.1 [7]. Хоризонталне гране S, на пример између чворова*

$(i+j-1, j)$ *и* $(i+j, j)$ *представљају однос битова* b_{i+j} *и* s_j*, док дијагоналне гране D+S, на пример између чворова* $(i+j-2, j-1)$ *и* $(i+j, j)$*, представљају однос битова* b_{i+j} *и* s_j *и једно брисање бита* b_{i+j-1}*. Брише се бит* b_{i+j-1}*, зато што чвор* $(i+j-2, j-1)$ *означава да је подсеквенца* B_{i+j-2} *сведена на подсеквенцу* S_{j-1}*. Како би се подсеквенца* B_{i+j} *свела на подсеквенцу* S_j*, на основу претходног чвора, потребно је обрисати бит између. Ако се посматра граф са два максимална узастопна брисања, дијагонална грана* $(i+j-3, j-1)$ *и* $(i+j, j)$ *би имала тежину 2, јер означава брисање подсеквенце* $\{b_{i+j-2}, b_{i+j-1}\}$ *, чија је дужина 2.*



Слика 2.1: Граф са највише једним узастопним брисањем

Пример 3. *Нека је низ кључа* S : 1101011*,* $M = 7$*, а излазни низ битова регистра P2 на основу неког почетног стања* B : 1110110111*,* $N = 10$*. Нека је карактеристични полином регистра P1 степена 2, дакле* $L_1 = 2$*. Тада је:*

$$W = \begin{Bmatrix} w_{0,1}=0 & w_{0,2}=0 & w_{0,3}=\infty & w_{0,4}=\infty & w_{0,5}=\infty & w_{0,6}=\infty & w_{0,7}=\infty \\ w_{1,1}=1 & w_{1,2}=1 & w_{1,3}=1 & w_{1,4}=1 & w_{1,5}=\infty & w_{1,6}=\infty & w_{1,7}=\infty \\ w_{2,1}=\infty & w_{2,2}=\infty & w_{2,3}=\infty & w_{2,4}=2 & w_{2,5}=2 & w_{2,6}=2 & w_{2,7}=2 \\ w_{3,1}=\infty & w_{3,2}=\infty & w_{3,3}=\infty & w_{3,4}=\infty & w_{3,5}=\infty & w_{3,6}=3 & w_{3,7}=3 \end{Bmatrix}$$

На основу последњег поља у последњој колони, закључује се да је минималан број брисања у B *да би се добио низ* S *заправо* 3*.*

Као што је приказано, нека поља матрице W су ∞. Ово произилази из услова да максималан број узастопних нула у селектујућем низу може да буде највише $L_1 - 1$, па самим тим не сме да буде више од $L_1 - 1$ узастопних брисања у низу B.

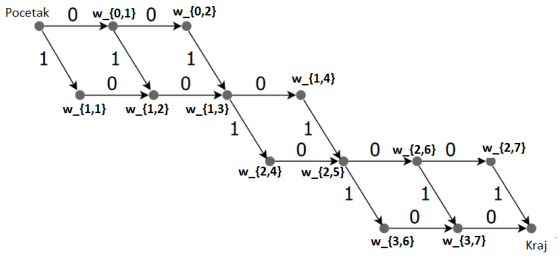
У наставку је неколико примера израчунавања елемената матрице:

- $w_{0,1} = P_0(b_1, s_1)$ – *битови b_1 и s_1 су једнаки.*
- $w_{1,1} = P_1(b_2, s_1)$ – *битови b_2 и s_1 су једнаки, да би се низ $\{b_1, b_2\}$ свео на $\{s_1\}$, потребно је једно брисање у префиксу $\{b_1\}$, јер је дужине 1.*
- $w_{2,1} = P_2(b_3, s_1)$ – *битови b_3 и s_1 су једнаки, да би се низ $\{b_1, b_2, b_3\}$ свео на $\{s_1\}$, потребна су два брисања у префиксу $\{b_1, b_2\}$. Међутим, с обзиром на услов да је максималан број узастопних брисања 1, ово поље матрице је ∞.*
- $w_{0,2} = w_{0,1} + P_0(b_2, s_2)$ – *битови b_2 и s_2 су једнаки. Да би се низ $\{b_1, b_2\}$ свео на $\{s_1, s_2\}$, потребно је да се пореде префикси $\{b_1\}$ и $\{s_1\}$, односно битови b_1 и s_1. Тај однос даје елемент $w_{0,1}$.*
- $w_{1,2} = \min\{w_{0,1}+P_1(b_3, s_2), w_{1,1}+P_0(b_3, s_2)\}$; $w_{0,1}+P_1(b_3, s_2)= 0+1$ *јединица одговара цени брисања једног бита b_2;* $w_{1,1} + P_0(b_3, s_2)= 1 + 0$ *јединица представља број брисања у подсеквенци $\{b_1, b_2\}$.*

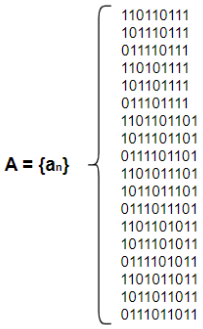
Граф заснован на матрици растојања у овом примеру приказан је на слици 2.2. Овај граф има 18 оптималних путева од чвора $Pocetak$ до чвора $Kraj$. Сваком оптималном путу одговара могући почетни садржај регистра $P1$. Рачунање 18 селектујућих низова може се извести из графичког приказа 2.2 на следећи начин. Хоризонталне гране у оптималном путу, у нашем случају, представљају бит 1 *у селектујућем низу јер им је тежина 0. Свака дијагонална грана у оптималном путу има цену 1, па се у селектујућем низу налази пар битова* 01*. Свака дијагонална грана из чвора последње колоне нема утицај на селектујући низ, зато што из тих чворова већ постоји сведени префикс или цео излазни низ регистра P2 на низ кључа S. Скуп могућих селектујућих низова приказан је на слици 2.3. Првих* 6 *оптималних путева пролазе кроз чвор $w_{2,7}$, осталих 12 путева пролазе кроз чвор $w_{3,7}$.*

Нису сви овако добијени селектујући низови могући излазни низови из регистра P1, тј. не задовољавају сви ти низови рекурентну релацију одређену

карактеристичним полиномом регистра P1. Да би низ представљао решење треба да буде у складу са карактеристичним полином регистра P1.



Слика 2.2: Граф



Слика 2.3: Селектујући низови

У наставку је приказан псеудокод овог напада, а у одељку 3.1 је приказана његова имплементација.

Алгоритам 1 Напад на регистар $P1$

Улаз: $p_1(x)$, $p_2(x), S$ дужине M

Излаз: Скуп парова $Pairs = (i_a, i_b)$ почетних стања за регистре $P1$ и $P2$ који генеришу S

```
Генерисање скупа I_b свих могућих почетних стања регистра P2
Иницијализација Pairs празним скупом
for each i_b из I_b do
    Генерисање излазног низа B дужине N на основу почетног стања i_b
    Генерисање графа за B и S
    Одређивање скупа свих оптималних путева од Pocetak до Kraj
    Генерисање скупа селектујућих низова A на основу оптималних путева
    for each a из A do
        if селектујући низ је у сагласности са p_1(x) then
            Израчунати почетно стање i_a регистра P1
            Додати (i_a, i_b) у скуп Pairs
        else
            Одбаци претпоставку a
        end if
    end for
end for
return Pairs
```

Рачунање Левенштајновог растојања и матрице растојања доприноси смањењу броја почетних стања регистра $P1$ за фиксирано почетно стање регистра $P2$. Ово побољшање доприноси бољој временској сложености у односу

на напад са потпуном претрагом. Временска сложеност овог алгоритма је $O(2^{L_2} \cdot ((N - M + 1) \cdot M + E + V + P \cdot M + P \cdot N \cdot L_1))$, где је E број грана, а V број чворова у графу, број P представља број оптималних путева дужине $M + 1$. Рачунање Левенштајновог растојања и формирање графа је сложености $O((N - M + 1) \cdot M)$ и има велики утицај на целокупну сложеност у односу на остале операције. С обзиром да сваки пут у графу представља оптималан пут, претрага свих путева од чвора $Pocetak$ до чвора $Kraj$, примењујући алгоритам претраге у дубину (енг. depth first search, *DFS*) као и формирање селектујућих низова, има сложеност $O(E + V + P \cdot M)$. Број оптималних путева P такође може да има велики утицај на целокупну сложеност. Провера да ли је селектујући низ, димензије N, у сагласности са $p_1(x)$ је сложености $O(N \cdot L_1)$. С обзиром да се за сваки селектујући низ проверава сагласност, сложеност је $O(P \cdot N \cdot L_1)$.

Ако се претпостави да је дужина низа кључа $L_1 + L_2$, а $2.4 \cdot (L_1 + L_2)$ дужина излазног низа регистра *P2*. Тада је сложеност тражења Левенштајновог растојања $O((L_1 + L_2)^2)$, стога је укупна временска сложеност напада $O(2^{L_2} \cdot ((L_1 + L_2)^2 + P \cdot (L_1 + L_2) \cdot L_1)$. Број оптималних путева P у најгорем случају, када сваки чвор има највише L_1 излазних грана је L_1^{M+1} и може да има доминантан утицај на целокупну временску сложеност напада.

Предност овог напада, у односу на напад са потпуном претрагом свих варијанти, јесте у томе да се знатно смањује број разматраних почетних стања тактујућег регистра *P1* које треба обрадити и проверити да ли су одговарајућа. У нападу са потпуном претрагом, за свако почетно стање регистра $P2$, потребно је проверити сва могућа почетна стања регистра $P1$. Ово има велику временску сложеност, поготово ако су дужине регистара велике, што је најчешћи случај у пракси. Сложеност напада са потпуном претрагом је $O(2^{L_1+L_2})$. Са растом дужине оба регистра сложеност расте експоненцијално. У нападу базираном на Левенштајновом растојању број почетних стања се знатно смањује итерирањем кроз селектујуће низове добијене из оптималних путева графа.

Мана овог напада је у томе што се не може са сигурношћу тврдити која дужина излазног низа регистра *P2* је довољна да би се добило решење криптоаналитичког напада. Дакле, постоји могућност да због кратке дужине излазног низа регистра *P2* овај напад не пронађе ни једно решење. Ово је последица лоше расподеле јединица и нула у излазном низу регистра *P1*.

Постоји побољшање овог напада које се састоји у једном рачунању Левенштајновог растојања, уместо $2^{L_2} - 1$ рачунања. Како би се обезбедила гаранција да је једно рачунање Левенштајновог растојања довољно да би се пронашао пар почетних стања који генерише низ кључа, потребно је обезбедити довољно дугачак излазни низ регистра *P2*. Неопходно је да излазни низ регистра *P2* садржи свих $2^{L_2} - 1$ почетних стања регистра *P2*. За произвољно почетно стање може се претпоставити да је дужина $2^{L_2} - 1 + 2.4 \cdot (L_1 + L_2)$ излазног низа *P2* довољно дугачка да задовољни претходни услов. На основу овог дугачког излазног низа и низа кључа формира се матрица W, где при рачунању прве колоне не постоји рестрикција броја узастопних брисања. Циљ овог побољшања остаје исти као у основној верзији напада, пронаћи све путеве од *Pocetak* и *Kraj* и генерисати селектујуће низове. Разлика у односу на основну верзију напада је проналазак почетних стања регистра *P2*. У основној верзији ако је селектујући низ у сагласности са $p_1(x)$ тада се за почетно стање узима првих L_2 битова из излазног низа регистра *P2*. У побољшаној верзији ако је прва грана на путу $Pocetak \rightarrow w_{i,1}$, тада је почетно стање регистра *P2* подсеквенца излазног низа *P2* почевши од позиције i до позиције $i + L_2 - 1$. Још једна разлика је то што свака прва грана на путу има тежину 0, односно уписује у прву позицију селектујућег низа бит 1. Остатак селектујућег низа генерише се на исти начин као у основној верзији напада. Како излазни низ дужине $2^{L_2} - 1 + 2.4 \cdot (L_1 + L_2)$ садржи сва почетна стања регистра *P2*, тада се са сигурношћу може тврдити да ће постојати пут од *Pocetak* до *Kraj* на основу ког се добија селектујући низ који обезбеђује валидно почетно стање регистра *P1*.

Ово побољшање има добре теоријске основе. Временска сложеност је $O((N - M + 1) \cdot M + E + V + P \cdot M + P \cdot N \cdot L_1)$. Проблем овог побољшања је у томе што се и за мале дужине регистра *P2* генеришу дуги излазни низови регистра *P2*. Самим тим матрица W је веће димензије, што узрокује да се граф састоји од много више чворова него у основној верзији напада за исте дужине регистара. На основу тога, доминантан утицај на целокупну временску сложеност овог побољшања има сложеност претраге свих оптималних путева у графу. У најгорем случају, ако се претпостави да сваки чвор има највише L_1 грана, тада ће укупан број путања бити L_1^{M+1}, где је M дужина низа кључа. Ако је $M = L_1 + L_2$ тада је укупан број чворова у графу $N \cdot M = (2^{L_2} - 1 + 2.4 \cdot (L_1 + L_2)) \cdot (L_1 + L_2)$.

2.2 Напад на тактовани регистар *P2*

У овом делу приказан је другачији приступ напада на генератор $P1 \rightarrow P2$, где се напада тактовани регистар $P2$, док се за тактујући регистар $P1$ претпостављају сва могућа почетна стања. Напад се базира на тесту линеарне конзистентности система једначина и заснива се на карактеристикама низа кључа $\{s_m\}$, његовог односа са битовима које генеришу *P1* и *P2*, објашњеним у одељку 1.6.

Низ кључа за шифровање који се користи у овом нападу је $\{s_m\} = \{s_0, s_2, \ldots, s_{M-1}\}$. Сваки бит овог низа заправо представља одговарајући бит из излазног низа регистра *P2* $\{b_0, b_1, \ldots\}$. Сваки бит s_i из $\{s_m\}$ може да се представи као b_{o_i}, где o_i представља индекс $(i+1)$ јединице из излазног низа *P1* $\{a_0, a_1, \ldots\}$. Стога, сваки бит низа кључа може да се представи преко линеарне комбинације почетних стања регистра *P2*.

Напад се остварује провером конзистентности и решавањем система линеарних једначина $Q \cdot x = s$ за свако могуће почетно стање регистра *P1*, где је Q бинарна матрица димензије $M \times L_2$. Она је параметризована на основу излазног низа регистра *P1*. Њене врсте су коефицијенти линеарних комбинација који изражавају одговарајуће битове излазног низа регистра P2 преко почетног стања тог регистра $\{b_0, b_1, \ldots, b_{L_2-1}\}$. Вектор $x = \{b_0, b_1, \ldots, b_{L_2-1}\}$ је вектор непознатих димензије $L_2 \times 1$, док је s вектор димензије $M \times 1$ низ кључа за шифровање.

Формирани скуп линеарних једначина може бити без решења или са јединственим решењем. За довољно дугачак низ кључа више решења није могуће. Ако систем нема решења, тада отпада претпоставка о почетном стању *P1*. Ако је систем са јединственим решењем, тада се може одредити почетно стање регистра *P2*, које одговара почетном стању *P1*.

Да би се постигло да напад проналази тачна почетна стање два регистра и потенцијално мали број погрешних, може се узети да је дужина низа кључа за шифровање $L_1 + L_2$. Излазни низ регистра *P1* треба да има бар толико јединица. С обзиром да је овај напад ефикасан и за велике дужине регистра P2, може се претпоставити да је $L_2 > L_1$.

У наставку је приказан псеудокод овог напада, његова имплементација приказана је у одељку 3.2.

Алгоритам 2 Напад на регистар $P2$

Улаз: $p_1(x)$, $p_2(x), S$ дужине M

Излаз: Скуп парова $Pairs = (i_a, i_b)$ почетних стања за регистре $P1$ и $P2$ који генеришу S

1: Иницијализација $Pairs$ празним скупом
2: Генерисање скупа свих могућих почетних стања $P1$, I_a
3: **for each** i_a из I_a **do**
4: Генерисање низа A на основу i_a, таквог да има M јединица
5: Креирање низа индекса јединица у низу A: $\{o_0, ., o_{M-1}\}$
6: Иницијализација $b_{o_k} = s_k$, $k = 0, \ldots, M-1$
7: **for each** b_{o_k} **do**
8: Креирање линеарне једначине $f_k(b_0, b_1, \ldots, b_{L_2-1}) = s_k$
9: **end for**
10: **if** систем је конзистентан **then**
11: **if** (i_a, i_b) генерише S **then** додати (i_a, i_b) у скуп $Pairs$
12: **else**
13: Одбаци претпоставку i_a
14: **end if**
15: **else**
16: Одбаци претпоставку i_a
17: **end if**
18: **end for**
19: **return** Pairs

Свака провера система линеарних једначина, базирана на алгоритму *Гаусове елиминације*, има кубну сложеност у односу на димензију матрице, $O((M \cdot L_2)^3)$. С обзиром да се за свако почетно стање регистра $P1$ проверава конзистентност система линеарних једначина, сложеност напада је $O(2^{L_1} \cdot (M \cdot L_2)^3)$.

Пример 4. *Нека су* $p_1(x) = 1 + x + x^3$, $p_2(x) = 1 + x + x^4$, *почетна стања регистара P1 и P2 редом* $\{1, 0, 0\}$, $\{1, 0, 0, 1\}$. *Излазни низ регистра P1 је* $A = \{1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1 \ldots\}$, *излазни низ регистра P2 је* $B = \{1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1 \ldots\}$. *Тада је низ кључа који они генеришу* $S = \{1, 1, 0, 1, 0, 1, 0, 0, 0 \ldots\}$.

Нека је део низа кључа за шифровање $\{s_m\} = \{1, 1, 0, 1, 0, 1, 0\}$ *дужине* $L_1 + L_2$. *Претпостављено почетно стање P1 је* $\{1, 0, 0\}$. *Индекси првих* $L_1 + L_2$ *јединица у низу A су:* $0, 3, 5, 6, 7, 10, 12$. *Тада је* $b_0 = s_0$, $b_3 = s_1$, $b_5 = s_2$, $b_6 = s_3$, $b_7 = s_4$, $b_{10} = s_5$, $b_{12} = s_6$. *Како се сваки бит из низа B може представити као линеарна комбинација почетних стања на основу*

$p_2(x)$, *добијају се линеарне једначине:*

- $b_0 = b_0$,
- $b_3 = b_3$,
- $b_5 = b_1 + b_2$,
- $b_6 = b_2 + b_3$,
- $b_7 = b_3 + b_4 = b_3 + b_0 + b_1$,
- $b_{10} = b_6 + b_7 = b_2 + b_3 + b_3 + b_0 + b_1 = b_0 + b_1 + b_2$,
- $b_{12} = b_8 + b_9 = b_0 + b_2 + b_1 + b_3$.

Матрица Q *је облика:*

$$Q = \begin{Bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{Bmatrix}$$

Решење система $Q \cdot x = s$ *је* $b_0 = 1, b_1 = 0, b_2 = 0, b_3 = 1$. *Ово решење система је почетно стање регистра P2, на основу претпостављеног почетног стања регистра P1.*

Глава 3

Имплементације напада

У наставку су приказане имплементације напада описаних у одељку 2. Имплементације су писане у програмском језику *C#*, окружење *.NET 7*. Такође, приказани су резултати мерења времена извршавања напада за различите улазе. За мерење времена извршавања програма за различите улазе коришћена је класа *Stopwatch*. У истраживању су примитивни полиноми бирани на основу табеле бинарних примитивних полинома степена n са $k \in \{3, 5, 7\}$ чланова [4].

Истраживање је вршено на рачунару са следећим спецификацијама:

- *Оперативни систем*: Windows 11,
- *CPU*: Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz,
- *RAM*: 16.0 GB,
- *Трајна меморија*: тип SSD, величина 512 GB.

3.1 Имплементација напада на тактујући регистар *P1*

Приказана је имплементација напада на тактујући регистар $P1$, који је описан у одељку 2.1. Овај напад се базира на рачунању Левенштајновог растојања, односно матрице растојања W и тражењу свих оптималних путева у придруженом графу.

За проналазак свих оптималних путева у графу, тачније свих путева од чвора *Pocetak* до чвора $Kraj$ коришћен је алгоритам *DFS* (енг. depth-first

search). Ради уштеде времена, сваки пут када се рачуна елемент матрице W, додају се чворови у граф и чувају се њихова растојања од чвора $Pocetak$. Сваки чвор има листу претходних чворова и листу грана. *DFS* је имплементиран тако да полази од чвора $Kraj$ до чвора $Pocetak$, рекурзивно се позива на основу листе претходних чворова коју има сваки чвор.

Метода која имплементира напад је *AttackR1*. Улаз методе је:

- *s*: `int[]` – низ кључа за шифровање, низ битова,
- *polyL1*: `int[]` – карактеристични полином регистра P1,
- *polyL2*: `int[]` – карактеристични полином регистра P2.

Излаз методе је:

- *initStates:* `List<(int[], int[])>` – парови почетних стања, свако почетно стање је низ битова.

Карактеристични полиноми су представљени у облику низа битова коефицијената полинома. На пример, $p(x) = 1 + x + x^4$ је низ $\{1, 1, 0, 0, 1\}$. Имплементација методе *AttackR1* дата је у секцији *Прилоī 1.*

У наставку је приказан пример покретања програма, за генератор $P1 \rightarrow P2$ заснован на тактујућем регистру дужине 9 и тактованом регистру дужине 4. Нека су карактеристични полиноми, $p_1(x) = 1 + x^4 + x^9$, $p_2(x) = 1 + x + x^4$. Нека је почетно стање регистра *P1* $\{0, 1, 1, 1, 0, 0, 1, 0, 1\}$, почетно стање регистра *P2* $\{1, 0, 1, 0\}$. Тада је:

- излазни низ регистра *P1*: $A = \{0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0 \dots\}$,
- излазни низ регистра *P2*: $B = \{1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1 \dots\}$,
- низ кључа $S = \{0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0 \dots\}$.

Улаз методе *AttackR1* је део низ кључа $\{s_m\} = \{0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0\}$ дужине $L_1 + L_2$, као и познате карактеристике генератора $P1 \rightarrow P2$. За дужину одсечка излазног низа регистра *P2* узето је $2.4 \cdot (L_1 + L_2)$.

Позив методе *AttackR1*:

```
static void Main() {
    // niz kljuca
    int[] s = new int[] { 0,1,0,1,0,0,0,0,0,1,1,1,0 };
    // karakteristican polinom registra R1 p1(x) = 1+x^4+x^9
    int[] polyL1 = new int[] { 1, 0, 0, 0, 1, 0, 0, 0, 0, 1 };
    // karakteristican polinom registra R2 p2(x) = 1 + x+x^4
    int[] polyL2 = new int[] { 1, 1, 0, 0, 1 };

    // poziv metode koja napada registar R1, izlaz ove metode su
        parovi pocetnih stanja oba registra, koji generisu niz
       kljuca
    List<(int[], int[])> initStates = AttackR1(s, polyL1, polyL2
       );
    Console.WriteLine("Number␣of␣pairs␣init␣states:" +
       initStates.Count );
}
```

Метода у овом примеру враћа четири пара почетних стања регистара, који генеришу дати низ кључа за шифровање:

- $\{1,1,1,0,0,1,0,1,1\},\{0,1,0,1\}$,
- $\{1,1,1,0,0,1,0,1,0\},\{0,1,0,1\}$,
- $\{0,1,1,1,0,0,1,0,1\},\{1,0,1,0\}$,
- $\{0,0,1,1,1,0,0,1,0\},\{1,1,0,1\}$.

Сви ови парови почетних стања су валидна решења, јер сви они генеришу исти низ кључа дужине $L_1 + L_2$. Један од ових парова заиста одговара почетним стањима изабраним на почетку примера, пар ({0,1,1,1,0,0,1,0,1}, {1,0,1,0}). Метода *AttackR1* је испунила свој задатак, вратила је парове почетних стања који генеришу дати низ кључа одређене дужине. У овом примеру, ако би се дужина низа кључа повећала бар за један, тада се скуп почетних стања које метода $AttackR1$ враћа смањује. Из тако добијеног скупа почетних стања, недостајао би први пар $(\{1,1,1,0,0,1,0,1,1\},\{0,1,0,1\})$, зато што овај пар генерише низ кључа $\{0,1,0,1,0,0,0,0,0,1,1,1,0,1,0,0,1,\ldots\}$. Првих $L_1 + L_2$ битова се поклапа са низом $\{s_m\}$, али већ на следећем биту се разликује. Да би се смањио скуп почетних стања које генерише програм, може се претпоставити да је одсечак низа кључа за шифровање веће дужине.

Дужина низа кључа за шифровање M може бити произвољна. Дужина N излазног низа регистра *P2*, чије се варијанте испробавају, одређује се у зависности од M тако да је $M \leq N \leq (M-1) \cdot L_1 + 1$. Што је већа дужина низа кључа, број кандидата за решење се смањује. Међутим, ако је дужина низа кључа велика, то повлачи да је дужина излазног низа *P2* велика, што успорава рад програма. Једнозначно решење се очекује ако је $M \geq L_1 + L_2$.

Ради уравнотежености излазних низова регистара, у просеку се брише сваки други бит из излазног низа регистра *P2*, стога се за његову дужину узима $2.4 \cdot M$. Истраживање је вршено са претпоставком да је низ кључа дужине $M = L_1 + L_2$. За дужину N излазног низа регистра *P2* узима се дужина $2.4 \cdot (L_1 + L_2)$. Пошто се у овом нападу пролази кроз сва почетна стања регистра *P2*, како би се демонстрирала снага напада, у тестним примерима су биране дужине регистара тако да је $L_2 \leq L_1$, док се за дужину регистра *P1* узима вредност $L_1 = 2 \cdot L_2 + 1$ и $L_1 = 3 \cdot L_2 + 1$.

Резултати добијени овим експериментима су наведени у табели 3.1. Дијаграм односа зависности логаритма (за основу 10) потребног времена за израчунавање од различитих дужина регистара, заједно са линеарном функцијом $L_1 + L_2 - c$, приказан је на слици 3.1. Дијаграм је приказан само за резултате добијене када је $L_1 = 2 \cdot L_2 + 1$. Константа c је изабрана тако да права буде отприлике пола тачака испод, а пола изнад праве $AttackR1$. У овом приказаном истраживању c је 13.

L_1	L_2	Време
5	2	0.036(s)
7	3	0.120(s)
9	4	1.479(s)
11	5	26.823(s)
13	6	605.743(s)

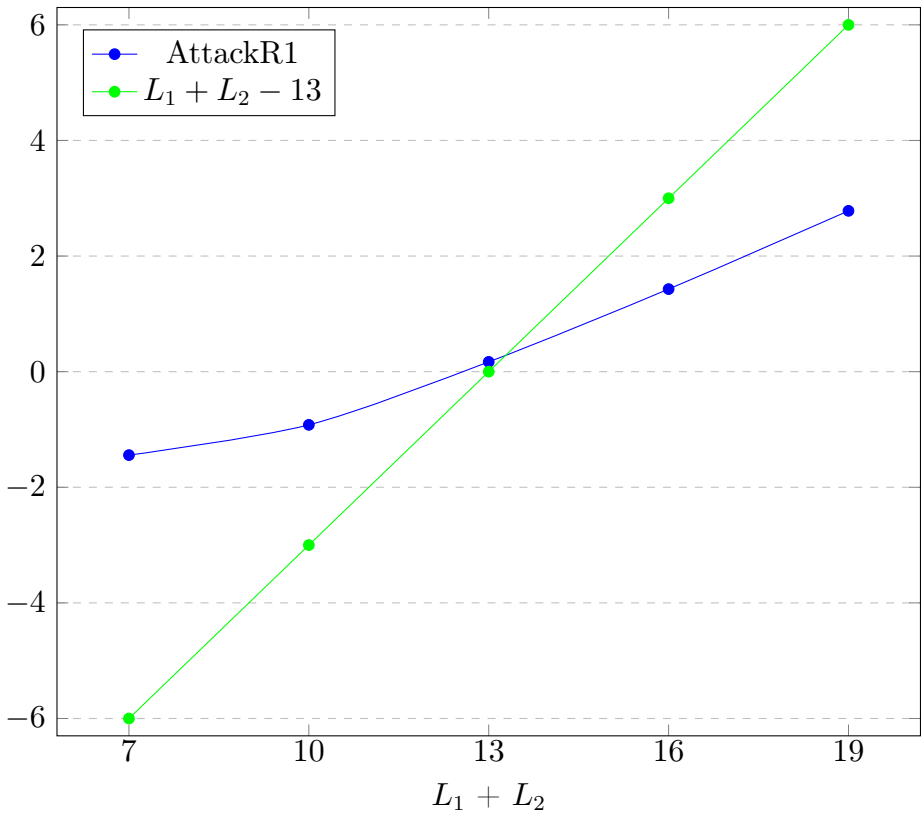
$L_1 = 2 \cdot L_2 + 1$

L_1	L_2	Време
7	2	0.009(s)
10	3	1.166(s)
13	4	40.646(s)
16	5	5508.108(s)

$L_1 = 3 \cdot L_2 + 1$

Табела 3.1: Време извршавања за различите дужине регистара

Са дијаграма се може уочити експоненцијални раст функције извршавања програма у односу на раст дужина регистра *P2*, на који приметно утичу фактори израчунавања Левенштајновог растојања и претраге кроз граф, што је у складу са просечном сложеношћу алгоритма приказаном у одељку 2.1. Време извршавања програма је мало за мале дужине регистра *P2*, док за веће дужине програм не показује добре перформансе. Дужина регистра *P1* може



Слика 3.1: Приказ логаритма времена извршавања напада на тактујући регистар у зависности од улаза величине регистара када је $L_1 = 2 \cdot L_2 + 1$

бити већа. Ако је дужина регистра *P1* три пута већа од дужине регистра *P2*, примећује се да програм није показао добре перформансе. Тада је дужина излазног низа регистра *P2* велика јер се посматра дужина $2.4 \cdot (L_1 + L_2)$, па самим тим број чворова у графу расте и претрага оптималних путева се дуже извршава.

Закључак који може да се изведе на основу ових истраживања јесте да дужина излазног низа регистра *P2* утиче на успешност овог криптоаналитичког напада, али са друге стране може знатно успорити извршавање програма. Ако је дужина велика, вероватноћа успешности напада расте, али брзина извршавања програма се смањује. Може се узети и краћа дужина регистра *P1*, на пример $2 \cdot L_2 + 1$. Програм се тада брже извршава, али се успешност проналаска почетних стања смањује, међутим то не искључује да програм неће наћи ваљана почетна стања регистара. Фактори који утичу на успешност програма су дужина излазног низа регистра *P2*, као и добра расподела јединица и нула у излазним низовима регистара.

3.2 Имплементација напада на тактовани регистар P2

У овом одељку приказана је имплементација напада на тактовани регистар $P2$, који је описан у одељку 2.2.

За проверу конзистентности система линеарних једначина и добијање решења, коришћен је алгоритам Гаусове елиминације, видети пример 3.2.

Свака матрица Q проширује се додатном колоном вредностима из вектора s. Тако добијена проширена матрица $[Q|s]$ се дијагонализује. Провера конзистентности система је заправо провера да ли у проширеној дијагонализованој матрици постоји ред облика $[0, \ldots, 0|1]$, тада је $Rang(Q) < Rang([Q|s])$, систем није конзистентан и нема решење. У супротном, систем је конзистентан када је $Rang(Q) = Rang([Q|s])$. У том случају постоји једно или више решења. За $Rang(Q) = Rang([Q|s]) = L_2$ постоји јединствено решење, то је последња колона проширене дијагонализоване матрице. С обзиром на претпоставку да постоји довољан број једначина, односно довољно дугачак низ кључа, решење ће у већини случајева бити јединствено. У случају да је $Rang(Q) = Rang([Q|s]) < L_2$ тада се за непознате битове у почетном стању регистра *P2* узимају комбинације битова 1 и 0.

Пример 5. *Нека су матрица Q и низ кључа за шифровање s из примера 4. Проширена матрица Q са s приказана је на слици 3.2a. Гаусовом елиминацијом дијагонализује се проширена матрица. Проласком кроз врсте матрице, проверава се да ли у врсти постоји вредност* 1 *на некој позицији, ако постоји узима се прво појављивање вредности* 1 *за пивот (Слика 3.2a). Тако одабрана врста сабира се са свим врстама које на тој позицији имају вредност* 1 *и добијају се вредности* 0 *изнад и испод пивота (Слика 3.2b). Поступак се понавља (Слика 3.2c, 3.2d и 3.2e). Када више нема кандидата за пивот, замењују се позиције врста како би се добила дијагонална матрица Q. У проширеној колони, ако је систем конзистентан, налазе се решење овог система, односно почетно стање за регистар P2 (Слика 3.2f).*

Метода која имплементира напад је *AttackR2*. Улаз методе је:

- *s*: `int[]` – низ кључа за шифровање, низ битова,
- *polyL1*: `int[]` – карактеристични полином регистра P1,

$$\left|\begin{array}{cccc|c} \mathbf{1} & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{array}\right| \qquad \left|\begin{array}{cccc|c} 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & \mathbf{1} & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \end{array}\right| \qquad \left|\begin{array}{cccc|c} 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & \mathbf{1} & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{array}\right|$$

(a) Матрица $[Q|s]$ (b) (c)

$$\left|\begin{array}{cccc|c} 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right| \qquad \left|\begin{array}{cccc|c} 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right| \qquad \left|\begin{array}{cccc|c} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right|$$

(d) (e) (f)

Слика 3.2: Гаусова елиминација

- *polyL2*: `int[]` – карактеристични полином регистра P2.

Излаз методе *AttackR2* је:

- *initStates:* `List<(int[], int[])>` – парови почетних стања, свако почетно стање је низ битова.

Карактеристични полином је низ битова, представљен исто као у методи *AttackR1* приказаној у одељку 3.1. Имплементација методе *AttackR2*, дата је у секцији *Прилоī 2*.

Нека генератор $P1 \to P2$ има тактујући регистар дужине 5 и тактовани регистар дужине 11. Карактеристични полиноми су $p_1(x) = 1+x+x^2+x^3+x^5$, $p_2(x) = 1+x+x^8+x^{10}+x^{11}$. Нека су почетна стања регистара *P1* и *P2* редом, $\{1,0,0,1,1\}$ и $\{1,1,0,0,0,1,1,1,0,0,1\}$. Тада су излазни низ *P1*, излазни низ *P2* и низ кључа који генеришу, редом:

- $A = \{1,0,0,1,1,0,0,0,0,1,0,1,1,0,1,0,1,0,0,0,1,1,1,0,1,1,1,1,1,0,\dots\}$,
- $B = \{1,1,0,0,0,1,1,1,0,0,1,1,0,1,0,1,0,0,0,0,1,1,0,0,0,1,0,0,1,1,\dots\}$,
- $S = \{1,0,0,0,1,0,0,0,1,1,0,0,1,0,0,1,1,0,0,0,1,1,1,0,1,0,1,0,0,1,\dots\}$.

Нека је низ кључа одређен на основу шифрата и одговарајућег текста $\{s_m\} = \{1,0,0,0,1,0,0,0,1,1,0,0,1,0,0,1\}$, дужине $M = L_1 + L_2$. У наставку је приказан позив методе *AttackR2* за дати низ кључа и одговарајуће карактеристичне полиноме.

```
static void Main() {
   // niz kljuca
    int[] s =  {1,0,0,0,1,0,0,0,1,1,0,0,1,0,0,1};
    // karakteristican polinom registra R1 p1(x) = 1 + x + x^2 +
        x^3+ x^5
    int[] polyL1 = new int[] { 1, 1, 1, 1, 0, 1 };
    // karakteristican polinom registra P2 p2(x) = 1 + x+ x^8+ x
       ^10 + x^11
    int[] polyL2 = new int[] { 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       1 };

    // poziv metode koja napada registar R2, izlaz ove metode su
        parovi pocetnih stanja oba registra koji generisu niz
       kljuca
    List<(int[], int[]) > initStates = AttackR2(s, polyL1,
       polyL2);
    Console.WriteLine("Number␣of␣pairs␣init␣states:␣" +
       initStates.Count);
}
```

Метода *AttackR2* у овом примеру враћа седам парова почетних стања регистара:

- $(\{0,0,0,1,1\}, \{0,1,0,1,0,0,1,0,1,0,0\})$,
- $(\{0,0,1,0,0\}, \{0,1,1,1,0,0,0,1,1,1,0\})$,
- $(\{0,0,1,1,1\}, \{1,0,1,0,0,1,0,1,0,0,0\})$,
- $(\{0,1,0,0,1\}, \{1,1,1,0,0,0,1,1,1,0,0\})$,
- $(\{0,1,1,1,0\}, \{0,1,0,0,1,0,1,0,0,0,1\})$,
- $(\{1,0,0,1,1\}, \{1,1,0,0,0,1,1,1,0,0,1\})$,
- $(\{1,1,1,0,1\}, \{1,0,0,1,0,1,0,0,0,1,0\})$.

Приметимо да је шести пар у овом скупу пар почетних стања који су заиста почетна стања регистара *P1* и *P2*. Сви ови парови генеришу исти низ кључа до дужине $L_1 + L_2$. Ако би се дужина низа кључа повећала, бар за дупло $M = 2 \cdot (L_1 + L_2)$, овај скуп би се смањио на четири потенцијална почетна стања. Стања из тренутног решења која би тада отпала, генеришу следеће низове кључа дужине $M = 2 \cdot (L_1 + L_2)$:

- први пар почетних стања генерише низ кључа: $\{1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1\}$,
- трећи пар почетних стања генерише низ кључа: $\{1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1\}$,
- пети пар почетних стања генерише низ кључа: $\{1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1\}$.

Дакле, метода *AttackR2* је успешно извршила свој задатак. Пронађени су тачни парови почетних стања који генеришу исти низ кључа задате дужине. Што је већа дужина низа кључа за шифровање, број кандидата у нападу се смањује.

Мерење брзине извршавања програма је вршено на полиномима различитих дужина. Претпоставка је да је низ кључа за шифровање дужине $L_1 + L_2$. С обзиром да овај напад пролази кроз сва могућа почетна стања регистра *P1*, како би се показала снага напада, за дужину регистра *P2* су биране вредности $2 \cdot L_1 + 1$ и $3 \cdot L_1 + 1$.

Резултати који су добијени истраживањем за различите дужине регистара су наведени у табели 3.2. Однос зависности логаритма (за основу 10) потребног времена за израчунавање од различитих дужина регистара, заједно са линеарном фукцијом $L_1 + L_2 - c$ приказани су на дијаграмима 3.3 и 3.4. Константа c се бира на исти начин као у претходном одељку. У овом приказаном истраживању c је 44 када је $2 \cdot L_1 + 1$, односно c је 60 када је $3 \cdot L_1 + 1$.

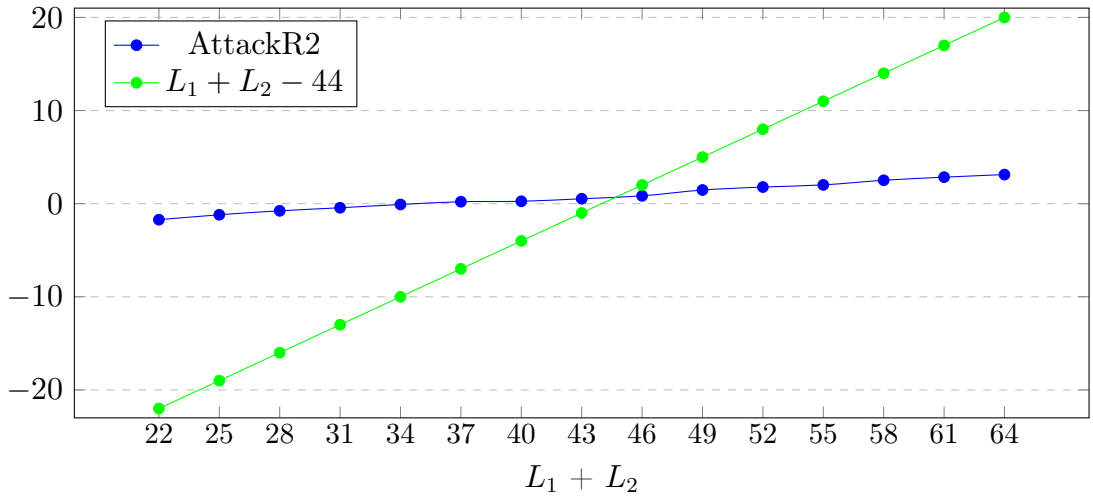
L_1	L_2	Време(s)
7	15	0.02
8	17	0.06
9	19	0.17
10	21	0.36
11	23	0.82
12	25	1.62
13	27	1.80
14	29	3.39
15	31	6.94
16	33	29.37
17	35	61.15
18	37	103.87
19	39	326.19
20	41	703.71
21	43	1331.80

$L_2 = 2 \cdot L_1 + 1$

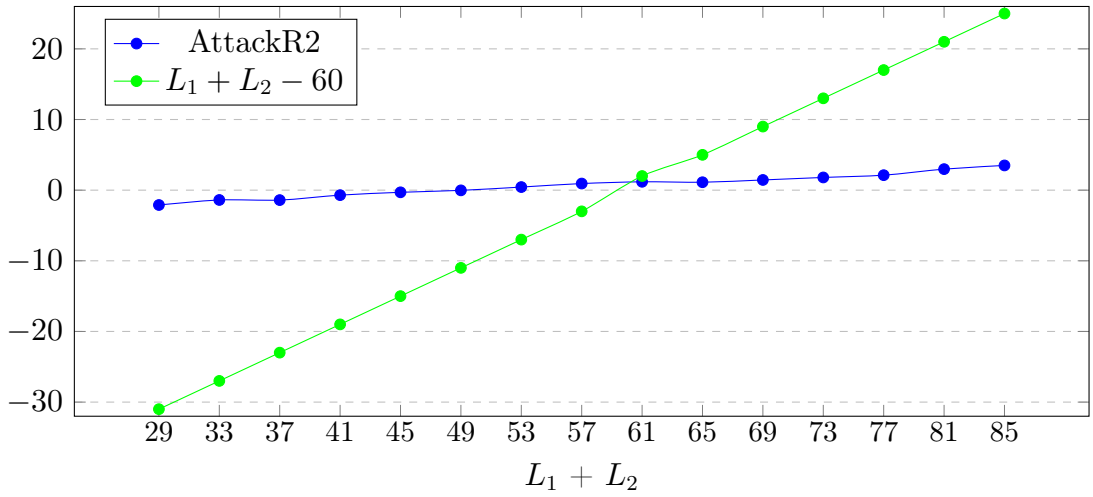
L_1	L_2	Време(s)
7	22	0.01
8	25	0.04
9	28	0.05
10	31	0.21
11	34	0.50
12	37	0.93
13	40	2.74
14	43	8.61
15	46	13.30
16	49	15.22
17	52	27.83
18	55	62.87
19	58	131.97
20	61	949.14
21	64	3157.99

$L_2 = 3 \cdot L_1 + 1$

Табела 3.2: Време извршавања за различите дужине регистара



Слика 3.3: Приказ логаритма времена извршавања напада на тактовани регистар у зависности од улаза величине регистара када је $L_2 = 2 \cdot L_1 + 1$



Слика 3.4: Приказ логаритма времена извршавања напада на тактовани регистар у зависности од улаза величине регистара када је $L_2 = 3 \cdot L_1 + 1$

На основу дијаграма примећује се да је логаритам функције времена извршавања програма *AttackR2* приближно права линија. Дакле, фунција експоненцијално расте са порастом дужина регистара. Може се закључити да су приказани резултати у сагласности са теоријском проценом напада датом у одељку 2.2. Такође, ако се посматрају криве $L_1 + L_2$ и $AttackR2$, може се закључити да је сложеност овог напада мања од сложености напада са потпуном претрагом. На основу спроведеног истраживања напад је ефикасан за мале дужине регистра *P1*, док дужине регистра *P2* могу бити много веће. Ако се посматрају брзине извршавања програма приказаним у табелама 3.2, може се закључити да повећање дужине регистра *P2* утиче на брзину извршавања поготово када су обе дужине регистара велике. То се јасно може приметити ако се посматрају парови $(L_1, L_2) = (21, 43)$ и $(L_1, L_2) = (21, 64)$.

Глава 4

Закључак

У овом раду приказана су два детерминистичка напада са познатим отвореним текстом на генератор *P1* $\rightarrow$ *P2*. Овај генератор се састоји од два регистра *P1* и *P2*, при чему регистар *P1* тактује регистар *P2*. Приказани напади су међусобно неупоредиви, зато што један напада тактујући регистар *P1*, а други тактовани регистар *P2*, пролазећи кроз сва могућа почетна стања другог регистра, ослањајући се на различите концепте при нападу.

Први напад користи Левенштајново растојање са специфичним ограничењем, да је дозвољено само брисање. Рачунање упрошћеног Левенштајновог растојања доприноси смањењу броја могућих почетних стања регистра $P1$ за фиксирано почетно стање регистра $P2$, па самим тим доприноси бољој временској сложености у односу на напад са потпуном претрагом. На основу анализе закључује се да је могуће извести успешан напад на генератор *P1* $\rightarrow$ *P2*, под претпоставком да је дужина регистра *P2* релативно мала, док дужина регистра *P1* може бити много већа.

Други напад се заснива на тесту линеарне конзистентности система линеарних једначина. Овај приступ захтева детаљније разумевање структуре генератора *P1* $\rightarrow$ *P2*, односа излазних битова регистара и низа кључа за шифровање. Овај напад је успешан када генератор *P1* $\rightarrow$ *P2* има регистар *P1* чија је дужина мала, док дужина регистра *P2* може бити већа. Напад се показао нешто бољим у погледу распона дужина регистра *P1* за које је реализовани напад ефикасан.

Могући правац даљег рада за први напад је реализација ефикаснијег алгоритма који се поред рачунања Левенштајновог растојања и дефинисања графа, заснива и на скуповима сечења који су корисни за дефинисање скупа

услова да би се смањио број почетних стања регистра *P2* [6]. За други напад, могући правац даљег рада је реализација ефикаснијег алгоритма који се базира на тесту линеарне конзистентности, где се уместо узастопних битова низа кључа користе укрштајући низови [9].

Библиографија

[1] A.J.Menezes, P.C.van Oorschot, and S.A.Vanstone. Handbook of applied cryptography. *CRC Press*, 2020.

[2] D.Coppersmith, H.Krawczyk, and Y.Mansour. The shrinking generator. In *Annual International Cryptology Conference*, pages 22–39. Springer, 1993.

[3] F.Masoodi, S.Alam, and M.U.Bokhari. An analysis of linear feedback shift registers in stream ciphers. *International Journal of Computer Applications*, 46:46–49, 2012.

[4] M.Zivkovic. A table of primitive binary polynomials. *Mathematics of Computation*, 62:385–386, 1994.

[5] M.Zivkovic. Algoritmi. *Matematicki fakultet, Beograd*, 2000.

[6] P.Caballero-Gil and A.Fúster-Sabater. A simple attack on some clock-controlled generators. *Computers & Mathematics with Applications*, 58:179–188, 2009.

[7] P.Caballero-Gil, A.Fúster-Sabater, and C.Hernandez-Goya. Graph-based approach to the edit distance cryptanalysis of irregularly clocked linear feedback shift registers. *Journal of Universal Computer Science*, 15:2981–2998, 2009.

[8] P.Caballero-Gil, A.Fúster-Sabater, and M.E.Pazo-Robles. New attack strategy for the shrinking generator. *Journal of Research and Practice in Information Technology*, 41:181–190, 2009.

[9] S.D.Cardell and A.Fúster-Sabater. *Cryptography with shrinking generators: fundamentals and applications of Keystream sequence generators based on irregular decimation.* Springer, 2019.

[10] W.Meier and O.Staffelbach. The self-shrinking generator. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 205–214. Springer, 1994.

Прилог

У овој секцији су приказани програми који доприносе практичној примени теоријских концепата напада приказаних у одељку 2. Програми су писани у $C\#$, окружење $.NET$ 7.

Прилог 1

Дата је програмска реализација алгоритма описаног у одељку 2.1. Пример покретања програма је у одељку 3.1.

```
// metoda AttackR1 koja vrsi napad na taktujuci registar R1
static List<(int[], int[])> AttackR1(int[] s, int[] polyL1, int
    [] polyL2)
{
    // duzina R1
    int L_r1 = polyL1.Length - 1;
    // period R1
    uint T_r1 = (uint)Math.Pow(2, L_r1) - 1;
    // duzina R2
    int L_r2 = polyL2.Length - 1;
    // period R2
    uint T_r2 = (uint)Math.Pow(2, L_r2) - 1;
    // duzina niya kljuca
    int lengthS = s.Length;
    // fiksiramo duzinu izlaznog niza bitova registra R2
    int lengthB = (int)((L_r1 + L_r2) * 2.4);

    // promenljiva koja cuva parove pocetnih stanja koji su
       resenja
    List<(int[], int[])> initStates = new();
```

```
for (uint i = 1; i < (uint)1 << polyL2.Length - 1; i++)
{
    string binaryString = Convert.ToString(i, 2).PadLeft(
        L_r2, '0');
    // pocetno stanje za R2
    int[] initStateR2 = binaryString.Select(c => int.Parse(c
        .ToString())).ToArray();
    // izlazni niz bitova registra R2 za odredjeno pocetno
        stanje
    int[] outputR2 = LFSR(polyL2, initStateR2, lengthB);

    // racunamo rastojanje niza bitova registra R2 i niza
        kljuca
    // racunanjem rastojanja, ova metoda ce kreirati i
        popuniti graph sa svim cvorovima koji postoje i
        tezinskim granama
    // ako postiji rastojanje, metoda ce vratiti graf, a ako
         ne postoji vraca null
    Graph? graph = LevenshteinDistance(s, outputR2, L_r1);

    if (graph is not null)
    {
        graph.FindPaths($"V_{lengthB␣-␣lengthS}_{lengthS␣+␣
            1}", "V_0_0", $"V_{lengthB␣-␣lengthS}_{lengthS␣+␣
            1}", new Stack<string>(), lengthB, polyL1);

        foreach (int[] selection in graph.SelectionSequences
            )
        {
            int[] initStateR1 = selection.Take(L_r1).ToArray
                ();
            // za svaki selektujuci niz, proverava se da li
                generise isti niz kljuca sa pocetnim stanjem
                initStateR2
            if (CheckGenerator(initStateR1, initStateR2,
                polyL1, polyL2, s))
                initStates.Add((initStateR1, initStateR2));
```

```
            }
        }
    }
    return initStates;
}

// Metoda koja proverava da li pocetna stanja initStateR1, i
   initStateR2, generisu niz kljuca s
static bool CheckGenerator(int[] initStateR1, int[] initStateR2,
    int[] polyL1, int[] polyL2, int[] s)
{
    int[] initR1 = new int[polyL1.Length - 1];
    int[] initR2 = new int[polyL2.Length - 1];
    Array.Copy(initStateR1, initR1, polyL1.Length - 1);
    Array.Copy(initStateR2, initR2, polyL2.Length - 1);

    int count = 0;
    while (count < s.Length)
    {
        (int, int) outputBits = Clock(initR1, initR2, polyL1,
           polyL2);

        if (outputBits.Item1 == 1)
        {
            if (outputBits.Item2 != s[count++])
                return false;
        }
    }

    return true;
}
// Metoda koja imitira taktovanje registara
static (int, int) Clock(int[] initR1, int[] initR2, int[] polyL1
   , int[] polyL2)
{
    (int, int) outputBits = new();
    int newBitR1 = 0;
```

```
    for (int j = 0; j < polyL1.Length - 1; j++)
    {
        newBitR1 = newBitR1 ^ (initR1[j] & polyL1[j]);
    }

    outputBits.Item1 = initR1[0];
    Array.Copy(initR1, 1, initR1, 0, initR1.Length - 1);
    initR1[initR1.Length - 1] = newBitR1;

    int newBitR2 = 0;
    for (int j = 0; j < polyL2.Length - 1; j++)
    {
        newBitR2 = newBitR2 ^ (initR2[j] & polyL2[j]);
    }

    outputBits.Item2 = initR2[0];
    Array.Copy(initR2, 1, initR2, 0, initR2.Length - 1);
    initR2[initR2.Length - 1] = newBitR2;

    return outputBits;
}

// Metoda generise izlazni niz bitova, duzine length, na osnovu
   pocetnog stanja state, za dati karakteristicni polinom poly
// Ulaz: polinom, pocetno stanje, duzina izlaznog niza bitova
// Izlaz: izlazni niz bitova
static int[] LFSR(int[] poly, int[] state, int length)
{
    int[] outputSequence = new int[length];
    int[] tmpState = new int[state.Length];
    Array.Copy(state, tmpState, state.Length);
    for (int i = 0; i < length; i++)
    {
        int newBit = 0;
        for (int j = 0; j < poly.Length - 1; j++)
        {
            newBit = newBit ^ (tmpState[j] & poly[j]);
```

```
        }

        outputSequence[i] = tmpState[0];
        Array.Copy(tmpState, 1, tmpState, 0, tmpState.Length -
           1);
        tmpState[tmpState.Length - 1] = newBit;
    }
    return outputSequence;
}


// Metoda provera da li je selektujuci niz u saglasnosti sa
   karakteristicnim polinomom p1(x)
static bool CheckSeqWithLFSR(int[] seq, int[] poly)
{
    for (int i = poly.Length - 1; i < seq.Length; i++)
    {
        int bit = 0;
        int k = i - (poly.Length - 1);
        for (int j = 0; j < poly.Length - 1; j++)
        {
            bit ^= (seq[k++] & poly[j]);
        }
        if (seq[k] == bit)
            continue;
        else
        {
            return false;
        }
    }
    return true;
}


// Metoda racunanjem matrice rastojanja, kreira graf, sa
   cvorovima iz matrice rastojanja, i granama cije tezine se
   dobijaju na osnovu metode Pk
```

```
// Cvorovi se kreiraju tako da za svaki element matrice[i,j],
   cvor oznacimo kao "V_{i}{j+1}"
// Cvor "Pocetak" oznacavamo sa "V_{0}{0}", cvor "Kraj"
   oznacavamo sa "V_{matrixN - 1}{matrixM + 1}"
// Ulaz: y - niz kljuca, x - izlazni niz bitova R2, L_1 -
   maksimalan broj uzastopnih brisanja
// Izlaz: graph - graf kreiran na osnovu matrice rastojanja, ili
    null - ako rastojanje ne postoji
static Graph? LevenshteinDistance(int[] y, int[] x, int L_1)
{
    Graph graph = new();

    int matrixN = x.Length - y.Length + 1;
    int matrixM = y.Length;
    int[,] matrix = new int[matrixN, matrixM];

    // inicijalizacija svakog elementa matrice na maksimalnu
       vrednost
    for (int i = 0; i < matrixN; i++)
        for (int j = 0; j < matrixM; j++)
            matrix[i, j] = int.MaxValue;

    // dodavanje cvora "Pocetak" u graf
    graph.AddNode("V_0_0");

    // racunanje elemenata matrice prve kolone
    for (int i = 0; i < (L_1 < matrixN ? L_1 : matrixN); i++)
    {
        int pk = Pk(i, x[i], y[0]);
        if (pk is not int.MaxValue)
        {
            matrix[i, 0] = pk;
            graph.AddEdge("V_0_0", $"V_{i}_1", matrix[i, 0]);

        }
    }
```

```
// racunanje redom svaku narednu kolonu matrice
for (int j = 1; j < matrixM; j++)
{
    for (int i = 0; i < matrixN; i++)
    {
        // posebna racunica ako element pripada prvom redu
        if (i == 0)
        {
            int pk = Pk(0, x[j], y[j]);
            if (pk is not int.MaxValue && matrix[0, j - 1]
                is not int.MaxValue)
            {
                matrix[0, j] = matrix[0, j - 1] + Pk(0, x[j
                    ], y[j]);
                graph.AddEdge($"V_0_{j}", $"V_0_{j␣+␣1}", pk
                    );
            }

            continue;
        }

        // za svaki element racunamo minimum od L_1
            elemenata iz prethodne kolone
        int minimum = int.MaxValue;
        for (int k = 0; k <= (L_1 - 1 < i ? L_1 - 1 : i); k
            ++)
        {
            if (matrix[i - k, j - 1] is int.MaxValue)
                continue;

            int pk = Pk(k, x[i + j], y[j]);
            int tmpMin = matrix[i - k, j - 1] + pk;

            if (pk is not int.MaxValue)
            {
                graph.AddEdge($"V_{i␣-␣k}_{j}", $"V_{i}_{j␣+
                    ␣1}", pk);
```

```
                    minimum = tmpMin < minimum ? tmpMin :
                        minimum;
                }
            }
            matrix[i, j] = minimum;
        }
    }

    // dodavanje cvora "Kraj" u graf
    graph.AddNode($"V_{matrixN␣-␣1}_{matrixM␣+␣1}");

    // dodavanje grana izmedju cvorova poslednje kolone i cvora
       "Kraj"
    for (int i = 0; i < matrixN; i++)
    {
        int previousVertex = matrix[i, matrixM - 1];
        if (previousVertex is not int.MaxValue)
        {
            int pk = Pk(matrixN - 1 - i, x[i + matrixM - 1], y[
               matrixM - 1]);
            graph.AddEdge($"V_{i}_{matrixM}", $"V_{matrixN␣-␣1}_
               {matrixM␣+␣1}", pk);
        }
    }
    if (graph.Nodes[$"V_{matrixN␣-␣1}_{matrixM␣+␣1}"].Previous.
       Count() is 0)
        return null;

    return graph;
}

// Metoda koja vraca cenu brisanja k prethodnih bitova ako su x
   i y jednaki
static int Pk(int k, int x, int y)
{
    return x == y ? k : int.MaxValue;
}
```

```
// Klasa koja predstavlja cvor u grafu
// Id - oblika V_{i}{j+1} za svaki element iz matrice d[i,j];
// Edges - sve grane koje izlaze iz tog cvora, grane su oblika (
   Node, tezina_grane)
// Previous - lista cvorova koji imaju direktnu granu ka tom
   cvoru
public class Node
{
    public string Id { get; set; }
    public List<string> Previous { get; set; } = new();
    public Dictionary<string, int> Edges { get; set; } = new
       Dictionary<string, int>();
}

// Klasa koja predstavlja indukovani graf
// SelectionSequences - lista svih selektujucih nizova dobijenih
    na osnovu osnovnog grafa
// Nodes - mapa svih cvorova u grafu
public class Graph
{
    public List<int[]> SelectionSequences = new();
    public Dictionary<string, Node> Nodes { get; set; } = new
       Dictionary<string, Node>();

    // metoda dodaje cvor u grafu
    public void AddNode(string node)
    {
        Node newNode = new()
        {
            Id = node
        };
        Nodes[node] = newNode;
    }

    // Metoda dodaje granu od cvora srcId do cvora destId, sa
       tezinom weight
```

```
public void AddEdge(string srcId, string destId, int weight)
{
    if (!Nodes.ContainsKey(srcId))
        AddNode(srcId);
    if (!Nodes.ContainsKey(destId))
        AddNode(destId);

    Node srcNode = Nodes[srcId];
    Node destNode = Nodes[destId];
    srcNode.Edges.Add(destId, weight);
    destNode.Previous.Add(srcId);
}

//Rekurzivna metoda trazenja svih optimalnih puteva u grafu,
    pretragu vrsimo od poslednjeg cvora, unazad
public void FindPaths(string vertexId, string startVertexId,
    string endVertexId, Stack<string> path, int lengthS, int
   [] polyL1)
{
    path.Push(vertexId);

    // da li smo stigli do cvora "Pocetak"?
    if (Nodes[vertexId].Previous.Count == 0)
    {
        // previousVertex se inicijalizuje na "Pocetak"
        string previousVertex = startVertexId;
        int[] selectionArray = new int[lengthS];
        int i = 0;

        // iteriramo od poslednjeg cvora koji je dodat u
           path
        foreach (string currentNodeId in path)
        {
            // preskacemo cvor "Pocetak"
            if (currentNodeId != startVertexId)
            {
```

```
// tezina grane (previousVertex) -> (
   currentNodeInPath)
int weight = Nodes[previousVertex].Edges[
   currentNodeId];
if (weight >= 1 && currentNodeId !=
   endVertexId)
{
    // grane oblika (i-k, j-1) -> (i,j)
    // tu imamo k brisanja, i jednakost
       bitova x[i] i y[j] => dakle u
       selektujucem nizu imamo k nula i
       jednu jedinicu
    Array.Fill(selectionArray, 0, i, weight)
       ;
    i += weight;
    selectionArray[i++] = 1;
}
else if (weight >= 1 && currentNodeId ==
   endVertexId)
{
    // grane oblika (i-k, m) -> (Kraj)
    // ne dodajemo nista u selektujuci niz
       jer smo pronasli prefiks koji je
       sveden na niz kljuca
    selectionArray = selectionArray.Take(
       lengthS - weight).ToArray();
    i = i + weight;
    break;
}
else if (weight == 0 && currentNodeId ==
   endVertexId)
    // grana oblika (poslednji element u
       poslednjoj koloni matrice)->(Kraj)
    break;
else
{
    // u svim ostalim slucajevima
```

```
                        selectionArray[i++] = 1;
                    }
                }

                previousVertex = currentNodeId;
            }
            // provera da li je selektujuci niz u saglasnosti sa
                karakteristicnim polinomom polyL1
            if(CheckSeqWithLFSR(selectionArray, polyL1))
                SelectionSequences.Add(selectionArray);
        }
        else
        {
            foreach (string prevVertexId in Nodes[vertexId].
               Previous)
            {
                FindPaths(prevVertexId, startVertexId,
                   endVertexId, path, lengthS, polyL1);
            }
        }
        path.Pop();
    }
}
```

Прилог 2

Дата је програмска реализација алгоритма описаног у одељку 2.2. Пример покретања програма приказан је у одељку 3.2.

```
// metoda AttackR2 koja vrsi napad na taktovani registar R2
static List<(int[], int[])> AttackR2(int[] s, int[] polyL1, int
   [] polyL2)
{
    // duzina R1
    int L_r1 = polyL1.Length - 1;
    // period R1
    ulong T_r1 = (ulong)Math.Pow(2, L_r1) - 1;
    // duzina R2
```

```
int L_r2 = polyL2.Length - 1;
// period R2
ulong T_r2 = (ulong)Math.Pow(2, L_r2) - 1;

// lista koja cuva sva pocetna stanja R1 i R2, koji su
   resenja
List<(int[], int[])> initStates = new();

int lenghtS = s.Length;
for (uint i = 1; i < (uint)1 << polyL1.Length - 1; i++)
{
    // konvertovanje broja i u string sa specificnom bazom
       2, popunjavajuci sa leve strane nulama do
       odgovarajuce dimenzije
    string binaryString = Convert.ToString(i, 2).PadLeft(
       L_r1, '0');
    // pocetno stanje za R1
    int[] initStateR1 = binaryString.Select(c => int.Parse(c
       .ToString())).ToArray();

    // generise se niz koji cuva indekse jedinica u izlaznom
        nizu bitova R1
    int[] indexesOfOnes = GenerateIndexOfOnes(polyL1,
       initStateR1, lenghtS);

    // kreira se klasa BinaryMatrixSolver ciji su clanovi
       Matrix i Vector
    // Matrix je matrica dimezije legthS i L_r2. Svaki njen
       red je linearna kombinacija bitova pocetnog stanja
       nekog bita iz izlaznog niza R2
    // Vector je niz kljuca
    BinaryMatrixSolver binaryMatrix = new(lenghtS, L_r2, s);

    for (int j = 0; j < lenghtS; j++)
    {
        // svaki bit izlaznog niza R2, na odgovarajucim
           pozicijama, predstavljamo kao linearnu
```

```
                kombinaciju bitova pocetnog stanja
            int[] dividend = new int[indexesOfOnes[j] + 1];
            dividend[indexesOfOnes[j]] = 1;

            // ostatak pri deljenju polinoma dividend i polyL2.
                Dividend je polinom oblika x^(d[i])
            // ostatak predstavlja linearnu kombinaciju pocetnih
                 stanja za odredjeni bit izlaznog niza R2
            int[] remainder = DividePoly(dividend, polyL2).
                ToArray();

            // dodaje se linearna kombinacija u matricu Matrix
            binaryMatrix.AddRow(remainder, j);
        }


        //  proverava da li je sistem konzistentan i ako jeste
            vraca resenje, u suprotnom vraca null
        List<int[]>? initStatesR2 = binaryMatrix.Solve();
        if (initStatesR2 is not null)
        {
            foreach (int[] initStateR2 in initStatesR2)
            {
                // za svako dobijeno pocetno stanje R2 vrsi se
                    provera da li sa pocetnim stanjem R1 generise
                     zadati niz kljuca
                if (CheckGenerator(initStateR1, initStateR2,
                    polyL1, polyL2, s))
                {
                    initStates.Add((initStateR1, initStateR2));
                }
            }
        }
    }
    return initStates;
}
```

```
// Metoda koja proverava da li pocetna stanja initStateR1, i
   initStateR2, generisu niz kljuca s
static bool CheckGenerator(int[] initStateR1, int[] initStateR2,
    int[] polyL1, int[] polyL2, int[] s)
{
    int[] initR1 = new int[polyL1.Length - 1];
    int[] initR2 = new int[polyL2.Length - 1];
    Array.Copy(initStateR1, initR1, polyL1.Length - 1);
    Array.Copy(initStateR2, initR2, polyL2.Length - 1);

    int count = 0;
    while (count < s.Length)
    {
        (int, int) outputBits = Clock(initR1, initR2, polyL1,
           polyL2);

        if (outputBits.Item1 == 1)
        {
            if (outputBits.Item2 != s[count++])
                return false;
        }
    }

    return true;
}
// Metoda koja imitira taktovanje registara
static (int, int) Clock(int[] initR1, int[] initR2, int[] polyL1
   , int[] polyL2)
{
    (int, int) outputBits = new();
    int newBitR1 = 0;
    for (int j = 0; j < polyL1.Length - 1; j++)
    {
        newBitR1 = newBitR1 ^ (initR1[j] & polyL1[j]);
    }

    outputBits.Item1 = initR1[0];
```

```
    Array.Copy(initR1, 1, initR1, 0, initR1.Length - 1);
    initR1[initR1.Length - 1] = newBitR1;

    int newBitR2 = 0;
    for (int j = 0; j < polyL2.Length - 1; j++)
    {
        newBitR2 = newBitR2 ^ (initR2[j] & polyL2[j]);
    }

    outputBits.Item2 = initR2[0];
    Array.Copy(initR2, 1, initR2, 0, initR2.Length - 1);
    initR2[initR2.Length - 1] = newBitR2;

    return outputBits;
}

// Metoda koja deli polinom divident i divisor
// Vraca ostatak pri deljenju
static List<int> DividePoly(int[] dividend, int[] divisor)
{
    List<int> remainder = new List<int>(dividend);
    int diff = remainder.Count - divisor.Length;
    while (diff >= 0)
    {
        for (int i = 0; i < divisor.Length; i++)
        {
            remainder[i + diff] = remainder[i + diff] ^ divisor[
                i]; // XOR operacija
        }

        // Uklanjanje vodecih nula
        while (remainder.Count > 0 && remainder.Last() == 0)
        {
            remainder.RemoveAt(remainder.Count - 1);
        }

        diff = remainder.Count - divisor.Length;
```

```
    }
    return remainder;
}

// Metoda koja generise niz, dimenzije numOfOnes, u kome se
   nalaze svi indeksi jedinica iz izlaznog niza
static int[] GenerateIndexOfOnes(int[] poly, int[] state, int
   numOfOnes)
{
    int[] indexesOfOnes = new int[numOfOnes];
    int index = 0;
    int[] tmpState = new int[state.Length];
    Array.Copy(state, tmpState, state.Length);
    for (int i = 0; i < numOfOnes;)
    {
        int newBit = 0;
        for (int j = 0; j < poly.Length - 1; j++)
        {
            newBit = newBit ^ (tmpState[j] & poly[j]);
        }

        if (tmpState[0] == 1)
            indexesOfOnes[i++] = index;

        Array.Copy(tmpState, 1, tmpState, 0, tmpState.Length -
           1);
        tmpState[tmpState.Length - 1] = newBit;

        index++;
    }
    return indexesOfOnes;
}

// Klasa  koja sluzi za proveru konzistentnosti sistema
   linearnih jednacina Matrix*Result = Vector
public class BinaryMatrixSolver
{
```

```
    private int Rows;
    private int Cols;
    public int[,] Matrix;
    private int[] Vector;

    public BinaryMatrixSolver(int rows, int cols, int[] vector)
    {
        Matrix = new int[rows, cols];
        Rows = rows;
        Cols = cols;
        Vector = new int[vector.Length];
        Array.Copy(vector, Vector, vector.Length);
    }

    // Metoda koja proverava konzistentnost sistema linearnih
       jednacina, i vraca resenje ako je sistem konzistentan
    public List<int[]>? Solve()
    {
        // Gausova eliminacija
        for (int i = 0; i < Rows; i++)
        {
            int pivotCol = -1;
            for (int j = 0; j < Cols; j++)
            {
                if (Matrix[i, j] == 1)
                {
                    pivotCol = j;
                    break;
                }
            }

            if (pivotCol == -1) continue;

            for (int j = 0; j < Rows; j++)
            {
                if (i != j && Matrix[j, pivotCol] == 1)
                {
```

```
                AdditionRows(j, i);
                Vector[j] ^= Vector[i];
            }
        }
    }

    int zeroRows = 0;

    // Provera da li je matrica konzistentna
    for (int i = 0; i < Rows; i++)
    {
        bool isZeroRow = true;
        for (int j = 0; j < Cols; j++)
        {
            if (Matrix[i, j] != 0)
            {
                isZeroRow = false;
                break;
            }
        }
        if (isZeroRow && Vector[i] == 0)
            zeroRows++;

        if (isZeroRow && Vector[i] != 0)
        {
            // Sistem nije konzistentan i ima 0 resanja
            return null;
        }
    }

    // Racunanje resenja konzistentnog sistema linearnih
       jednacina
    List<int[]> solutions = new();
    int[] solution = new int[Cols];
    Array.Fill(solution, -1);
    List<int> indexOfEmptyPosition = new();
    for (int i = 0; i < Rows; i++)
```

```
{
    int pivot = -1;
    for (int j = 0; j < Cols; j++)
    {
        if (Matrix[i, j] == 1)
        {
            pivot = j;
            solution[pivot] = Vector[i];
        }
    }

    if (pivot == -1)
    {
        continue;
    }
}

// ako je  Rang(Matrix) == Rang([Matrix|s]) < L2
// treba pronaci indekse "-1" u solution, na tim mestima
    fali resenje, odnosno resenje za tu poziciju je "1"
   ili "0"
for (int i = 0; i < solution.Length; i++)
{
    if (solution[i] == -1)
    {
        indexOfEmptyPosition.Add(i);
    }
}

for (int i = 0; i < Math.Pow(2, indexOfEmptyPosition.
   Count); i++)
{
    solutions.Add((int[])solution.Clone());
}

// ako je Rang(Matrix) == Rang([Matrix|s]) < L2
int k = 0;
```

```
        foreach (int emptyPosition in indexOfEmptyPosition)
        {
            int bit = 0;
            int counter = (int)Math.Pow(2, k);
            foreach (int[] sol in solutions)
            {
                if (counter == 0)
                {
                    counter = (int)Math.Pow(2, k);
                    bit ^= 1;
                    sol[emptyPosition] = bit;
                }
                else
                {
                    sol[emptyPosition] = bit;
                    counter--;
                }
            }
            k++;
        }

        return solutions;
    }

    // Metoda koja sabira red source sa redom target
    private void AdditionRows(int target, int source)
    {
        for (int i = 0; i < Cols; i++)
        {
            Matrix[target, i] ^= Matrix[source, i];
        }
    }

    // Dodavanje linearne kombinacije row u matrix, u vrsti i
    public void AddRow(int[] row, int i)
    {
        for (int j = 0; j < row.Length; j++)
```

```
            Matrix[i, j] = row[j];
    }
}
```

Биографија аутора

Јована Рибаћ рођена је 31.03.1996. у Приштини. Основну школу завршила је 2011.године у Крушевцу као вуковац. Исте године уписује Гимназију у Крушевцу, природно-математички смер. Гимназију завршава као вуковац 2015.године. Исте године уписује Математички факултет Универзитета у Београду. Основне академске студије завршава 2020.године, исте године уписује мастер студије, на модулу Рачунарство и информатика.