Математички факултет
Универзитет у Београду

Мастер рад

Тема: **Алгоритми засновани на рођенданском парадоксу и примене**

СТУДЕНТ
Мина Бадовинац

МЕНТОР
проф. др Миодраг Живковић

ЧЛАНОВИ КОМИСИЈЕ
проф. др Филип Марић
доц. др Весна Маринковић

Београд, 2018.

Сажетак

У теорији вероватноће, проблем рођендана или рођендански парадокс бави се вероватноћом да у групи од n случајно одабраних особа постоје бар две које имају рођендан истог дана. Рођендански парадокс формулише се уз претпоставку да је сваки дан у години (осим 29. фебруара) једнако вероватан за рођендан. Када је број случајно одабраних особа бар 366, јасно је да је вероватноћа таквог догађаја 1. Међутим, вероватноћа од 0.99 се постиже са само 70 случајно одабраних особа, а вероватноћа од 0.5 са само 23 случајно одабране особе.

Постоји доста алгоритамских техника које омогућују примену рођенданског парадокса над произвољном листом случајних објеката. Неке од њих не узимају у обзир детаље и начине на које су ови објекти генерисани. У овом раду анализира се одређенији проблем и потрага за колизијом над објектима који су генерисани итерацијама неке фиксиране функције. У овом случају могућа су многа побољшања. Пре свега, представљени алгоритми имају доста мање меморијске захтеве. Ова побољшања су такође важна за криптоанализу, јер постоји доста примена у којима се она могу искористити. Постоје веома важни алгоритми у теорији бројева, као што је Полардов, који користе ову технику за факторизацију целих бројева или рачунање дискретних логаритама. Поред примене у теорији бројева, у раду се разматрају примене у проналажењу колизија код хеш функција, као и директна криптографска примена у контексту сигурности блоковских шифри.

Садржај

1. Увод

Последице рођенданског парадокса у криптографији су бескрајне: у криптографији јавног и тајног кључа, криптоанализи и безбедности, као и утицај на све подобласти криптографије, од проучавања проточних шифри до теорије бројева. Из тог разлога је значајно истраживање и разумевање рођенданског парадокса и његове примене у криптографији. У овом поглављу најпре су дате основе рођенданског парадокса. Након тога анализиране су границе рођенданског парадокса и представљено је неколико уопштења овог проблема.

Рођендански парадокс је свеобухватна парадигма у криптографији. Он је интересантан пример сукобљавања интуитивног размишљања и математичких прорачуна и из тог разлога се назива парадоксом. Иначе, рођендански парадокс није прави парадокс у смислу да ће нас довести до логичке контрадикције. У неком смислу, може се посматрати као ефикасна пробабилистичка варијација Дирихлеовог принципа. Овај принцип је веома опште правило и наводи да ако је n голубова потребно сместити у m голубарника, при чему је $n > m$, онда постоји најмање један голубарник у којем се налазе барем два голуба. Дакле, уколико је дато n дисјунктних класа објеката и скуп од N објеката, где је $N > n$, онда постоји најмање једна класа која садржи два или више објеката. У овом контексту кажемо да се два објекта сударају у истој класи или једноставно да имамо колизију. Користећи ову терминологију, Дирихлеов принцип једноставно наводи да за n класа, са најмање $n + 1$ објеката, увек постоји колизија. У пробабилистичком контексту, под претпоставком да су објекти равномерно распоређени између класа, природно је поставити питање: Колико објеката би требало да имамо да би се десила колизија са вероватноћом већом од $1/2$? Одговор на ово питање је срж рођенданског парадокса. За n класа није неопходно имати $n/2$ објеката, заправо, број потребних објеката је доста мањи. Иако изненађујуће, тачан одговор је реда величине $\sqrt{n}$.

Проблем рођендана. Питање које се поставља, а које је познато и као проблем рођендана јесте: Колико особа је потребно да се нађе у истој просторији тако да је вероватноћа да две од тих особа имају рођендан истог дана бар $1/2$? Тачан одговор на ово питање је заправо 23. Дакле, уколико се у просторији налазе 23 особе, вероватноћа да ће неке две од њих имати рођендан истог дана је већа од 50%. У поређењу са могућим бројем дана у години (365), само мали број особа је потребан да би се испунио услов проблема. Занимљиво је напоменути да статистика показује да рођендани заправо нису равномерно распоређени. Често постоји пристрасност према одређеним месецима и одређеним данима у години. Ипак, проблем рођендана формулише се уз претпоставку да рођендани јесу равномерно распоређени.

Пре него што се почне разматрање алгоритамске ефикасности у проналажењу колизија важно је проценити вероватноћу постојања колизија у скуповима случајних објеката. Постојање таквих

колизија може бити од суштинског значаја за сигурност криптографских алгоритама и протокола. Наравно, тачна вероватноћа зависи од расподеле ових случајних објеката. На пример, уколико су ови објекти изгенерисани применом случајне пермутације над различитим вредностима или уколико су случајно бирани из скупа без враћања, не може доћи до колизије, тј. вероватноћа постојања колизије у том случају је нула. Са друге стране, вероватноћа да близанци имају рођендан истог дана је скоро један.

Најједноставнији случај за анализу јесте случајно извлачење објеката из датог скупа са враћањем. За анализу овог случаја потребно је размотрити два параметра, број случајно извучених објеката n и величину скупа N. Да би се одредила вероватноћа постојања колизије може се искористити једноставна хеуристичка метода. У скупу од n објеката може се креирати $n(n-1)/2$ парова. С друге стране, за било који пар, вероватноћа да постоји колизија је N^{-1}. Претпостављајући независност између парова, процењује се да је просечан број колизија једнак:

$$\frac{n \cdot (n-1)}{2N}.$$

Ова једноставна хеуристичка анализа већ указује да је граница на којој се колизија јавља око $n \approx \sqrt{N}$. Међутим, ова анализа није задовољавајућа из више разлога. Прво, неправилно претпоставља независност између парова објеката у случајном скупу. Друго, не даје вероватноћу јављања колизије, већ уместо тога процењује очекивани број колизија унутар скупа. Из тог разлога у наставку је извршена прецизнија анализа.

Анализа горње границе. За дати скуп од n случајно извучених објеката из скупа величине N, нека $Coll_N^n$ означава вероватноћу постојања најмање једне колизије у таквом скупу. Ради једноставности, претпоставља се да је редослед извлачења елемената из случајног скупа непромењен. У том случају, сваком елементу може се приступити његовим бројем, од 1 до n. Нека је елемент i означен са X_i. За сваки пар (i, j), вероватноћа да је $X_i = X_j$ је N^{-1}. Уколико постоји пар који је у колизији, онда постоји колизија у скупу. Како је вероватноћа уније догађаја увек одозго ограничена сумом вероватноћа појединачних догађаја, без обзира на независности, важи да је:

$$Coll_N^n \leq \sum_{\text{Парови } (i,j)} \frac{1}{N} = \frac{n \cdot (n-1)}{2N}$$

Из овога директно произилази да је за $n \ll \sqrt{N}$ вероватноћа постојања колизије мала. Ова чињеница се користи у сигурносним доказима када се показује да ниједан рођендански напад није применљив. Следећи став даје нам прецизну оцену, изнад горње границе.

Став: *Нека $k_1, k_2, \dots, k_n \in \{1, \dots, N\}$ и нека је k_j случајна вредност из интервала од 1 до N. Ако знамо да су ове случајне вредности међусобно независне са једнаком расподелом вероватноћа и ако имамо узорак величине $n = 1.2 * \sqrt{N} \approx \sqrt{N}$, тада је вероватноћа да постоји i, $i \neq j$, такво да је $k_i = k_j$ већа или једнака $1/2$.*

У наставку је дат доказ горњег става. Треба имати у виду да се број 23 из рођенданског парадокса добија уколико се испрати логика која је представљена у наставку. Пре него што дамо доказ, наводимо две претпоставке:

- Случајне величине су независне.
- Обрађује се само случај равномерне расподеле вероватноћа. Испоставља се да уколико расподела није равномерна, број узорака који је потребан је заправо мањи од $1.2 * \sqrt{N}$, али тај случај сада неће бити покривен.

Доказ:

$$P\left[\exists i \neq j : k_i = k_j\right] = 1 - P[\forall i \neq j : k_i \neq k_j]$$

Другим речима, одређује се вероватноћа обрнутог догађаја, када не постоји ниједна колизија. Када се одреди, та вредност се одузима од 1 и добија се тражена вероватноћа.

$$P\left[\exists i \neq j : k_i = k_j\right] = 1 - \left(\frac{N-1}{N}\right)\left(\frac{N-2}{N}\right)\ldots\left(\frac{N-(n-1)}{N}\right) = 1 - \prod_{i=1}^{n-1}\left(\frac{N-i}{N}\right)$$

$$= 1 - \prod_{i=1}^{n-1}\left(1 - \frac{i}{N}\right)$$

Сада се може искористити неједнакост $1 - x \leq e^{-x}$ која важи за $x < 1$, што је овде испуњено. Истинитост ове неједнакости следи из Тејлоровог развоја функције e^{-x}, $e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \cdots$ и чињенице да је $\frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \cdots$ позитивна вредност. Применом ове неједнакости добија се:

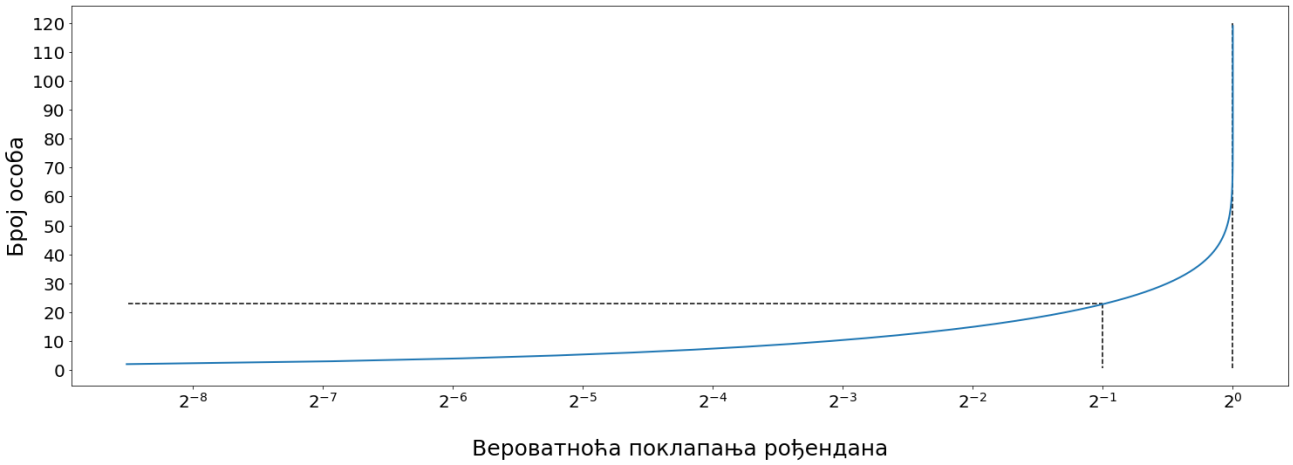
$$P\left[\exists i \neq j : k_i = k_j\right] = 1 - \prod_{i=1}^{n-1}\left(1 - \frac{i}{N}\right) \geq 1 - \prod_{i=1}^{n-1} e^{-\frac{i}{N}} = 1 - e^{-\frac{1}{N}\sum_{i=1}^{n-1} i}$$

Такође, може се апроксимирати и $1 - e^{-\frac{1}{N}\sum_{i=1}^{n-1} i} \approx 1 - e^{-\frac{n^2}{2N}}$, што важи ако је апсолутна вредност експонента много мања од 1. Ако је $n = 1.2 * \sqrt{N}$, добија се $\frac{n^2}{2N} = 0.72$. Сада се лако добија и да је $1 - e^{-0.72} \approx 0.53 > 1/2$, што чини доказ комплетним.

■

Иако доказ укључује само множење вероватноћа и манипулацију једноставним неједнакостима, он даје важну статистичку информацију. Заиста, ако се за N узме 365, добија се да је $n = 1.2 * \sqrt{365} \approx 23$. Интересантно је напоменути да вероватноћа веома брзо опада уколико имамо мање од 23 особе. С друге стране, вероватноћа достиже 1 врло брзо, уколико имамо више од 23 особе. Заправо, са 70 особа вероватноћа је скоро 1, тачније 0.99. За потребе овог рада имплементиран је алгоритам који за дати број особа израчунава вероватноћу да неке две од њих имају рођендан истог дана, као и онај који за дату вероватноћу израчунава колико најмање особа је потребно да

се нађу у истој просторији. Подаци су приказани на слици 1 где је дат график зависности вероватноће да неке две особе имају рођендан истог дана од броја датих особа.



Слика 1: График зависности потребног броја особа од вероватноће да неке две од њих имају рођендан истог дана

У неким криптографским контекстима, занимљиво је уопштити анализу. Једно важно уопштење јесте постојање колизије између два различита скупа. Друго питање на које се често наилази је постојање вишеструке колизије, тј. постојање 3, 4 или више случајних елемената који деле заједничку вредност. За ова уопштења, представљена је само упрошћена хеуристичка анализа заснована на очекиваном броју (вишеструких) колизија под претпоставком независности. Ова поједностављена анализа може бити прилагођена тако да даје горње и доње границе као и у случају колизија. Међутим, у већини случајева ово није неопходно и једноставна процена је довољна.

Колизија између два скупа. За ово уопштење дата су два подскупа, сваки добијен случајним извлачењем без враћања из истог надскупа. Са n_1 и n_2 означимо одговарајуће кардиналности подскупова, а са N кардиналност скупа. Према конструкцији колизија не постоји унутар било којег подскупа и разматра се само колизија између елемената првог скупа и елемената другог скупа. Како се може направити $n_1 \cdot n_2$ парова елемената, очекивани број колизија је:

$$\frac{n_1 \cdot n_2}{N}.$$

Када су ова два подскупа исте величине $n_1 = n_2 = n$, очекивани број колизија је између очекивања које смо имали за један подскуп од n елемента и очекивања за подскуп величине $2n$. Треба имати у виду да је разматрање подскупа величине $2n$ сасвим природно, с обзиром да је то величина уније два разматрана подскупа.

Вишеструка колизија. За вишеструке колизије се такође могу разматрати два случаја. Или је дат један подскуп и трага се за l различитих елемената са истом вредношћу, или је дато l различитих

подскупова и трага се за елементом који је заједнички за све њих. Укратко, вишеструка колизија од l елемената назива се l-тострука колизија. Уз уобичајену хеуристичку анализу под претпоставком независности хипотеза, налазимо да је очекивани број l-тоструких колизија у подскупу величине n скупа од N елемената:

$$\frac{\prod_{i=1}^{l}(n+1-i)}{n!\cdot N^{l-1}} \approx \frac{n^l}{n!\cdot N^{l-1}}$$

За l-тоструку колизију међу подскуповима одговарајућих величина $n_1 \ldots n_l$ налазимо да је очекивани број једнак:

$$\frac{\prod_{i=1}^{l} n_i}{N^{l-1}}$$

Под претпоставком да је l довољно мало да се фактор $l!$ може занемарити, закључујемо да вероватноћа појаве l-тоструке колизије унутар датог подскупа остаје мала за $n \ll N^{(l-1)/l}$.

Неравномерне статистичке расподеле. До сада се трагајући за колизијом претпостављала равномерна случајна расподела објеката који су у потенцијалној колизији. У пракси ово није увек случај и у неким случајевима може се трагати за колизијом под другим статистичким расподелама. Под претпоставком да случајни објекти остају независни једни од других, вероватноћа јављања колизије је увек већа са неравномерном расподелом. Ово произилази из веће вероватноће појаве колизије унутар једног пара случајно извучених објеката. Заиста, под претпоставком случајног извлачења из скупа од n вредности, где се вредност i извлачи са вероватноћом p_i, вероватноћа појаве колизије је:

$$\sum_{i=1}^{n} p_i^2$$

Под условом да је $\sum_{i=1}^{n} p_i = 1$, ова вероватноћа је максимална када су све вредности p_i једнаке $1/n$.

На крају, када објекти нису извучени независно једни од других, не може се много рећи о колизијама. У зависности од прецизних детаља, колизије могу бити или чешће или потпуно нестати. У зависним случајевима, специфична анализа је потребна за сваки од датих проблема.

2. Алгоритамски аспекти

У овом поглављу представљене су идеје и могуће имплементације алгоритама за проналажење циклуса у секвенци итеративних вредности функције, док се њихова примена проучава кроз даља поглавља овог рада. Основни циљ јесте проналажење колизије над објектима који су генерисани итерацијама неке фиксиране функције. За разлику од уобичајених алгоритама за одређивање периода низа, ови алгоритми користе знатно мању количину меморије за чување података. Пре преласка на алгоритме, даље у тексту прво је дат увид у опште претпоставке и представљен је главни задатак.

Дат је скуп S и функција $F: S \to S$. Полазећи од тачке X_0 из S, генерише се низ, према:

$$X_{i+1} = F(X_i), i \geq 0$$

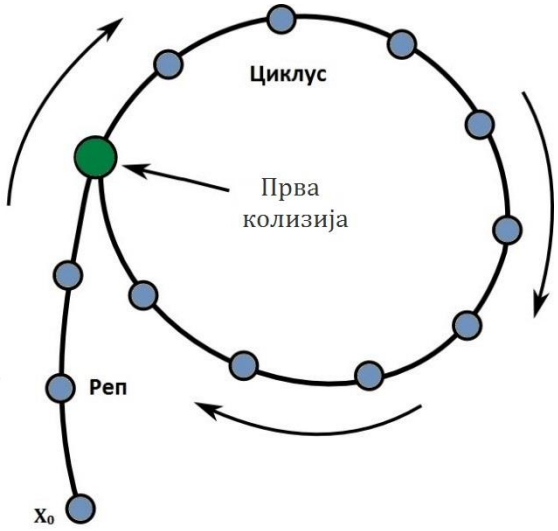
Главни задатак је пронаћи колизију у датом низу, рецимо $X_i = X_j$. Треба имати у виду да се, због детерминистичке природе израчунавања чланова низа, из било које такве колизије могу добити многе. Када год је $X_i = X_j$, налазимо да је $X_{i+1} = F(X_i) = F(X_j) = X_{j+1}$ и поново се добија колизија. Даљом итерацијом добија се $X_{i+2} = X_{j+2}$, итд.

У циљу проучавања овог алгоритамског проблема, важно је претпоставити да се F понаша мање или више случајно. Без ове претпоставке не би постојао разлог за коришћење рођенданског парадокса. Штавише, уколико се ова претпоставка избаци лако је направити контрапримере. Нека је $S = [0 \dots 2^n - 1]$ скуп целих бројева по модулу 2^n и $F(X) = X + 1 \ (mod\ 2^n)$. Тада се за било које X_0 не може пронаћи колизија у низу X, осим уколико се не размотри цео скуп од 2^n елемената. У овом случају, рођендански парадокс би предвидео колизију после $2^{n/2}$ елемената. Према томе, видимо да није применљив на овај контрапример.

У циљу теоријског анализирања понашања датих низова, F се обично моделује као случајно пресликавање. Овакав модел и одговарајућа анализа представљени су у поглављу 3 овог рада. За сада, једноставно претпостављамо да се након неког времена низ X поново враћа у неку од својих претходих вредности. Како је F функција, а не пермутација, ништа не гарантује да ће се вратити у X_0. Насупрот, обично је то нека каснија позицију у низу. Уобичајен начин да се ова чињеница представи је цртање низа X, водећи рачуна о томе да узастопни елементи низа буду суседи на цртежу. Као резултат, представљено на слици 2, добија се облик који подсећа на слово ρ. Из овог разлога, Полардов ρ алгоритам факторизације носи такво име. На слици се може уочити да низ има два одвојена дела: циклус и реп. Ако је прва колизија у низу X колизија између X_s и X_{s+l}, онда реп садржи елементе низа од X_0 до X_{s-1} , а циклус почиње од X_s и његова дужина је l, период низа X.

Главни задатак у овом поглављу је пронаћи колизију у низу X или још боље, пронаћи дужине циклуса и репа низа X. Једно решење овог проблема је да се користе уобичајени алгоритми за

тражење понављања у низу. На пример, може се користити бинарно стабло претраге да се у њега смештају елементи низа X, у сваком тренутку по један све док се не појави први дупликат. Прво понављање се догађа у тренутку убацивања X_{s+l} у стабло. Тада се долази до закључка да је та вредност већ присутна тј. присутно је X_s. Време извршавања овог алгоритма је $O\big((s+l)\log(s+l)\big)$ и чува се $O(s+l)$ елемената. Време извршавања је суштински оптимално јер, осим што функција F има врло специфична својства, мора се изгенерисати низ све до X_{s+l} да би његов циклус био детектован. Штавише, овај алгоритам директно даје и дужину циклуса l и дужину репа s. Међутим, дужина периода који се овим алгоритмом може открити ограничена је величином расположиве меморије. На срећу постоје алтернативни алгоритми којима не треба толика количина меморије. Они не дају директно s и l, тј. не проналазе одмах прву колизију већ неку каснију из циклуса. Међутим, са разумним повећањем цене и уз додатна израчунавања могуће је добити s и l полазећи од прве пронађене колизије.

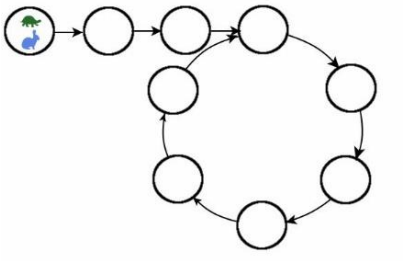


Слика 2: Облик ρ добијен током случајног пресликавања

Почињемо са представљањем два једноставнија алгоритма који задовољавају додатни рестриктиван услов. Прецизније, ова два алгоритма приступају вредностима низа X на рестриктиван начин, или као аргументима функције F или при испитивањима на једнакости. Овакво коришћење вредности низа може бити веома важно јер дозвољава алгоритмима за проналажење циклуса да буду коришћени у контексту црне кутије, где се низу X може приступати само индиректно. Ова рестрикција је посебно битна за Полардов ρ алгоритам факторизације (поглавље 4.1). Заиста, у Полардовој ρ факторизацији, испитивање на једнакост може се извести једино индиректно, кроз израчунавање највећег заједничког делиоца (НЗД).

2.1. Флојдов алгоритам за проналажење циклуса

Флојдов алгоритам за проналажење циклуса такође је познат и као алгоритам корњача и зец. Овај алгоритам је прилично једноставан и један је од најпопуларнијих алгоритама за проналажење циклуса. Идеја алгоритма је да корњача и зец крену из исте почетне тачке, као што је приказано на слици 3, а да се кроз сваку наредну итерацију корњача помера за један корак, а зец за два корака. Уколико постоји циклус, зец ће се наћи у њему и вероватно га обићи више од једног пута, док се коначно не сретне са корњачом када она уђе у циклус. Уколико не постоји циклус, зец ће доћи до краја пута, а на том путу се неће сусрести са корњачом.



Слика 3: Корњача и зец на путу облика ρ

Формалније, дефинишемо нови низ Y (зец), чија је веза са низом X (корњача) дата следећом једнакошћу:

$$Y_i = X_{2i}.$$

Алтернативно, низ Y може бити дефинисан изразима:

$$Y_0 = X_0 \text{ и } Y_{i+1} = F(F(Y_i)).$$

За овако дефинисано Y Флојдов алгоритам проналази први индекс t такав да је $Y_t = X_t$. Јасно, ово може бити обављено у времену $O(t)$ користећи меморију величине $O(1)$, као што је представљено алгоритмом 1. Довољно је рачунати вредности чланова два низа паралелно, док се не поклопе.

```
Улаз: Почетна тачка X0, max број итерација M
x ← X0
y ← x
for i from 1 to M do
   x ← F(x)
   y ← F(F(y))
   if x = y then
      print 'Колизија између чланова низа на индексима i и 2i, пар (x, y)'
      exit
print 'Свих M чланова низа су различити'
```

Алгоритам 1: Флојдов алгоритам за проналажење циклуса

За анализу понашања наведеног алгоритма потребно је разумети везу између позиције t (прве колизије између X и Y), дужине циклуса l и дужине репа s низа X. Прво, како је $X_{2t} = X_t$, закључујемо да је t период низа X и према томе умножак периода l. Друго, како је X_t у циклусу низа X, онда $t \geq s$. Заправо, t је најмањи умножак броја l већи (или једнак) од s:

$$t = \left\lceil \frac{s}{l} \right\rceil l$$

Штавише, једном када је t познато добијање дужине циклуса l је једноставно, довољно је рачунати низ X_{t+i} док се не дође поново до X_t. Ово ће се десити након тачно l корака.

2.2. Брентов алгоритам за проналажење циклуса

Главни недостатак Флојдовог алгоритма је чињеница да сваки корак захтева три примене функције F. Брентов алгоритам за проналажење циклуса је алтернатива која поред тога што ради у линеарном времену, захтева мање корака од Флојдовог алгоритма. Наравно, идеја и реализација Брентовог алгоритма су и нешто сложеније.

Брентов алгоритам се такође може описати на примеру корњаче и зеца. Карактеришу га покретни зец и непокретна корњача која се једино може преместити. Наиме, на почетку се корњача и зец налазе на истој почетној тачки, баш као и код Флојдовог алгоритма (слика 2). У свакој наредној итерацији зец се помера за један корак, док корњача мирује. Након неког времена корњача се премешта у тачку у којој се у том тренутку налази зец, а зец наставља да се креће. Време чекања пре првог премештања корњаче је један корак зеца, а време чекања за свако наредно премештање, од тренутка када је последњи пут премештена, је дупло веће него за претходно. Уколико зец дође у исту тачку у којој је непокретна корњача онда очигледно постоји циклус, уколико достигне крај пута онда нема циклуса.

Зашто се уопште корњача премешта? Циклус можда не садржи цео пут којим се зец креће и ако се на таквом путу зец нађе у циклусу без корњаче, он ће остати заглављен и кретаће се заувек унутар циклуса. Зашто се за време чекања на следеће премештање корњаче узима дупло дужи интервал него претходног пута? У једном тренутку време чекања између два премештања ће постати дуже од величине циклуса, а тада ће корњача бити у циклусу чекајући зеца да стигне у ту тачку.

Једном када је позната тачка X_k унутар циклуса, може се приметити да је дужину циклуса l могуће пронаћи у тачно l корака. То може бити обављено коришћењем истог метода као код проналажења l у Флојдовом алгоритму. Потешкоћу представља проналажење почетне тачке циклуса. У Брентовом алгоритму овај проблем се решава покушајима са неколико почетних тачака, све док се не пронађе адекватна. У циљу минимизације укупних трошкова, израчунавања коришћена за проналазак потенцијалног циклуса зависна су и такође се користе за одређивање

следеће почетне тачке за тестирање. Прецизније, претпоставља се да је низ почетних тачака X_1, X_2, $X_4, \ldots, X_{2^i}, \ldots$ индексиран степенима броја 2. За сваку почетну тачку X_{2^i} рачуна се низ следбеника X_{2^i+j}, све док се не пронађе циклус или док се не дође до $X_{2^{i+1}}$. У првом случају алгоритам се завршава, док се у другом случају прелази на следећу почетну тачку $X_{2^{i+1}}$. Овај приступ приказан је алгоритмом 2.

```
Улаз: Почетна тачка X_0, max број итерација M
x ← X_0
y ← x
trap ← 0
nexttrap ← 1
for i from 1 to M do
  x ← F(x)
  if x = y then
    print 'Колизија између чланова низа на индексима trap и i, пар
          (x, y)'
    exit
  if i = nexttrap then
    trap ← nexttrap
    nexttrap ← 2 · trap
    y ← x
print 'Свих M чланова низа су различити'
```

Алгоритам 2: Брентов алгоритам за проналажење циклуса

Као и раније, понашање Брентовог алгоритма анализира се када се у обзир узму дужина циклуса l и дужина репа s низа X. Он је успешан за почетну тачку X_{2^k} ако и само ако $2^k \geq max(s, l)$. Заиста, ако је пронађен циклус са датом почетном тачком, онда та тачка мора бити у циклусу тј. $2^k \geq s$ и дужина циклуса l може бити највише 2^k. Тачније, сложеност Брентовог алгоритма је $O(s + l)$.

На крају, треба имати у виду да постављање позиција у облику 2^k није једини избор. У зависности од задатог проблема могу се разматрати различити брзорастући низови. На пример то може бити Фибоначијев низ.

2.3. Проналажење почетка циклуса

Видели смо да и Флојдов и Брентов алгоритам могу бити коришћени за проналажење дужине циклуса l низа X. Међутим, ниједан од њих не даје дужину репа s или еквивалентно, позицију почетка циклуса. Из угла проналажења колизије ова позиција је јако важна. Заиста:

$$F(X_{s-1}) = X_s = X_{s+l} = F(X_{s+l-1}) \text{ и } X_{s-1} \neq X_{s+l-1}.$$

Према томе, проналажење уласка у циклус открива прву колизију у F и ово је једино место у низу X где таква колизија може бити пронађена.

Након Флојдовог или Брентовог алгоритма познат је број $t \leq 2(s + l)$, такав да је X_t у циклусу низа X. Према томе, почетак циклуса је цео број s из интервала $[0 \ldots t]$. На основу ове информације могуће је пронаћи s на неколико различитих начина. У наставку су представљена два начина за решавање овог проблема.

```
Улаз: Почетна тачка X0, дужина циклуса l, max број итерација M
x ← X0
y ← x
for i from 1 to l do
   y ← F(y)
if x = y then
  print 'Циклична секвенца, прва колизија је између чланова низа на
         индексима 0 и l, пар (x, y)'
  exit
for i from 1 to M do
   x′ ← F(x)
   y′ ← F(y)
   if x′ = y′ then
      print 'Прва колизија је између чланова низа на индексима i и i + l,
            пар (x′, y′)'
      exit
   x ← x′
   y ← y′
print 'Свих M чланова низа су различити'
```

Алгоритам 3. Алгоритам за проналажење почетка циклуса

2.3.1. Бинарна претрага

Једном када је l познато, за било које σ из интервала од 0 до t могуће је установити да ли је $s \leq \sigma$, испитивањем да ли важи једнакост $X_\sigma = X_{\sigma+l}$. Заиста, ако је услов испуњен X_σ је у циклусу и $\sigma > s$, иначе $\sigma < s$. Ово испитивање може се обавити израчунавањем низа X до $\sigma + l$, што захтева $O(s + l)$ корака.

Проналажење елемента у сортираном низу може се ефикасно обавити коришћењем бинарне претраге. Ова претрага захтева $O(\log N)$ операција ако је дужина низа N. Уколико елемент није присутан у сортираном низу, алгоритам може вратити позицију у низу на којој би елемент требало да буде смештен: почетак, крај или нека од позиција између два узастопна члана низа. Према томе, користећи бинарну претрагу s се може пронаћи алгоритмом сложености $O((s + l) \log(s + l))$.

2.3.2. Директна претрага

Главни проблем претходне методе је што се више пута израчунава неколико истих делова низа, тј. прерачунава се иста информација више пута. Да би се избегао овај недостатак приступа се на други начин. Прво се израчунава X_l. Даље, од X_0 и X_l паралелно се израчунавају два низа X_i и X_{l+i}, испитујући једнакост у сваком кораку. Када се ова два низа подударе добија се да је $s = i$. Идеја је представљена алгоритмом 3.

Директна претрага омогућава израчунавање вредности s и проналажење прве колизије (осим за $s = 0$) у времену $O(s + l)$. На основу овога, видимо да је проналажење колизије у рекурзивно дефинисаном низу овог типа практичније него проналажење колизије у листи случајно изабраних објеката.

2.4. Нивашов алгоритам за проналажење циклуса

Као што је показано, и Флојдов и Брентов алгоритам детектују циклус у низу X коришћењем X на рестриктиван начин, или преко функције F или испитивањем једнакости над вредностима чланова низа. Ово јесте веома корисно у неким случајевима, међутим некада је низ X директно доступан. Према томе, поставља се питање да ли алгоритми за проналажење циклуса могу бити унапређени коришћењем више израчунавања која зависе од вредности. Зачуђујућ је одговор да је то заиста могуће. Први корак у овом правцу може се постићи употребом методе истакнуте тачке, као што је представљено на примеру у раду [6]. Ово се састоји од памћења вредности које имају одређена својства, тј. памћења истакнутих тачака и чекања на њихов повратак. Главна потешкоћа је изабрати добру вероватноћу појављивања истакнутих тачака. Ако су веома учестале онда је број вредности које се морају чувати велики и алгоритам је неефикасан. Ако су истакнуте тачке ретке онда је могуће да циклус не садржи ниједну истакнуту тачку и техника је неупотребљива.

Бољи начин коришћења вредности чланова низа у алгоритму за проналажење циклуса предложио је Ниваш (*Gabriel Nivasch*) 2004. године у раду [2]. Главна предност овог алгоритма је што гарантује заустављање негде између X_{s+l} и X_{s+2l-1}, тј. у току другог проласка кроз циклус низа. Ово је посебно корисно код низова где је l мало у поређењу са s. Главна идеја је одржавање стека већ пронађених вредности на начин који гарантује детекцију циклуса. Да би креирао овакав стек, Нивашов алгоритам се ослања на постојање потпуне уређености на скупу вредности низа и фокусира се на чување малих вредности на стеку.

Главни задатак је осигурати да је најмања вредност из циклуса увек смештена на стеку и присутна за детекцију у току другог проласка кроз циклус. У алгоритамском погледу имплементација оваквог стека није тежак посао. Када год се израчуна нови члан X_i низа X, са стека се уклањају све вредности веће од X_i и X_i се додаје на стек. Нека се на стеку налазе редом вредности $m_1, m_2, m_3, \ldots$. Јасно, минимална вредност из циклуса је једна од ових вредности,

рецимо нека је то m_k. Било који члан низа X мањи од m_k је ван циклуса, дакле у делу пре циклуса. Како се m_k појављује после ових, могуће мањих елемената, према дефиницији стека она ће бити постављена на стек и остати тамо касније.

Лако је уочити да је стек у растућем поретку ако се X_i увек додаје на врх стека. Према томе, брисање већих вредности може се лако обавити коришћењем бинарне претраге над стеком и скраћивањем стека за део после тачке у којој додајемо X_i. У раду [2] је показано да је очекивана величина стека у тренутку i једнака $O(\log i)$. Дакле, изузев у јако ретким случајевима, меморијски захтеви Нивашовог алгоритма су мали. За анализу временске сложености овог алгоритма у обзир треба узети два скупа операција: израчунавања функције F и остале операције. Заиста, у скоро свим случајевима, израчунавања вредности функције F доминирају у времену извршавања. Јасно, како се X израчунава једном за све разматране позиције, све до највише позиције $s + 2l - 1$, то ограничава број израчунавања функције F. Бинарна претрага унутар стека доминира над осталим операцијама и цена сваке претраге је логаритамска у односу на величину стека. Према томе, временска сложеност осталих операција је ограничена са $O((s + 2l)\log(\log(s + 2l)))$. Ова сложеност може бити већа од сложености примена функције F. Међутим, треба имати у виду да F оперише над бројевима који имају најмање $O(\log(s + 2l))$ битова, иначе би циклус био краћи. Дакле, свако израчунавање функције F има цену најмање $O(\log(s + 2l))$, па израчунавања функције F ипак доминирају у времену извршавања.

Према томе, када је применљив, Нивашов алгоритам даје добро побољшање алгоритама за проналажење циклуса. Још једна предност у поређењу са Флојдовим или Брентовим алгоритмом је та што као директан излаз даје дужину циклуса l. Нивашов алгоритам представљен је алгоритмом 4.

Улаз: Почетна тачка X_0, max број итерација M
креирање празног стека S
x ← X_0
додати пар (x, 0) на стек
for i from 1 to M **do**
  x ← F(x)
  тражити x бинарном претрагом по (сортираној) првој компоненти
  if (x, j) је пронађено **then**
    print 'Колизија између чланова низа на идексима j и i'
    exit
  else
    знамо да је $S_k < x < S_{k+1}$
    скратити S за део након тачке S_k
    додати пар (x, i) на врх стека S
print 'Свих M чланова низа су различити'

Алгоритам 4: Нивашов алгоритам за проналажење циклуса

2.4.1. Проналажење почетка циклуса

Након Нивашовог алгоритма и даље може бити потребан проналазак уласка у циклус и добијање прве колизије. У том случају, као и раније, може се искористити алгоритам 3. Међутим, овај алгоритам се сада може побољшати коришћењем већ прикупљених информација. Нека је колизија детектована у тренутку i. Размотримо стање стека у тачки где је детектована колизија. Познато је да је тренутна вредност x једнака вредности са стека $S_k = (x, j)$. Јасно, захваљујући принципима Нивашовог алгоритма, x је минимална вредност из целог циклуса. Према томе, осим за $k = 0$, S_{k-1} одговара вредности која је изван циклуса. Ако важи да је $S_{k-1} = (X_t, t)$, вредност X_t може се искористити као замена за почетну тачку X_0 у алгоритму 3. Када је $k = 0$ користи се оригинална почетна тачка X_0. Све у свему, ово омогућава нешто брже одређивање почетне тачке циклуса.

3. Анализа случајних пресликавања

Добро понашање Флојдовог и Брентовог алгоритма зависи од функције F, коришћене за дефинисање разматраног рекурзивног низа. Већ знамо да се неке функције лоше понашају, као на пример $F(X) = X + 1\ (mod\ 2^n)$. С тим у вези, у овом поглављу представљена су случајна пресликавања.

Параметри случајних пресликавања су проучавани дуго времена. Флажолет (*Philippe Flajolet*) и Одлиско (*Andrew M. Odlyzko*) у свом раду [3] дају добар преглед ове теме и општи приступ за анализу асимптотског понашања параметара случајних пресликавања. У овом поглављу представљен је приступ који су они предложили и дат је кратак преглед неких од основних својстава случајних пресликавања. Све чињенице потичу из наведеног рада и односе се на граф случајних пресликавања на скупу од n елемената. То су асимптотски резултати и дају асимптотску процену за различите параметре када n тежи бесконачности.

3.1. Случајна пресликавања и функционални граф

Случајна пресликавања су пробабилистички метод који проучава карактеристике пресликавања скупа од n елемената на самог себе. Због једноставности, претпоставља се да ова пресликавања оперишу над скупом $\{1 \dots n\}$.

Дефиниција: *Нека је* $\mathrm{F}_n = \{\varphi \mid \varphi: \{1, 2, \dots n\} \to \{1,2, \dots n\}\}$ *скуп свих функција које пресликавају скуп од* n *елемента у самог себе. Кажемо да је* Φ *случајно пресликавање ако сваку вредност* $\varphi \in \mathrm{F}_n$ *узима са истом вероватноћом* $P[\Phi = \varphi] = 1/n^n$.

Случајна пресликавања представљају или хеуристички или тачан модел за различите примене у генерисању случајних бројева, теорији бројева, криптографији и анализи алгоритама уопште. Случајна пресликавања такође се јављају у многим проблемима дискретне вероватноће. Размотримо, на пример, следеће тврдње:

1. Насумично се убацује n лоптица у m кутија. Тада се очекује да ће око $e^{-n/m}$ кутија остати празне. [Хеширање]
2. У соби се налазе 23 особе. Већ смо видели да је добра идеја кладити се да две особе у соби имају исти рођендан (шансе су 50,7% у нашу корист). [Рођендански парадокс]
3. Чоколадице које се продају у трафици садрже сличице. Постоји n могућих различитих сличица. За сакупљену комплетну колекцију сличица очекује се да је потребно купити око $n \log n$ чоколадица. [Проблем колекционара сличица].
4. Уколико се за генерисање случајних бројева користи метод средњег дела квадрата (или неки други "насумично" дизајниран генератор случајних бројева) који ради са l цифара, очекује се

да ће он ући у циклус након приближно $2^{l/2}$ корака. ["Насумични" генератори случајних бројева].

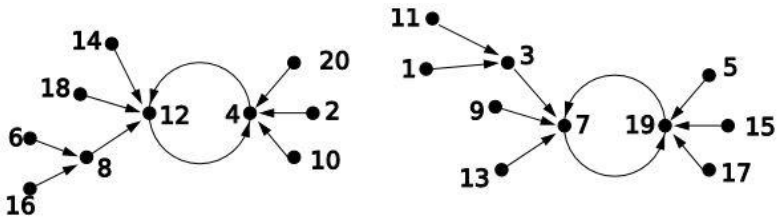
5. Полардов алгоритам за факторизацију вероватно ће наћи фактор сложеног целог броја n унутар $O(n^{1/4})$ корака. [Полардов ρ-метод].
6. Постоји n шпијуна који присуствују криптографској конференцији и остављају своје шешире у гардероби. Када се предавање заврши, сваки шпијун насумице узима шешир. Тада са вероватноћом око e^{-1} нико неће имати свој шешир на глави. [Проблем неуређености].

Одмах се може приметити да ова тврђења носе неке информације о (случајним) пресликавањима из коначног скупа на коначан скуп. Нека $\mathrm{F}_n^{<m>}$ означава колекцију свих функција које коначан скуп величине n пресликавају на коначан скуп величине m. За посебан случај када је $n = m$, користи се ознака $\mathrm{F}_n \equiv \mathrm{F}_n^{<m>}$. Модели у којима се разматрају случајни елементи из F_n познати су као модели случајних пресликавања.

Тврђење 1 је типично за читав низ проблема који се представљају приликом анализе очекиваних перформанси хеш алгоритама [7]. Тврђење 2 је познато из ранијих поглавља овог рада и видели смо да своју славу дугује прилично неинтуитивној ниској вредности 23. Тврђење 3 је нешто компликованије од ранијих, пошто је сада и само m случајна променљива у процесу. Међутим, ако се размотри вероватноћа догађаја да је фиксиран број чоколадица m довољан за целу колекцију, онда се може приметити да се овај проблем своди на стандардни статистички проблем у $\mathrm{F}_n^{<m>}$. Тврђење 4 је блиско теми овог рада, јер се бави итеративном структуром коначног скупа у себе. То је тврђење која се тиче $\mathrm{F}_n^{<m>}$ за $n = m = 2^l$. Оно што ово тврђење у суштини каже јесте да случајно пресликавање $\varphi \in \mathrm{F}_n$ има тенденцију да уђе у циклус након приближно $\sqrt{n}$ корака. Као што је познато, ова чињеница у комбинацији са Флојдовом идејом за тестирање генератора случајних бројева довела је до Полардове ρ-методе факторизације (тврђење 5). Ово је довело и до факторизације осмог Фермаовог броја $F_8 = 2^{2^8} + 1$, видети [5]. Последње тврђење односи се на случајне permутације које чине посебан подскуп F_n.

Приступ Флажолета и Одлиска да пронађу асимптотске параметре случајних пресликавања заснива се на идеји о анализи структуре функционалног графа функције $\varphi \in \mathrm{F}_n$.

Дефиниција: *За функцију* $\varphi \in \mathrm{F}_n$, *функционални граф дефинише се скупом чворова* $V(\varphi) = \{1, 2, \ldots n\}$ *и скупом усмерених грана* $\mathrm{E}(\varphi) = \{(i, \varphi(i)) \mid 1 \le i \le n\}$.



Слика 4: Пример функционалног графа за $\varphi: x \to ((x-1)^2 + 2) \pmod{20} + 1 \in \mathrm{F}_n$

Дакле, свако случајно пресликавање φ на скупу S_n од n елемената се такође посматра и као усмерен граф G_φ чији чворови v су тачке из S_n и чије усмерене гране су облика $(v, \varphi(v))$, за све чворове v из S_n. Сваки чвор графа има излазни степен 1, док је улазни степен променљив и може бити 0 или неки већи број. Наравно, просек излазних степена једнак је просеку улазних степена, тј. 1.

3.2. Општа својства

Повезане компоненте. Повезана компонента добијена је груписањем свих тачака до којих се може доћи из почетне тачке пратећи гране графа G_φ у било ком смеру. На примеру функционалног графа (слика 4) могу се уочити две повезане компоненте. Асимптотски, број повезаних компоненти у графу G_φ је око $\ln\sqrt{n}$.

Листови. Лист у графу G_φ је чвор v улазног степена нула. Еквивалентно, то је елемент v из S_n који не припада слици функције φ. На слици 4 неки од листова су: 2, 10, 20, итд. Асимптотски, број листова у графу G_φ је око n/e.

Унутрашњи чворови. Унутрашњи чвор у графу G_φ је чвор v улазног степена већег од нула. Еквивалентно, то је елемент v из S_n који припада слици функције φ. На слици 4 неки од унутрашњих чворова су: 8, 12, 4, итд. Асимптотски, број унутрашњих чворова у графу G_φ је око $(1 - 1/e)n$.

Изведени унутрашњи чворови. Елемент v из S_n је k-ти изведени унутрашњи чвор ако и само ако постоји елемент x из S_n такав да је $v = \varphi^{(k)}(x)$, где $\varphi^{(k)}$ означава k узастопних примена функције φ. Може се приметити да је k-ти изведени унутрашњи чвор такође и $(k-1)$-и изведени унутрашњи чвор. На слици 4, на примеру функционалног графа може се приметити да је чвор 8 други изведени унутрашњи чвор, а чворови 12 и 4 четврти изведени унутрашњи чворови. Асимптотски, број k-тих изведених унутрашњих чворова у графу G_φ је око $(1 - \tau_k)n$, где је τ_k дефинисано рекурзивним изразима:

$$\tau_0 = 0 \text{ и } \tau_{k+1} = e^{\tau_k - 1}.$$

Циклични чворови. Цикличан чвор у графу G_φ је чвор v који припада неком циклусу графа G_φ. Еквивалентно, то је елемент v из S_n који за неко k може бити добијен као свој k-ти изведени унутрашњи чвор, тј. $v = \varphi^{(k)}(v)$. На слици 4 циклични чворови су: 12, 4, 7 и 19. Асимптотски, број цикличних чворова у графу G_φ је око $\sqrt{\pi n/2}$.

3.3. Локална својства

Локална својства односе се на граф G_φ када се он посматра из случајне почетне тачке. Ова својства су веома важна јер углавном диктирају понашање алгоритама за проналажење циклуса.

Дужина репа и циклуса. Дужина репа је удаљеност између почетне тачке и тачке почетка циклуса. Дужина циклуса је удаљеност између тачке почетка циклуса до њеног првог понављања. За случајну почетну тачку просечна дужина репа је око $\sqrt{\pi n/8}$. Просечна дужина циклуса једнака је просечној дужини репа, дакле око $\sqrt{\pi n/8}$. Према томе, просечна дужина путање (реп и циклус) добијена рекурзивном применом функције φ из случајне почетне тачке је око $\sqrt{\pi n/2}$.

Величина повезане компоненте и величина стабла. Просечна величина повезане компоненте која садржи дату почетну тачку у графу G_φ је око $2n/3$. Ако се ограничи на нециклични део ове компоненте и ако се посматра стабло које садржи све чворове који на истом месту улазе у исти циклус као и почетна тачка, просечна величина стабла у графу G_φ је око $n/3$. На основу ових резултата примећује се да је велики део тачака у графу G_φ груписан у једну повезану компоненту. Ова компонента се често назива гигантска компонента.

Број претходника. Претходник изабране почетне тачке је свака тачка која се у њу пресликава. Просечан број претходника изабране почетне тачке је око $\sqrt{\pi n/8}$.

3.4. Екстремална својства

Најдуже путање. Важно питање везано за случајно пресликавање јесте његово понашање када је изабрана најгора могућа почетна тачка. Колико траје алгоритам проналажења циклуса у овом случају? На срећу, очекивана дужина најдужег репа, најдужег циклуса и најдуже путање (реп и циклус) је истог реда као и у просечном случају и износи $O(\sqrt{n})$. Само се константе разликују.

Прецизније, очекивана дужина најдужег циклуса је $c_1\sqrt{n}$, где је $c_1 \approx 0.782$. Очекивана дужина најдужег репа је $c_2\sqrt{n}$, где је $c_2 = \sqrt{2\pi}\ln 2 \approx 1.737$. Очекивана дужина најдуже путање је $c_3\sqrt{n}$, где је $c_3 \approx 2.415$. У раду [3] је примећено да је $c_3 < c_1 + c_2$. Као последица тога, за незанемарљив број случајних пресликавања најдужи реп не води до најдужег циклуса.

Гигантска компонента. Друга два значајна екстремална својства су просечна величина гигантске компоненте и просечна величина највећег стабла у графу G_φ. Према резултатима о просечној величини повезане компоненте и просечној величини стабла за случајну почетну тачку, знамо да су ове величине веће од $2n/3$ и $n/3$, респективно. Прецизније, очекивана величина гигантске компоненте је $d_1 n$, где је $d_1 \approx 0.758$. Очекивана величина највећег стабла је $d_2 n$, где је $d_2 \approx 0.48$.

4. Примене у теорији бројева

Теорија бројева је грана математике која се пре свега бави проучавањем својстава целих бројева. Има врло дугу и богату историју (Еуклид, Ојлер, Гаус). Дуго је сматрана најчистијом граном математике, у смислу да је била најдаља од било каквих конкретних примена. Данас је теорија бројева једна од најважнијих грана математике за примене у криптографији и сигурности размене информација.

Неки од практичних проблема теорије бројева јесу факторизација природних бројева и проблем дискретног логаритма.

Факторизација природних бројева. Основна теорема аритметике (или теорема о јединственој факторизацији) јесте тврђење да сваки природан број већи од 1 је или прост број или се може на јединствен начин представити као производ простих бројева, до нивоа редоследа чинилаца. Факторизација великих природних бројева је тежак проблем и управо су на његовој тежини засновани неки од савремених метода за шифровање. Када се говори о проблему факторизације природног броја N подразумева се да се тражи било који нетривијалан фактор броја N, па се претпоставља да је N непарно. Тако је $105 = 7 \cdot 15$ успешна факторизација, без обзира на то што је 15 сложен број.

Проблем дискретног логаритма. Нека је G коначна циклична група реда n. Нека је g генератор групе G и h произвољан елемент из G. Под дискретним логаритмом броја h по бази g, у ознаци $\log_g h$, подразумева се јединствени цели број α, $0 \leq \alpha \leq n-1$, такав да је $h = g^\alpha$. У коначној мултипликативној групи G реда n, за свако $b \in G$ и свако $\alpha \in \mathbb{Z}$ лако је израчунати b^α. За разлику од тога, за дато $a \in G$ пронаћи број $\alpha \in \mathbb{Z}$ такав да је $b^\alpha = a$, тј. решити проблем дискретног логаритма је тешко.

Алгоритам факторизације заснован на чињеници да се у току случајног лутања по скупу од n елемената очекује наилазак на већ раније прегледани елемент након отприлике $1.2\sqrt{n}$ корака, био је први алгоритам суштински бржи од факторизације низом пробних дељења. Овај алгоритам назива се Полардов ρ алгоритам, а зашто носи такав назив показано је у поглављу 2 овог рада. Даље у тексту представљена је идеја Полардовог алгоритма факторизације. Такође је показано да се иста идеја може користи за решавање проблема дискретног логаритма, као и да ове методе у великој мери користе Флојдов и Брентов алгоритам.

4.1. Полардов ρ алгоритам факторизације

Полардов ρ метод је веома практичан алгоритам факторизације који је Полард (*John Pollard*) представио 1975. године у свом раду [13]. Овај алгоритам има за циљ проналажење релативно

малих фактора целог броја N. Он у великој мери користи Флојдов и Брентов алгоритам у циљу проналажења циклуса у рекурзивно дефинисаном низу. За Полардов ρ метод, низ који се разматра мора да поседује још једно додатно својство. Прецизније, ако означимо овај низ са X, захтева се да за било који прост фактор p броја N, низ $X\ (mod\ p)$ такође буде дефинисан рекурзивним изразима. Ово ограничава могуће изборе за функцију F која се користи за израчунавање низа. Да би ово својство било обезбеђено и низ коректно дефинисан по модулу сваког од простих фактора, добар приступ је да се за F изабере полином. У суштини, најчешћи избор је:

$$F(x) = x^2 + c \ \ (mod\ N), \text{ за неку константу } c.$$

Ово можда делује изненађујуће, јер се анализа низа дефинисаног рекурзивним изразима ради под претпоставком да је F случајно пресликавање, док $x^2 + c$ то није. Међутим, Полардов ρ алгоритам се често користи и ради веома добро. Дакле, упркос очигледној неусклађености, овај избор за F је добар. Делимично објашњење дато је у раду [3]. Прецизније, ту се напомиње да је граф функције $F(x) = x^2 + c$ веома специфичан, сваки чвор сем једног (са вредношћу c) има или 0 или 2 претходника. Као последица тога, анализирани су графови бинарних случајних пресликавања, где сваки чвор има улазни степен 0 или 2. Дошло се до закључка да су мултипликативне константе, које се појављују у статистици графова претходног поглавља, погођене овом променом, али да су редови њихових величина непромењени.

Када је за F изабран полином лако се може проверити да за низ дефинисан рекурзивним изразом $X_{i+1} = F(X_i)\ \ (mod\ N)$, редукција по модулу p, за било који фактор p броја N, даје исте рекурзивне изразе (по модулу p уместо по модулу N). За прост делилац p броја N, нека $X^{(p)}$ означава низ $X\ (mod\ p)$. Тада $X^{(p)}$ задовољава:

$$X_{i+1}^{(p)} = F(X_i^{(p)}) \quad (mod\ p).$$

Користећи уобичајену анализу ових низова, очекује се да $X^{(p)}$ има циклус са параметрима s и l реда $\sqrt{p}$. У том случају, Флојдов и Брентов алгоритам проналазе циклус у времену $O(\sqrt{p})$, под претпоставком да се може вршити испитивање постојања колизије у оквиру низа $X^{(p)}$. С тим у вези, довољно је приметити да је:

$$X_i^{(p)} = X_j^{(p)} \Leftrightarrow NZD(X_i^{(p)} - X_j^{(p)},\ N) \neq 1.$$

Дакле, постојање колизије може се ефикасно испитати коришћењем израчунавања НЗД. Штавише, када се деси колизија по модулу p не постоји посебан разлог да постоје колизије по модулу других фактора броја N, тј. оне могу, али не морају да постоје. Према томе, када се деси колизија обично се може израчунати и одговарајући фактор броја N, израчунавањем НЗД разлике два члана низа који чине колизију и броја N. Полардов ρ алгоритам, који користи Брентов алгоритам за проналажење колизије представљен је алгоритмом 5.

```
Улаз: Број за факторизацију N, параметар c, max број итерација M
x ← 0
y ← x
trap ← 0
nexttrap ← 1
for i from 1 to M do
  x ← x^2 + c (mod N)
  f ← NZD(x − y, N)
  if f ≠ 1 then
    if f = N then
      print 'Неуспех: Колизија по модулу N'
      exit
    else
      print 'Факторизација завршена. Пронађени фактор броја N је f'
      exit
  if i = nexttrap then
    trap ← nexttrap
    trap ← 2 · trap
    y ← x
print 'Неуспех: Фактор броја N није пронађен унутар M итерација'
```

Алгоритам 5. Полардов ρ алгоритам факторизације

Наравно, Полардов ρ метод се може користити и са Флојдовим алгоритмом. У оба случаја, скуп део методе је то што свако испитивање једнакости захтева израчунавање НЗД. У пракси, број израчунавања НЗД може се умањити. Идеја је да се помноже неколико узастопних вредности облика $X_i^{(p)} - X_j^{(p)}$ и да се затим израчуна НЗД резултата и броја N. Ако је овај НЗД једнак 1, ниједна једнакост није испуњена. Ако је НЗД једнак одговарајућем фактору, факторизација је завршена. Међутим, ако је НЗД једнак броју N онда се мора вратити уназад и поново се морају извршити испитивања једнакости, једно по једно, надајући се одговарајућем фактору броја N. Разлог који стоји иза потребе за враћањем уназад јесте то што се груписањем неколико испитивања једнакости заједно повећава вероватноћа постојања симултане колизије за различите факторе броја N.

Полардов ρ алгоритам је веома занимљива примена алгоритама за проналажење циклуса којег карактерише неколико специфичних детаља. Један од њих је то што код Полардове ρ методе нема потребе рачунати параметре s и l низа $X^{(p)}$. Уместо тога, довољно је пронаћи колизију користећи две вредности $X_i^{(p)}$ и $X_j^{(p)}$ које могу бити произвољно одабране. Према томе, не мора се користити алгоритам 3 за проналажење почетка циклуса. Интересантно питање које се односи на Полардов ρ алгоритам јесте: Да ли се може осмислити варијација овог алгоритма која је базирана на алгоритму заснованом на рођенданском парадоксу? Другим речима, имајући две листе бројева L_1 и L_2 упоредиве величине и цео број N, да ли се ефикасно могу открити два броја, x_1 из L_1 и x_2

из L_2, таква да је $NZD(x_2 - x_1,\ N) \neq 1$? Ово је тежак проблем који је у свом раду [4] решио Монтгомери (*Peter Montgomery*) као алат за побољшавање факторизације помоћу елиптичких кривих.

4.2. Полардов ρ алгоритам за решавање проблема дискретног логаритма

Као и многи алгоритми факторизације и Полардов ρ алгоритам може бити прилагођен за израчунавање дискретних логаритама у произвољној цикличној групи. Нека је G мултипликативна група простог реда p, g генератор групе G и h произвољан елемент из G. Као што речено, решавање проблема дискретног логаритма одговара проналажењу целог броја α таквог да је $h = g^{\alpha}$. У циљу примене алгоритма за проналажење циклуса, потребно је изабрати случајно пресликавање F из скупа функција над G. За конструкцију функције F прво се G дели на три дисјунктна подскупа G_1, G_2 и G_3, апроксимативно исте величине уз пожељан услов да 1 није у G_1. Даље, F се дефинише на следећи начин:

$$F(x) = \begin{cases} x^2 & ako\ x \in G_1, \\ gx & ako\ x \in G_2, \\ hx & ako\ x \in G_3. \end{cases}$$

Након овога, дефинише се и рекурзиван низ X, стављајући $X_0 = 1$. Ако 1 припада G_1 онда је 1 фиксна тачка функције F, па тада мора бити изабрана друга почетна тачка. То је разлог зашто је пожељно да 1 не припада G_1. Као и иначе, након $O(\sqrt{p})$ корака очекује се колизија $X_i = X_j$ и детекција циклуса у F. Сама по себи колизија није довољна за израчунавање α, па се морају добити и додатне информације о низу X.

Са циљем да се добију додатне информације о низу X, следећим пресликавањем дефинише се функција $\phi: H = \mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/p\mathbb{Z} \to G$:

$$(a, b) \to g^a h^b.$$

Ова функција је сурјекција на G и сваки елемент из G има p различитих тачака у H које ће се сликати у њега. Штавише, α се може израчунати ако се пронађу две тачке из H са истом сликом у G. Заиста, ако је $\phi(a, b) = \phi(a', b')$, онда је $g^{a+\alpha b} = h^{a'+\alpha b'}$ и $\alpha = \frac{a'-a}{b'-b}\ (mod\ p)$.

Нека је низ Y дефинисан условом да за његов i-ти члан Y_i важи $X_i = \phi(Y_i)$. Следећи корак је померање низа X у H и проналажење низа Y. Ово може бити обављено коришћењем следеће функције $G: H \to H$:

$$G(a, b) = \begin{cases} (2a, ab) & ako\ \phi(a, b) \in G_1, \\ (a + 1, b) & ako\ \phi(a, b) \in G_2, \\ (a, b + 1) & ako\ \phi(a, b) \in G_3. \end{cases}$$

Лако је проверити да за све елементе (a, b) из H важи да је $\phi(G(a, b)) = F(\phi(a, b))$. Према томе, низ Y дефинисан са $Y_0 = (0, 0)$ и $Y_{i+1} = G(Y_i)$, задовољава $X = \phi(Y)$. Наравно, Y је низ над много већим скупом него X, те се не очекује постојање колизију у низу Y већ у низу X. Дакле, колизија враћена за низ X алгоритмом за проналажење циклуса не мора да буде колизија за низ Y. На овај начин добијају се два различита елемента који се сликају у исти елемент из G и може се израчунати α.

4.3. Полардови кенгури

Поред ρ метода, Полард је предложио и други алгоритам за израчунавање дискретног логаритма, када су познате и још неке додатне информације. Прецизније, он разматра проблем $g^x = h$ у G и жели да одреди x на основу g и h, ако је још познато да x припада неком интервалу $[a, b]$. Коришћење Полардове ρ методе у групи G без наведеног услова није баш добра идеја, посебно у случају када је $b - a$ мање него квадратни корен из величине групе G. Тада је много ефикасније добити x грубом силом, покушавајући све вредности из $[a, b]$. Да би се изборио са овим проблемом Полард предлаже коришћење онога што назива кенгури. Овај метод је такође познат и као Полардов ламбда алгоритам.

Кенгур у Полардовом методу је низ елемената у G чији се логаритми увећавају узастопним скоковима. За примену ове метода потребна су два кенгура, један питом чија је почетна тачка g^a и један дивљи са почетном тачком h. За првог се каже да је питом јер се логаритам одговарајућег члана низа увек зна. За прецизно дефинисање ових кенгура потребна је функција скока j која сваком елементу из G додељује позитиван број до горње границе $\sqrt{(b-a)}$. Уобичајен начин је да се G издели на k подскупова $S_0, \dots, S_{k-1}$, $k = \lfloor \log_2 b - a/2 \rfloor$, отприлике једнаких величина и да се за свако x из S_i дефинише $j(x) = 2^i$. Пошто је дефинисана функција скока j, дефинише се функција $F(x) = xg^{j(x)}$.

Питом кенгур је низ дефинисан са:

$$T_0 = g^a$$
$$T_{i+1} = F(T_i).$$

Дивљи кенгур је низ дефинисан са:

$$W_0 = h$$
$$W_{i+1} = F(W_i).$$

Док се израчунава скок питомог кенгура, важно је пратити дискретан логаритам броја T_i, тј. вредности $a + \sum_{k=0}^{i-1} j(T_k)$. Чим дискретан логаритам пређе b, завршава се са низом T. Нека је T_n последњи израчунати члан низа и нека је l_n одговарајући дискретан логаритам, тј. $T_n = g^{l_n}$. Када је позната ова крајња тачка, покреће се дивљи кенгур и одржава се дужина његових скокова. Са

дивљим кенгуром завршава се или када се стигне до T_n или када дужина скокова постане већа од $b - a$. У другом случају алгоритам не успева. Иначе, пронађена је позиција W_i са одговарајућом сумом s_i, тако да је $W_i = T_n$. Како је $W_i = hg^{s_i}$ и $T_n = g^{l_n}$ следи да је:

$$h = g^{l_n - s_i}.$$

Као и Полардов ρ метод, и метод Полардови кенгури је генерички алгоритам. Овај алгоритам има добру вероватноћу успеха и сложеност $O(\sqrt{b-a})$ за израчунавање дискретног логаритма броја h. За ригорозније анализе видети [10].

5. Директна криптографска примена у контексту сигурности блоковских шифри

Пре преласка на тему и значај овог поглавља, у наставку су дати основни појмови криптографије и криптоанализе који се у овом поглављу користе, а који су преузети из текста [14].

Отворени текст. Отворени текст је порука коју треба послати.

Шифрат. Шифрат је шифрована порука.

Шифровање. Шифровање је трансформација отвореног текста у шифрат.

Дешифровање. Дешифровање је инверзна трансформација шифрата у отворени текст.

Проточна шифра. Проточна шифра трансформише отворени текст симбол по симбол, односно најчешће бит по бит.

Блоковска шифра. Блоковска шифра примењује се на симболе отвореног текста груписане у блокове.

Шифарски систем (или само систем). Шифарски систем је пар чији су елементи алгоритам шифровања и алгоритам дешифровања. Ови алгоритми скоро увек зависе од посебног параметра који се назива кључ. Кључ је параметар којим се бира конкретна шифарска трансформација у оквиру изабраног система.

Напад са познавањем само шифрата. Нападач је пресрео шифрат, али нема одговарајући отворени текст. Уобичајена је претпоставка да нападач има приступ шифрату. Могуће су две ситуације:

- Нападач зна алгоритам шифровања, али не зна кључ. Ово важи за већину система у комерцијалној употреби.
- Нападач не зна алгоритам шифровања. Опрезни корисници се никада не ослањају на претпоставку да ће ова ситуација потрајати дуго.

Напад са познатим отвореним текстом. Нападач има неке парове шифрат, отворени текст. Нападач поред тога може имати још шифрата.

Напад са познавањем изабраног отвореног текста. Претпоставља се да нападач може изабрати отворени текст који жели да пропусти кроз процес шифровања. Иако је овакав напад мало реалан, он има теоријски значај, јер ако се зна довољно отвореног текста, онда постају употребљиве технике за напад са изабраним отвореним текстом.

У овом поглављу најпре се разматрају неки од уобичајених начина коришћења, тј. режима рада блоковских шифри. Битна карактеристика ових режима рада је то што блоковске шифре користе као црне кутије. Стога, конкретан режим рада се може користити за доста различитих блоковских шифри, те је корисно проучавати сигурност датог режима независно од изабране блоковске шифре.

У наставку поглавља представљени су неки од резултата истраживања проблема сигурности блоковских шифри дати у радовима [8] и [9]. У овим радовима показано је да постоји граница изнад које се губи сигурност одређених режима рада блоковских шифри. Под датим условима, изнад ове границе, могуће је извести одређене нападе на неке од режима рада блоковских шифри. Ови напади представљени су у наставку поглавља.

На крају поглавља представљен је генерички метод за одређивање кључа блоковске шифре, познат као Хелманов временско-меморијски компромис. Упркос чињеници да овај напад користи велику количину меморије, он је присутан у овом раду из два разлога. Први разлог је то што је понашање Хелмановог компромиса уско повезано са анализом случајних пресликавања. Други разлог је то што је Хелманов компромис веома користан за одређивање кључа блоковске шифре након напада који су представљени у овом поглављу.

5.1. Начини коришћења блоковских шифри

Отворени текст P који је потребно шифровати може да буде веома дугачак. Тада се он мора поделити на више блокова. Уколико је дужина блока l, отворени текст P се дели на блокове од по l бита. Ако дужина поруке није умножак броја l, онда се порука допуњава. Тада се природно јављају питања као што су: Да ли користити нови кључ за сваки блок? Да ли би требало шифровати сваки блок независно или шифровање учинити зависним од шифровања претходних блокова? Одговоре на ова питања дају различити начини коришћења блоковских шифри. Начини коришћења, односно режими рада блоковских шифри описују начине на које се блоковска шифра примењује на низ узастопних блокова како би се сигурно пренела количина података већа од једног блока. У овом раду изложена су три уобичајена режима рада: режим електронске кодне књиге ECB (*Electronic Codebook*), режим уланчавања блокова шифрата CBC (*Cipher Block Chaining*) и режим бројача CRT (*Counter Mode*).

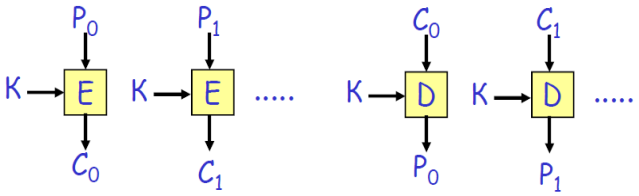
Нека p_i означава блок отвореног текста $P = p_0 p_1 \dots p_m \dots$, а c_i одговарајући блок шифрата. Поред тога нека $E_k(\cdot)$ означава примену блоковске шифре, тј. операцију шифровања, а $D_k(\cdot) = E_k^{-1}(\cdot)$ операцију дешифровања, при чему је k примењени кључ.

ECB

Сваки блок се шифрује независно.

Шифровање: $c_i = E_k(p_i)$

Дешифровање: $p_i = D_k(c_i)$



ECB је најједноставнији режим рада блоковске шифре. Углавном се користи за размену кратких порука, нпр. за размену кључева за шифровање. Његов главни недостатак је то што за исти кључ два иста отворена текста дају исте шифрате. Према томе, за дуге поруке, сигурност ECB режима је смањена. Уколико у отвореном тексту постоје блокови који се често понављају (заглавља, титуле, ...), то је могуће открити анализом шифрата, иако се отворени текст не зна.

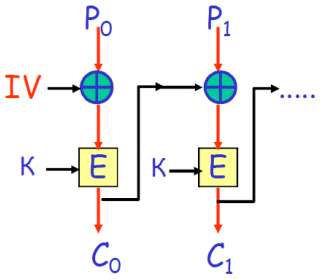
CBC

Блокови се уланчавају.

Потребан је иницијализациони вектор IV.

Шифровање: $c_i = E_k(c_{i-1} \oplus p_i)$, где је $c_{-1} = \text{IV}$

Дешифровање: $p_i = c_{i-1} \oplus D_k(c_i)$



CBC је најчешће коришћен режим рада и он је много сигурнији од ECB режима. За исти кључ и различите иницијализационе векторе два иста отворена текста дају различите шифрате. Сигурност овог режима разматра се у наставку овог поглавља.

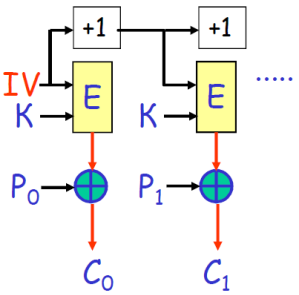
CRT

Понаша се као проточна шифра.

Потребан је иницијализациони вектор IV.

Шифровање: $c_i = p_i \oplus E_k(T_i)$, где је $T_i = T_{i-1} + 1$ и $T_{-1} = \text{IV}$

Дешифровање: $p_i = c_i \oplus E_k(\text{T}_i)$



CRT је популаран режим рада и има примену код контроле приступа. Овај режим блоковску шифру користи као да је у питању проточна шифра. *IV* се шифрује, а потом се врши *XOR* операција шифроване вредност и блока отвореног текста. За сваки наредни блок отвореног текста вредност *IV* се увећава за 1.

5.2. Сигурност блоковских шифри

За квалитет шифровања постоје два основна критеријума, сигурност и ефикасност. Сигурност се мери отпорношћу шифровања на све познате нападе. Ефикасност је комбинација брзине

шифровања/дешифровања и мере у којој алгоритам користи ресурсе (потребан простор на чипу за хардверску реализацију, односно потребна меморија за софтверску реализацију).

Сигурност блоковских шифри може се разматрати у односу на начин на који се одговарајући шифрат шаље. С једне стране блоковско шифровање се може посматрати као процес у којем се одговарајући шифрат шаље у целини. Прецизније, уређај за шифровање прихвата отворени текст у целини, дели га на блокове и блок по блок укључује у процес шифровања, док одговарајуће блокове шифрата интерно памти. Тек када је последњи блок шифрата доступан, тј. када је одговарајући шифрат комплетиран, уређај за шифровање шаље овај шифрат у целини. Имплементација овог приступа у пракси може бити проблематична, јер уређај за шифровање мора бити у могућности да интерно прихвати и смести целу поруку која може бити врло дугачка. Из овог разлога се често практикује употреба блоковског шифровања као процеса у којем се одговарајући шифрат шаље блок по блок. Дакле, уређај за шифровање прихвата отворени текст блок по блок, сваки блок укључује у процес шифровања и одговарајући шифрат одмах шаље.

Сигурност блоковских шифри када се шифрат шаље блок по блок је један аспект сигурности режима рада блоковских шифри који је откривен независно од стране две групе истраживача. У раду [9] је показано да имплементација алгоритма *secure shell* (*SSH*), који се заснива на режиму CBC, није сигурна у односу на напад са познавањем изабраног отвореног текста. У раду [8] је представљена теоријска идеја сигурности блоковских шифри када се одговарајући шифрат шаље блок по блок. У истом раду показано је да за неколико уобичајених режима рада блоковских шифри, постоји веза између сигурности блоковских шифри када се шифрат шаље блок по блок и одговарајућих блоковских шифри када се шифрат шаље у целини. Након овог открића неколико радова проучавало је сигурност блоковских шифри када се шифрат шаље блок по блок и откривени су сигурни режими рада. Као главни резултат, дошло се до закључка да постоје сигурносни нивои режима рада блоковских шифри, када се шифрат шаље блок по блок, који су у суштини еквивалентни уобичајеним сигурносним нивоима режима рада блоковских шифри када се шифрат шаље у целини. Прецизније, показано је да су режими рада сигурни све до границе рођенданског парадокса након чега се ова сигурност губи. Под границом рођенданског парадокса подразумева се најмањи број блокова поруке који је потребан да би се са великом вероватноћом догодила колизија међу блоковима шифрата. Као што је већ познато, за l-битни блок ова граница је $O(2^{l/2})$. Према томе, у циљу да добије колизију између два блока шифрата, нападач мора укључити $O(2^{l/2})$ блокова отвореног текста у процес шифровања. У том смислу, ако се може доказати да нападачу треба више блокова, рецимо $O(2^{2l/3})$ или $O(2^{5l/6})$, каже се да је обезбеђена сигурност изнад границе рођенданског парадокса.

5.2.1. Сигурност блоковског шифровања у режиму CBC

Због поменутог проблема са имплементацијом блоковског шифровања као процеса у којем се шифрат шаље у целини, блоковско шифровање у режиму CBC се чешће користи као процес у којем се одговарајући шифрат шаље блок по блок. Међутим, као што је примећено у радовима [8] и [9], овакво блоковско шифровање у режиму CBC није сигурно у односу на напад са изабраним блоком отвореног текста. Прецизније, на основу тренутног блока шифрата c_i, а пре укључивања блока отвореног текста p_{i+1} у процес шифровања, нападач бира произвољну вредност v као наредни улаз за E_k, тј. он рачуна $p_{i+1} = v \oplus c_i$ и овај блок укључује у процес шифровања као наредни блок отвореног текста. Према томе, уређај за шифровање укључује задати блок $p_{i+1} = v \oplus c_i$ у процес шифровања и као наредни блок шифрата добија се вредност $c_{i+1} = E_k(c_i \oplus p_{i+1}) = E_k(c_i \oplus v \oplus c_i) = E_k(v)$.Ово својство може се искористити на неколико начина да би се показало да блоковско шифровање у режиму CBC није сигурно када се шифрат шаље блок по блок. На пример, може се креирати једноставнан статистички раздвајач (енг. *distinguisher*) који је у стању да направи поруку чији се CBC шифрат може статистички разликовати од случајне поруке исте дужине. Да би се то постигло довољно је осигурати да се у режиму CBC иста вредност v шифрује два пута. У том случају, у CBC шифрату појављују се два идентична блока, док за случајну поруку то не мора да важи.

За описани проблем блоковског шифровања у режиму CBC, када се шифрат шаље блок по блок, постоји веома једноставно решење. Назива се одложени CBC, а сигурност је доказана у раду [11]. Основна идеја одложеног CBC режима је следећа: Када уређај за шифровање израчуна блок шифрата c_i, уместо да га одмах пошаље, он чека на наредни блок отвореног текста p_{i+1} и тек по пријему овог блока шаље c_i. Последњи шифрован блок уређај за шифровање шаље по пријему специјалног блока који говори да је порука завршена. Овај специјални блок има само ту улогу, он није шифрован. Одложени CBC онемогућује нападача да контролише улаз за блоковску шифру. Према томе, одложени CBC спречава једноставан напад са изабраним блоком отвореног текста.

5.3. CBC режим рада и рођендански напад

У овом поглављу разматра се сигурност блоковског шифровања, када се шифрат шаље у целини, у режиму CBC изнад границе рођенданског парадокса. Прецизније, изнад границе рођенданског парадокса очекивано је да се међу блоковима шифрата може пронаћи бар једна колизија, односно $c_i = c_j$. Ако се у овој једнакости сваки од блокова шифрата замени својим изразом, добија се да је:

$$E_k(c_{i-1} \oplus p_i) = E_k(c_{j-1} \oplus p_j).$$

Како је E_k пермутација (инвертибило пресликавање), важи да је:

$$c_{i-1} \oplus p_i = c_{j-1} \oplus p_j.$$

Ово се другачије може записати као $p_i \oplus p_j = c_{i-1} \oplus c_{j-1}$. Дакле, свака пронађена колизија даје информацију о резултату XOR операције одговарајућих блокова отвореног текста. Како дужина поруке расте, може се пронаћи све више колизија и према томе открити све више информација о поруци отвореног текста.

Да би криптоаналитичар искористио овај напад, он мора бити у стању да ефикасно детектује колизије између блокова шифрата. Једна од најчешћих техника јесте да се сортирају блокови шифрата. Међутим, ово захтева или велики простор у оперативној меморији или брзу секундарну меморију. За 64-битне блоковске шифре криптоаналитичар мора да сачува и сортира око 2^{32} блокова, односно 2^{35} бајтова. Заправо, ситуација је мало сложенија. Док сортира, криптоаналитичар мора да прати и почетне позиције блокова да би знао који блокови отвореног текста су укључени у једначину која је изведена из колизије.

Уколико криптоаналитичар нема довољно меморије за сортирање табеле која садржи и вредности шифрата и оригиналне позиције, други начин је да се несортиране вредности држе негде у секундарној меморији. Тада се сортирају блокови шифрата без индекса. Када је заједничка вредност позната, изврши се претрага и пронађу се позиције блокова који се поклапају. Чак и за овај приступ потребно је 2^{35} бајтова брзе меморије. Упркос брзом напредовању рачунарског хардвера, 32 GB је и даље велика количина меморије. Према томе, у пракси, чак и приликом коришћења шифровања у режиму CBC са 64-битном блоковском шифром, нпр. троструки DES, извођење напада захтева велики напор криптоаналитичара. Поред осталих разлога и ово је један због којег се на CBC шифровање са троструким DES-ом и даље често наилази.

5.4. Одложени CBC режим рада и рођенданski напад

У овом поглављу разматра се сигурност блоковског шифровања, када се шифрат шаље блок по блок, у одложеном CBC режиму изнад границе рођенданског парадокса. Да би се конструисао напад, користи се трик сличан коришћеном за напад на блоковску шифру у режиму CBC, када се шифрат шаље блок по блок. Прецизније, када год је доступан блок шифрата, у функцији од овог блока одређује се следећи блок отвореног текста и укључује се у процес шифровања. Услед закашњења од једног блока, напад може почети тек од другог блока отвореног текста. Као први блок отвореног текста, у процес шифровања се може укључити било која вредност по избору нападача, нпр. блок који се састоји само од нула.

Почевши од другог блока, блок отвореног текста се бира према $p_i = p_{i-1} \oplus c_{i-2}$. Чак и у одложеном CBC режиму, обе вредности p_{i-1} и c_{i-2} су ефективно познате у тренутку када би p_i требало да се укључи у процес шифровања. Сада, посебним избором блокова отвореног текста, вредности шифрата се одређују на основу следеће једнакости:

$$c_i = E_k(c_{i-1} \oplus p_{i-1} \oplus c_{i-2})$$
$$= E_k\big(E_k(p_{i-1} \oplus c_{i-2}) \oplus (p_{i-1} \oplus c_{i-2})\big).$$

Другим речима, ако се са Z_i означи улаз блоковске шифре у i-тој рунди, тј. $Z_i = p_i \oplus c_{i-1}$, онда Z_i задовољава:

$$Z_i = E_k(Z_{i-1}) \oplus Z_{i-1}.$$

Сасвим занимљиво, XOR улаза и излаза блоковске шифре је стандардна конструкција која се користи за креирање псеудо-случајних функција из псеудо-случајних пермутација, позната као Дејвис-Мејер (*Davies-Meyer*) конструкција. Ова конструкција је често коришћена у хеш функцијама, нпр. среће се је код *SHA-1*.

Ако се функција F дефинише са $F(x) = E_k(x) \oplus x$, добија се да је $Z_i = F(Z_{i-1})$. Према анализи из поглавља 3, познато је да низ Z креира циклус после $O(2^{l/2})$ корака. Штавише, за откривање циклуса у обзир се могу узети раније представљени алгоритми за проналажење циклуса.

5.4.1. Флојдов алгоритам

За коришћење Флојдовог алгоритма низови Z_i и $W_i = Z_{2i}$ се морају паралелно израчунавати у циљу проналажења једнакости $Y_i = W_i$. Одмах се може приметити да је ово проблематично у контексту блоковског шифровања у одложеном CBC режиму, када се шифрат шаље блок по блок. Заиста, нападач не контролише почетне вредности c_0 коришћене током шифровања, па не постоји начин да израчуна вредности оба низа Z_i и W_i у исто време. Наравно, увек се може чувати Z и поредити Z_i и Z_{2i} у сваком кораку, али ово захтева доста меморије што је баш оно што би требало избећи. Ипак, треба имати у виду да је овај приступ бољи од сортирања и да може радити са спором меморијом.

5.4.2. Брентов алгоритам

Са друге стране, одмах се примећује да се циклус може лако детектовати коришћењем Брентовог алгоритма. Према објашњењу из поглавља 2.2, када год је индекс i степен броја два ($i = 2^t$) памти се вредност Z_{2^t} и упоређује се са вредностима подниза Z_i, све до следећег степена броја два. Ово се може обавити без потребе за израчунавањем вредности истог низа два пута или за контролисањем иницијалних вредности.

У неком смислу, откривање циклуса је већ напад. Заиста, ово се може посматрати као статистички раздвајач који је у стању да открије да ли је задати низ добијен на овај начин или је случајан. Ако је излаз прави тј. представља излаз блоковског шифровања режима CBC, циклус се са великом вероватноћом открива у времену $O(2^{l/2})$. Ако је излаз случајан, Брентов алгоритам скоро никада не проналази нову вредност једнаку сачуваној вредности. Међутим, напад се може појачати. Прецизније, циклус се може искористити да се добије шифрат $E_k(v)$, неке вредности v

по избору нападача. Да би се то постигло, када се чува улаз Z_{2^t} блоковске шифре, потребно је сачувати и одговарајући излаз c_{2^t}. Због закашњења од једног блока, излаз c_{2^t} није доступан одмах, али се чува када пристигне. Када је колизија детектована добија се нови улаз Z_i једнак Z_{2^t}. Наравно, ова једнакост између улаза у блоковску шифру имплицира једнакост између излаза. Дакле, унапред се зна да је $c_i = c_{2^t}$. Захваљујући овоме, може се рећи да је $p_{i+1} = c_i \oplus v$ и према томе добити шифрована вредност $E_k(v)$. Према томе, захваљујући Брентовом алгоритму блоковско шифровање, када се шифрат шаље блок по блок, у одложеном CBC режиму изнад рођенданске границе подложније је нападима него блоковско шифровање, када се шифрат шаље у целини, у обичном CBC режиму.

5.4.3. Нивашов алгоритам

Нивашов алгоритам може бити врло користан за представљен напад. Он дозвољава да се смањи количина потребних шифрата пре него што се циклус детектује. Заправо, користећи алгоритам 4, као што је представљено у поглављу 2, гарантовано је да је потребно највише $s + 2l$ блокова шифрата. Наравно, изазов је смањити овај број. Јасно, не може се користити мање од $s + l$ блокова, с обзиром да се прва колизија јавља на позицији $s + l$. Следећи Ниваша у раду [2] могуће је осмислити варијанту алгоритма 4 која користи додатну меморију и детектује циклус са $s + (1 + \alpha)l$ блокова, за произвољно мало α. Овај алгоритам користи технику која се назива партиционисање и меморијски захтеви овог алгоритма расту када се α смањује.

5.5. CRT режим рада и рођендански напад

Сигурност шифровања у режиму CRT је такође интересантан пример. Заправо, он се чини потпуно имуним на рођендански напад. Заиста, како је псеудо-случајан низ добијен шифровањем различитих вредности колизија се не може појавити. Прецизније, кренувши од почетне вредности која се у сваком кораку повећава за један, није могуће добити ту вредност поново све док се не прођу све могуће вредности b-битног блока. Како постоји 2^b различитих вредности, ово је далеко изнад границе рођенданског парадокса.

Међутим, када се у обзир узме шифровање више порука све постаје компликованије. Пре свега мора се избећи коришћење истог бројача вредности између порука. Једноставно решење за то може бити меморисање тренутне вредности након шифровања, па коришћење те вредности за следећу поруку. Међутим, ово решење захтева уређај за шифровање са меморијом, а у неким случајевима то није могуће. У таквим ситуацијама алтернатива која се често користи јесте да се одабере случајна почетна вредност за сваку поруку. Са овим решењем колизије између почетних вредности воде до напада и рођендански парадокс се враћа у игру.

Оно што је изненађујуће јесте да како се напади са изабраним отвореним текстом одвијају, чак ни памћење бројача није довољно да би се избегао рођендански парадокс. Нека је нападач послао на шифровање две дуге случајне поруке M_0 и M_1, једнаке дужине L (мерено у блоковима). Он добија шифрат C, поруке M_0 или поруке M_1. Тада он рачуна $C \oplus M_0$ и та вредност представља првог кандидата за псеудо-случајни проток коришћен током шифровања. Затим, као другог кандидата рачуна $C \oplus M_1$. Према конструкцији, не може доћи до колизије између блокова унутар правог кандидата. Међутим, ништа не спречава случајне колизије да се појаве у оном другом. Као последица, ако нападач открије колизију у једном од два псеудо-случајна кандидата, он сигурно зна која од порука, M_0 или M_1, је шифрована и то може објавити. У супротном, може бацити новчић и најавити одговарајућу претпоставку. Јасно, ако вероватноћа колизије у погрешном псеудо-случајном кандидату није занемарљива, нападач добија незанемарљиву предност.

Занимљиво је напоменути да могућност да се одреди прави кандидат заправо није заснована на својству самог CRT режима, већ на чињеници да је он конструисан над пермутацијом. Ако се блоковска шифра или случајна пермутација замене (псеудо) случајним пресликавањем ова могућност нестаје. Међутим, то не би уклонило потребу за меморисањем вредности бројача. Штавише, у пракси се режими рада блоковских шифри који су засновани на псеудо-случајном пресликавању ретко срећу. Главни разлог је вероватно то што су блоковске шифре стандардизоване и самим тим лако доступне програмерима.

5.6. Хелманов временско-меморијски компромис

Хелманов временско-меморијски компромис је генерички метод за одређивање кључа K блоковске шифре E, под претпоставком да су познати шифрати произвољних отворених текстова. Овај метод претпоставља да криптоаналитичар може унапред да изврши огромна израчунавања везана за E и да запамти неке од резултата који у каснијем тренутку могу убрзати одређивање кључа K. Циљ је да се постигне нетривијалан временско-меморијски компромис за каснију фазу напада. Две тривијалне могућности за одређивање кључа K су: да се не меморише ништа и да се користи груба сила, или да се меморише табела која садржи све парове $(E_K(0^n), K)$ и да се у табели тражи $E_K(0^n)$, шифрат познате поруке.

Хелманов алгоритам базиран је на анализи функције $F(K) = E_K(0^n)$ која пресликава скуп свих могућих кључева у скуп свих могућих кључева. Треба имати у виду да дату функцију треба прилагодити у случају када се величина кључа и величина блока разликују. Ако је величина кључа мања него величина блока резултат се мора скратити. Ако је величина кључа већа од величине блока надовезује се неколико шифровања (рецимо одабиром константних блокова, $0^n, 0^{n-1}1, \ldots$) и скраћује се надовезано ако је потребно. Разбијање блоковске шифре E и проналажење кључа K је еквивалентно израчунавању инверза функције F у тачки $E_K(0^n)$. Према томе, Хелманов

компромис се обично сматра за напад са познавањем изабраног отвореног текста. Заправо он је и нешто бољи јер не инсистира да се изабере 0^n. Једини захтев је осигурати да постоји фиксирана вредност v за коју је лако добити шифровање $E_K(v)$. На пример, ако нападач зна да пошиљалац обично шифрује поруку са неким фиксираним заглављем и претпостављајући детерминистички режим шифровања, напади са познавањем изабраног отвореног текста нису неопходни.

Може се приметити да Хелманов алгоритам показује зашто је важно коришћење режима рада који користе случајне иницијализационе векторе. Прецизније, код оваквих режима, нападач не може добити шифровање изабране фиксиране вредности, те се Хелманов метод не може искористити.

5.6.1. Поједностављен случај

Да би се разјаснила кључна идеја која стоји иза Хелмановог алгоритма прво се разматра посебан случај и претпоставља се да је F циклична пермутација. У пракси ово никада није случај, али је под овом претпоставком анализа доста једноставнија. Израчунавање низа кључева почиње од иницијалног кључа K_0 коришћењем израза $K_{i+1} = F(K_i)$. Овај низ прати циклус функције F и иде редом кроз све могуће кључеве за блоковску шифру, враћајући се у вредност K_0 после 2^k итерација, где је k број битова кључа. На сваких 2^l корака, где $l = \lfloor k/2 \rfloor$, меморише се пар $(K_{j\cdot 2^l}, K_{(j-1)\cdot 2^l})$. Другим речима, памти се тренутни кључ заједно са његовим предаком који се налази 2^l корака испред. Када је сакупљено свих 2^{k-l} могућих парова дуж циклуса, они се сортирају по вредности и чува се тако сортиран низ.

Када је низ парова сортиран, F се лако може инвертовати у 2^l корака. Низ се претражује полазећи од вредности $E_K(0^n)$, за непознат кључ K. Ако та вредност није присутна даље се трага за вредношћу $F(E_K(0^n))$. Након t корака проналази се вредност $F^{(t)}(E_K(0^n))$. За ову вредност чита се други члан пронађеног пара. Према томе, иде се 2^l корака уназад и над добијеном вредношћу се F примењује $2^l - t - 1$ пута. Тада се добија (јединствена) вредност која се слика у $E_K(0^n)$, тј кључ K.

У поједностављеном случају Хелманов компромис меморише 2^{k-l}, тј. 2^l или 2^{l+1}, за изабрано l, парова и време извршавања фазе напада (изостављајући фазу претходних израчунавања) је $l \cdot 2^l$, где је урачунато и време потребно за бинарне претраге над сортираним низом.

5.6.2. Општи случај

У општем случају F није циклична пермутација. Штавише, сада се очекује да се F понаша као случајно пресликавање. Дакле, итерације функције F за случајно изабрану почетну тачку врло брзо доводе до циклуса величине $O\left(\sqrt{2^k}\right)$. Према томе, да би се покрило више од занемарљивог дела кључева, мора се користити доста почетних тачака. Још једна потешкоћа је то што за

неколико почетних тачака ништа не спречава два различита низа да се преклопе у неком тренутку, односно да имају заједнички део. Заправо, било које две почетне тачке у гигантској компоненти функционалног графа воде до преклапања низова. Ако не пре, низови се најкасније преклапају негде у циклусу одговарајуће гигантске компоненте. Што је још већи проблем, вредност која се слика у $E_K(0^n)$ није јединствена, па проналажење такве вредности није довољно за одређивање кључа K. У циљу заобилажења ових проблема Хелманов компромис захтева више времена и више меморије него у поједностављеном случају. Уобичајена равнотежа када су у питању и време и меморија (занемарујући логаритамски фактор) је $2^{2k/3}$, тј. потребна количина меморије и број корака су $2^{2k/3}$.

При уобичајеној анализи Хелмановог компромиса претпоставља се да су низови, који се израчунавају за m случајних почетних тачака, дужине t. За сваки низ почетне и крајње тачке се чувају сортиране по вредностима крајњих тачака. Да би се пронашле вредности које се функцијом F пресликавају у дате вредности користи се иста идеја као и у поједностављеном случају. Од почетне тачке понавља се примена функције F све док се не дође до запамћене крајње тачке. Онда се алгоритам враћа на одговарајућу почетну тачку и наставља извршавање. Могуће је неколико случајева. У најбољем случају открива се K, а у осталим случајевима алгоритам не успева. До неуспеха долази или тако што се проналази погрешна вредност која се пресликава у $E_K(0^n)$, односно различит кључ који даје исти шифрат, или тако што се уопште не долази до тачке $E_K(0^n)$. Вероватноћа успеха је реда $mt/2^k$, докле год је mt^2 мање од 2^k (видети [1]). Изнад ове границе број колизија између ланаца постаје веома велики и компромис постаје неисплатив. За $m = t = 2^{k/3}$ вероватноћа успеха је $2^{-k/3}$, меморијски захтеви су $O(2^{k/3})$ кључева, а време извршавања је $O(k2^{k/3})$. Да би се побољшала вероватноћа, Хелман предлаже да се користи $2^{k/3}$ различитих избора за функцију F, правећи тако мале промене у начину на који се из шифрата $E_K(0^n)$ добија кључ K. Претпостављајући независност између табела направљених за сваку од ових функција, вероватноћа успеха се побољшава за константан удео простора кључева, са временским и меморијским захтевима $2^{2k/3}$ (занемаривајући логаритамски фактор).

6. Примена у проналажењу колизије код хеш функција

Алгоритми за проналажење циклуса још једну примену проналазе у потрази за колизијом код хеш функција, посебно када је дужина њиховог излаза релативно кратка. На пример, за дату функцију са 128-битним излазом захваљујући рођенданском парадоксу може се пронаћи колизија са отприлике 2^{64} израчунавања хеш функције. Међутим, јасно је да са тренутним рачунарима није изводљиво сачувати и сортирати одговарајућу листу од 2^{64} хеш вредности у циљу откривања колизије. Прво решење је да се једноставно позове алгоритам за проналажење циклуса над рекурзивним низом, где је свака вредност добијена хеширањем претходне вредности. И даље, ово решење није задовољавајуће из више разлога. Први разлог је то што се на овај начин могу добити само колизије које нису од користи, тј. могу се добити две случајне поруке са истим хешом. Наравно, за криптоаналитичара би било пожељније да добије две смислене поруке са различитим значењем и истим хешом. Други разлог је то што су алгоритми за проналажење циклуса, представљени у поглављу 2, суштински секвенцијални. И док је обављање 2^{64} операције изводљиво са данашњим рачунарима, извођење 2^{64} узастопне операције на једном рачунару је потпуно друга ствар.

У овом поглављу најпре је дат кратак увид у то шта су криптографске хеш функције и који су њихови сигурносни захтеви. Представљен је колизиони напад на хеш функције и показано је како се технике за проналажење циклуса могу искористити за проналажење колизија између смислених порука. Након тога показано је како се у комбинацији са паралелним израчунавањима Нивашов алгоритам може искористити у циљу проналажења колизија.

6.1. Криптографске хеш функције

Хеш функције су функције које врше трансформацију поруке произвољне дужине у израз фиксне дужине. Оне представљају изузетно важну класу функција и моћан алат који је у широкој употреби у модерној криптографији. Главни циљ у криптографији је задовољити три основне карактеристике сигурности информација: поверљивост, интегритет и аутентикација. Поверљивост значи да само прималац треба бити у стању да прочита послату поруку. Интегритет је особина поруке да није доживела промене садржаја од тренутка када је послата до тренутка када стигне до примаоца. Очување интегритета поруке једна је од најважнијих примена криптографских хеш функција. Рачунањем хеша (вредност хеш функције) пре и после преноса поруке преко мреже можемо утврдити да ли је порука промењена. Аутентикација представља процес у оквиру којег се за корисника или извор информација утврђује да је заиста оно за шта се представља, другим речима аутентикација је процес утврђивања идентитета.

Иако су хеш функције у широкој употреби, испоставља се да није свака хеш функција погодна за коришћење у криптографији. Таква хеш функција мора поседовати одређене сигурносне особине. Оне хеш функције које задовољавају ове особине називамо криптографским хеш функцијама. Криптографске хеш функције би требало да се понашају као случајна пресликавања, али да остану детерминистичке и лако израчунљиве. Ако се може доћи до поруке полазећи од њеног хеша или уколико се могу пронаћи две поруке са истим хешом, криптографска хеш функција сматра се несигурном.

Сигурност данашње комуникације се заснива на криптографским протоколима, где мноштво таквих протокола користи хеш функције као градивне блокове. Стварање криптографских хеш функција са јаким сигурносним карактеристикама је важан и истовремено тежак задатак. Од мноштва понуђених, само су ретке преживеле пробу времена и показале задовољавајуће особине.

6.1.1. Отпорност хеш функција на колизију

Колизија за функцију H је пар различитих улаза x и x' таквих да је $H(x) = H(x')$. У овом случају кажемо да је пар x, x' у колизији. Функција H је отпорна на колизију ако се не зна алгоритам чија је сложеност испод сложености рођенданског напада, а који може пронаћи колизију, тј. пронаћи било који пар x, x' такав да је $H(x) = H(x')$. Овај рад се бави функцијама H које имају бесконачне домене (нпр. као улаз узимају стрингове свих могућих дужина) и коначне кодомене. Према Дирихлеовом принципу, у таквим случајевима колизије морају постојати. Тада се тежи томе да проналажење колизија буде веома тешко. У вези са функцијама за које је и улазни и излазни скуп вредности коначан, мисли се само на оне функције које компримују свој улаз, тј. дужина излаза је краћа од дужине улаза. Уколико се ово не захтева отпорност на колизију је тривијално постићи. На пример, функција идентитета је тривијално отпорна на колизију. Отпорност на колизију је јак сигурносни захтев и доста га је тешко постићи. Међутим, у неким применама хеш функција довољно је ослонити се на блаже захтеве.

Приликом разматрања криптографских хеш функција обично се у обзир узимају три нивоа сигурности:

Отпорност на колизију. Ово је најјачи захтев и онај који је до сада разматран.

Слаба отпорност на колизију. Неформално говорећи, хеш функција је слабо отпорна на колизију ако је за дато x практично немогуће да се пронађе $x' \neq x$, такво да је $H(x') = H(x)$.

Једносмерност. Хеш функција треба да буде једносмерна функција, тј. да је за дато y практично немогуће да се пронађе вредност x таква да је $H(x) = y$.

Свака хеш функција која је отпорна на колизију је такође и слабо отпорна на колизију. Ово важи јер за дато x противник може пронаћи $x' \neq x$ за које је $H(x') = H(x)$, а онда може пронаћи пар x, x' који је у колизији. Исто тако, свака хеш функција која је слабо отпорна на колизију је

такође једносмерна. Ово важи због чињенице да ако је могуће инвертовати y и пронаћи x' такво да је $H(x') = y$, онда је могуће узети дати улаз x, израчунати $y = H(x)$, а затим инвертовати y и добити x'. Тада је са великом вероватноћом $x' \neq x$ (ослањајући се на чињеницу да H компримује улаз и да се доста различитих улаза пресликава у исти излаз). Може се приметити да наведена три сигурносна захтева формирају хијерархију, свака дефиниција имплицира ону која је испод ње.

6.2. Рођендански напад - проналажење колизије између смислених порука

Генерички рођендански напад проналази колизију у било којој хеш функцији, мада у времену које је експоненцијално у односу на дужину излазног хеша. Нека је дата хеш функција $H: \{0,1\}^* \rightarrow \{0,1\}^l$. Због једноставности се разматрају само оне хеш функције чији је улаз произвољне дужине. Напад ради на следећи начин: Изабере се q произвољних различитих улаза $x_1, \ldots, x_q$, израчуна се $y_i := H(x_i)$ и изврши се провера да ли је било која од две вредности y_i једнака. Као што је већ познато, за излазну дужину од l бита рођендански напад проналази колизију са великом вероватноћом помоћу $q = O\left(2^{l/2}\right)$ израчунавања хеш функције. Сортирањем излаза колизија се може пронаћи у времену $O\left(l \cdot 2^{l/2}\right)$.

Иста идеја напада може се искористити за проналажење "смислених" колизија. Проналажење колизије између смислених порука је први пут представљено у раду [12]. Идеја је да се почне одабиром две поруке M и M' са различитим значењем. Даље, у овим порукама је потребно пронаћи број места на којима се могу применити сигурне модификације. Сигурне модификације су све промене које неће променити значење поруке. Обично постоји велики број сигурних модификација у порукама које су читљиве за човека. Можемо заменити размак новим редом, на неким позицијама у поруци интерпункцијски знаци могу бити додати или уклоњени, велика и мала слова могу бити замењена без губитка значења поруке. Такође, неке речи могу бити замењене својим синонимима, остављајући смисао поруке непромењен.

Претпоставимо да нападач Алиса жели да пронађе две поруке M и M' такве да је $H(M) = H(M')$. Прва порука M нека је писмо од њеног послодавца које објашњава зашто је отпуштена са посла, а друга порука M' нека је ласкаво писмо препоруке. Уколико постоји t места за сигурне модификације поруке M, Алиса може направити скуп од 2^t различитих порука истог значења. Размотримо следећи пример:

Тешко/Немогуће је *поверовати/замислити* да ћемо *пронаћи/запослити* још једног *радника/запосленог/особу* која има сличну *способност/вештину/стручност* као Алиса. Урадила је *добар/одличан/сјајан/изванредан* посао.

Било која комбинација искошених речи је могућа и носи исту поруку. Стога се дата реченица може написати на $2 * 2 * 2 * 3 * 3 * 4 = 288$ различитих начина. Ово је само једна реченица, те је заправо веома лако изгенерисати поруку која се може преписати на 2^{64} различита начина. Све што је потребно је 64 речи са једним синонимом.

Може се приметити да су за хеш функцију Алисиних 2^t порука потпуно различите и да се добија 2^t случајних, тражених хеш вредности. На исти начин добија се 2^t различитих копија поруке M'. Јасно, за n-битну хеш функцију, за $t > n/2$ очекује се постојање колизије између копије поруке M и копије поруке M'. Алиса може извршити напад рођенданског типа генерисањем $O\left(2^{n/2}\right)$ порука првог типа и $O\left(2^{n/2}\right)$ порука другог типа, а затим потражити колизије између порука ова два типа.

Генерички рођендански напад захтева велику количину меморије. Конкретно, он захтева од нападача да сачува свих q вредности $\{y_i\}$, јер нападач не зна унапред који ће пар дати колизију. Побољшани рођендански напад, који користи технике алгоритама за проналажење циклуса, је напад са драстично смањеним меморијским захтевима. Заправо, он има сличну временску сложеност и вероватноћу успеха као генерички напад, али користи константну величину меморије. Напад почиње одабиром случајне вредности x_0, а затим се за $i = 1,2, \dots$ рачунају вредности $x_i = H(x_{i-1})$ и $x_{2i} = H(H(x_{2(i-1)}))$. Приметимо да је $x_i = H^i(x_0)$, где H^i означава i узастопних примена функције H. У сваком кораку се упоређују вредности x_i и x_{2i}. Ако су оне једнаке онда постоји колизија негде у низу $x_0, \dots, x_{2i-1}$ и алгоритам тада улази у процес проналажења колизије. Оно што је кључно за овај напад јесте да он захтева чување само две хеш вредности у свакој итерацији.

Алгоритам који је управо описан можда се не чини погодним за проналажење смислених колизија јер нема контролу над разматраним елементима. Ипак, могуће је искористити га. Као и раније, нека Алиса жели да пронађе колизију између порука два различита типа, писмо које објашњава зашто је Алиса отпуштена и писмо препоруке, где се оба хеширају у исту вредност. Идеја сигурних модификација може бити искоришћена и у конјункцији са техником тражења циклуса. Међутим, потребно је водити рачуна о неким додатним стварима како би се избегле нежељене колизије између две копије поруке M или две копије поруке M'. Алиса пише сваку од порука тако да у њој постоји t речи које могу бити замењене. Имајући t тачака сигурне модификације и t-битни број i, нека је са M_i означена копија поруке M добијена када се одређене вредности сваке тачке модификације поставе према вредности одговарајућих битова у бинарном запису броја i. Слично, са M_i' означимо i-ту копију поруке M'. Користећи ове ознаке, може се увести функција F, из $t + 1$ у n битова, на следећи начин:

$$F(v, i) = \begin{cases} M_i & ako\ v = 0, \\ M_i' & ako\ v = 1. \end{cases}$$

Размотримо реченице:

0: Алиса је *добар/вредан* и *искрен/одан радник/запослени*.

1: Алиса је *тежак/проблематичан* и *напоран/иритантан радник/запослени*.

Функција F узима 4-битне улазе, где први бит одређује тип излазне реченице, а наредна три бита одређују избор речи у тој реченици. На пример:

$F(0{,}000) =$ Алиса је *добар* и *искрен радник.*

$F(0{,}001) =$ Алиса је *тежак* и *напоран радник*.

$F(1{,}010) =$ Алиса је *вредан* и *искрен запослени*.

$F(1{,}011) =$ Алиса је *проблематичан* и *напоран запослени*.

Скраћивањем улаза са n битова на $t + 1$ може се добити низ хеш вредности од смислених порука и тражити колизија. Међутим, овај директан приступ са $t \approx n/2$, често доводи до тривијалне колизије. Заиста, ако се две хеш вредности поклопе после скраћивања на $t + 1$, у следећој итерацији хешира се порука која је претходно наишла и улази се у циклус са колизијом између хеш вредности два пута исте поруке. Да би се избегао овај случај потребно је више тачака сигурних модификација, наиме требало би изабрати $t = n - 1$. Са овим избором, улаз у циклус одговара првој колизији. Штавише, са вероватноћом $1/2$ ова прва колизија је између копије поруке M и копије поруке M'. Ако није, једноставно се може поново покренути алгоритам са другом почетном тачком.

6.3. Паралелно тражење колизије помоћу Нивашовог алгоритма

У циљу проналажења колизије користећи паралелна израчунавања мора се имати другачији приступ јер су алгоритми за проналажење циклуса суштински секвенцијални. С тим у вези не може се тражити циклус унутар једног рекурзивно дефинисаног низа. Уместо тога требало би се ослонити на неколико независних низова. Алгоритам овог типа је већ представљен у овом раду. То је алгоритам Полардови кенгури, за израчунавање дискретног логаритма на задатом интервалу (поглавље 4.3). Међутим, за паралелизацију се овај приступ мора адаптирати.

За извршавање на паралелном рачунару најважније је да се смањи количина комуникације и синхронизације између паралелних процеса. Уобичајен приступ, када се тражи одређена вредност или одређено својство међу великим скупом података, је да се користи велики број рачунара за креирање могућих кандидата и да се онда ови кандидати пријаве на наменски сервер који ће одрадити крајњу проверу. За проналажење колизије, требало би пријавити вредности за које је врло могуће да ће се преклапати. Ово се може постићи коришћењем технике истакнутих тачака из рада [6]. Треба имати у виду да у алгоритмима истакнутих тачака један чвор паралелног рачунара има посебну улогу. Овај чвор прихвата истакнуте тачке од свих осталих чворова и проверава да ли постоје колизије унутар тог скупа коришћењем рођенданског напада базираног

на сортирању. Уколико постоје колизије, овај чвор их користи за израчунавање стварних вредности. Веома је важно осмислити паралелан алгоритам за проналажење колизије на начин који минимизује цену комуникације, меморијске захтеве и много израчунавања сервера. Даље у тексту представљен је један од могућих начина за паралелно тражење колизије над случајним пресликавањем F.

Нека случајно пресликавање F оперише над скупом од N елемената и нека се паралелан рачунар састоји од P идентичних паралелних чворова и једног посебног чвора - сервера. Такође, потребан је и скуп од D истакнутих тачака. Да би алгоритам био погодан, истакнуте тачке морају бити лаке за препознавање и лаке за генерисање. Типичан пример таквог скупа истакнутих тачака је скуп од n-битних целих бројева са d водећих нула. Скуп се састоји од $D = 2^{n-d}$ елемената, у већим групама кардиналности $N = 2^n$. Свака истакнута тачка x се лако препознаје, својством: $0 \leq x \leq 2^{n-d}$. Штавише, са оваквом дефиницијом истакнутих тачака једноставно је генерисати случајне истакнуте тачке.

Постављањем ових услова алгоритам ради на следећи начин: Сервер додељује (посебну) истакнуту тачку сваком чвору за израчунавање. Када добије истакнуту тачку, чвор за израчунавање је користи као почетну тачку за Нивашов алгоритам (алгоритам 4) са малом изменом. Наиме, када Нивашов алгоритам наиђе на истакнуту тачку он се зауставља и шаље извештај. Извештај садржи три вредности: почетну истакнуту тачку, крајњу тачку и дужину путање између две тачке. Сервер одржава листу извештаја и детектује колизију између крајњих тачака. Ако се током израчунавања вредности функције F деси колизија, то сигурно значи да је присутан или циклус на једном чвору за израчунавање или да је присутна колизија између два различита чвора. Обе могућности су детектоване описаним алгоритмом. У првом случају Нивашов алгоритам детектује циклус, а у другом случају сервер открива колизију између крајњих тачака. Прва колизија се лако добија из откривене колизије. Заиста, претпоставимо да постоји путања дужине L_1 од истакнуте тачке S_1 до крајње тачке P и друга путања дужине L_2 од S_2 до P. Без губитка општости, може се још претпоставити да је $L_1 \leq L_2$. Онда, полазећи од S_1 сервер пролази прву путању кроз тачно $L_1 - L_2$ корака. Из те тачке је довољно да прође обе путање паралелно, све док се не пронађе колизија. Треба имати у виду да је то увек прва колизија. Заиста, ако је E прва истакнута тачка на коју се наилази идући ка S_1, S_2 не може бити на првој путањи. Према томе, два паралелна пута почињу из различитих тачака и завршавају се на истој позицији. Штавише, они имају једнаку дужину и на обе путање се колизија сигурно јавља после истог броја корака.

Као и код секвенцијалног алгоритма за проналажење циклуса, када је пронађена било која од колизија преостаје да се израчуна $O(\sqrt{N})$ вредности. Према томе, са P процесора убрзање паралелизације је у основи оптимално. Треба напоменути да израчунате вредности нису укључене у глобално бројање све док одговарајуће истакнуте тачке нису пријављене. Према томе, број истакнутих тачака треба да буде адекватно изабран. Са D истакнутих тачака, удео

истакнутих тачака износи N/D. Према томе, просечна дужина путање између две истакнуте тачке је N/D. Треба имати у виду да се за неке почетне тачке може ући у циклус. У том случају се не може лепо дефинисати удаљеност између две истакнуте тачке. Ако се занемаре техничке потешкоће, закључујемо да покретање описаног алгоритма на P процесора дозвољава израчунавање PN/D тачака. Са циљем да F узима $O(\sqrt{N})$ вредности, потребно је изабрати $D = O(\sqrt{N})$. Како сервер мора да чува P путања, тј. једну по чвору за израчунавање, количина потребне меморије је $O(P)$. Време извршавања кода који издваја колизију на F од колизије између истакнутих тачака је одозго ограничено временом потребним да се прође кроз две путање, тј. $O(N/D) = O(\sqrt{N}/P)$.

Треба имати у виду да за горње објашњење и анализу, израчунавање није савршено уравнотежено између процесора. Заиста, неке путање између истакнутих тачака су краће него друге. Да би се побољшала та неуравнотеженост, једно једноставно унапређење је да се серијски покрену неколико инстанци Нивашовог алгоритма на сваком чвору за израчунавање, са циљем да се добије путања просечне дужине. Ово повећава параметар P и као последица потребна је меморија на серверу. Друга примедба је то што када P постаје велико, колизије које долазе из циклуса унутар једне путање постају веома ретке. Према томе, Нивашов алгоритам би требало заменити са неким једноставнијим, који се извршава од истакнуте тачке до следеће, са могућношћу заустављања када му превише времена треба да пронађе крајњу тачку. Једина мана овог приступа је што анализа понашања алгоритма постаје много компликованија.

6.3.1. Проналажење већег броја колизија

Један интересантан аспект наведеног алгоритма за паралелно тражење колизије је то што он може бити коришћен за ефикасно проналажење великог броја колизија. Треба имати у виду да секвенцијални алгоритам за проналажење циклуса није баш ефикасан за проналажење великог броја колизија. Заиста, за проналажење k колизија, k пута се мора поновити алгоритам за проналажење циклуса, са укупном ценом $O(k\sqrt{N})$. Насупрот томе, са горе наведеним паралелним алгоритмом, рођендански парадокс наставља да се користи и да проналази прву колизију. Као последица тога, укупна цена за конструкцију k колизија се смањује на $O(\sqrt{kN})$. Међутим, мора се направити опрезан избор за скуп истакнутих тачака. Јасно је да свака сачувана путања може дати највише једну колизију. Према томе, параметар P требало би да буде већи него што је број жељених колизија k.

Једна алтернатива би могла да буде да се као почетне тачке, уместо истакнутих, користе произвољне тачке. Међутим, овај приступ уводи нов недостатак. Заиста, ништа не би спречило добијање исте путање два пута или прецизније, две путање где је једна подпутања од друге. Ове две путање не би довеле до нове колизије и повећале би цену израчунавања без икаквих добитака.

7. Програмска реализација и резултати

Изложена тврђења из поглавља 3 о општим, локалним и екстремалним својствима случајних пресликавања $\varphi: \{1, \dots, n\} \to \{1, \dots, n\}$ су асимптотки резултати који дају асимптотску процену за различите параметре када n тежи бесконачности. Циљ овог поглавља је спровођење тестова о структури функционалног графа датих функција, којима би се проверило да ли постоје значајна одступања од приближних вредности параметара представљених у раду [3]. У овом поглављу представљен је програм који је коришћен за експерименталну проверу асимптотских резултата параметара случајних пресликавања и приказани су резултати спроведеног тестирања.

7.1. Програмска реализација

За програмску реализацију коришћен је програмски језик *Python* и пакет *NetworkX*. *NetworkX* је *Python* пакет за креирање, манипулацију и проучавање структуре, динамике и функција комплексних мрежа. Пакет *NetworkX* не пружа директу подршку за рад са функционалним графовима, али је коришћена класа *DiGraph* овог пакета, као помоћ при реализацији класе *FunGraf*. Класа *DiGraph* је класа која описује усмерене графове. Она чува чворове и усмерене гране произвољног типа података.

Коришћене функције пакета *NetworkX* које оперишу над графовима су:

`find_cycle`(*G*, *source=None*, *orientation='original'*) - За задати граф *G* функција враћа листу грана графа које формирају циклус. Други аргумент је опциони и може бити задат као чвор од којег обилазак графа почиње. Трећи аргумент је такође опциони и односи се само на усмерене графове; може бити задат као индикација да се при обиласку графа оријентација грана не мора поштовати.

`ancestors`(*G*, *source*) - За задати усмерени граф *G* и чвор *source* функција враћа скуп свих претходника датог чвора у графу.

`shortest_path_length`(*G*, *source=None*, *target=None*, *weight=None*) - За задати граф *G* функција враћа најкраће путање у графу. Други аргумент је опциони и може се задати као чвор од којег се рачунају најкраће путање до свих осталих чворова у графу. Трећи аргумент је такође опциони и може се задати као чвор до којег се рачунају најкраће путање од свих осталих чворова у графу. Уколико су оба аргумента задата функција враћа најкраћу путању између датих чворова у графу. Четврти аргумент је опциони и односи се на тежину грана, подразумевана вредност је *None* што значи да је тежина сваке гране 1.

`number_connected_components`(*G*) - За задати неусмерени граф *G* функција враћа број повезаних компоненти у графу.

`connected_component_subgraphs`(*G*, *copy=True*) - За задати неусмерени граф *G* функција враћа генератор који се може искористити за креирање новог графа на основу сваке од повезаних компоненти

графа G. Други аргумет је опциони и подразумевана вредност је *True*, што значи да ће се копирати атрибути задатог графа G.

Класа *FunGraf* реализована је као омотач класа класе *DiGraph*. На тај начин омогућено је да се искористе специфичности функционалних графова, нпр. чињеница да свака повезана компонента функционалног графа садржи један циклус. Објекат класе *FunGraf* се креира позивом конструктора FunGraf(f, n), где је f случајно пресликавање, а n величина улазног скупа, тј. број чворова функционалног графа. При креирању објекта ове класе, за свако x из $\{1, \dots, n\}$ и $y = f(x)$ одређује се усмерена грана (x, y). Добијена листа усмерених грана користи се за креирање атрибута G (објекат класе *DiGraph*). Затим се позивају методе класе *FunGraf* које рачунају општа и локална својства. Програмска реализација ових метода изложена је у прилогу рада.

У групу општих својстава спадају: број повезаних компоненти, број листова, број унутрашњих чворова и број цикличних чворова датог функционалног графа. У групу локалних својстава спадају: просечна дужина путање, просечна дужина циклуса, просечна дужина репа, просечна величина повезане компоненте, просечна величина стабла и просечан број претходника у датом функционалном графу. Екстремална својства доступна су након израчунавања одређених локалних својстава. То се постиже памћењем максималних вредности локалних параметара. Након што су ова својства израчуната, сваки од параметара доступан је кроз одређену *property* методу, заједно са временом које је било потребно за израчунавање траженог параметра.

7.2. Приказ резултата

У овом одељку приказани су резултати спроведене анализе општих, локалних и екстремалних својстава случајних пресликавања. Тестирање је заснивано на анализи структуре функционалног графа ових функција.

Тестирање је спроведено за десет случајних пресликавања облика $ax^2 + bx + c \ (mod\ n) + 1$, где је n први прост број већи или једнак од 2^k, за $k = 5, \dots, 20$. Коефицијенти a, b и c су случајни прости бројеви из интервала $\{1, \dots, 2^{20}\}$.

Графички приказ добијених резултата општих, локалних и екстремалних својстава тестиране случајне функције $39x^2 + 257x + 11 \ (mod\ n) + 1$, за све вредности параметра n, дат је графицима 1, 2 и 3, респективно. За сваку групу својстава приказан је график зависности вредности одређених параметара од величине улазног скупа n. За локалне параметре представљени су и тзв. *boxplot* графици (график 4). На свим графицима, и вредности добијених параметара и вредности броја n скалиране су логаритамски.

Добијени резултати, свих тестираних функција, су потврдили очекиване приближне вредности параметара случајних пресликавања и показали да нема значајнијих одступања, посебно за веће n.

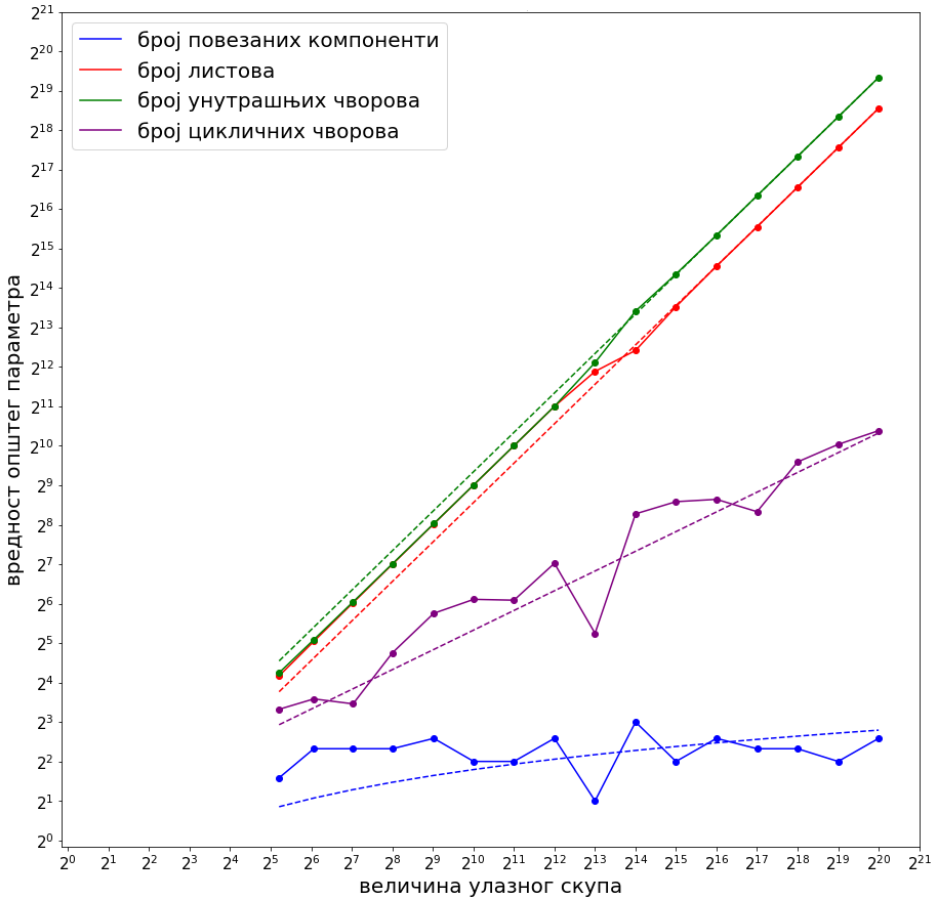


График 1: Општа својства

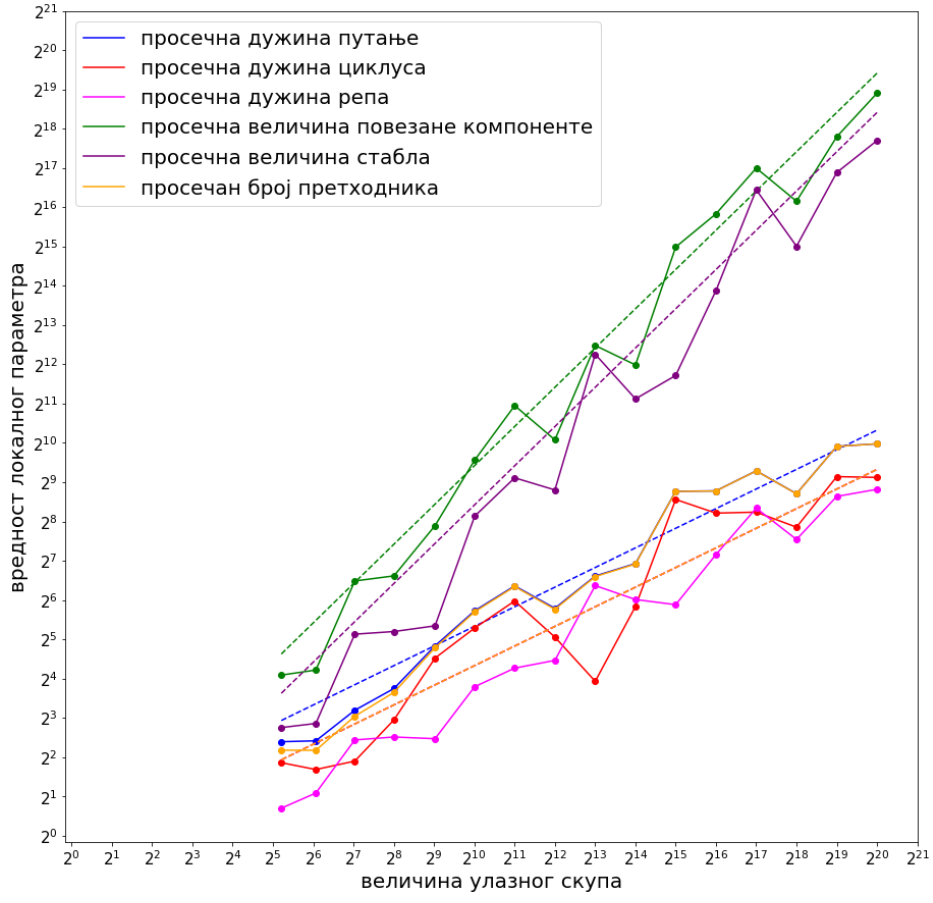


График 2: Локална својства

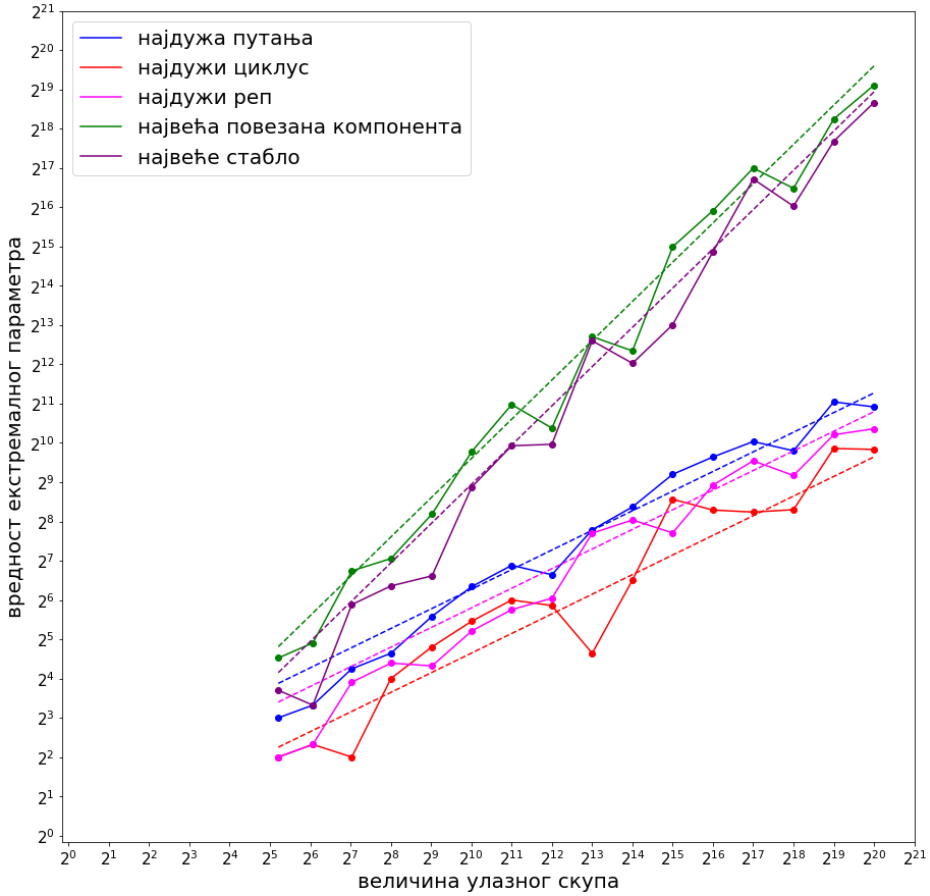


График 3: Екстремална својства

Да би се одредила просечна вредност локалног параметра из случајне почетне тачке, параметар се рачуна за сваку од могућих почетних тачака, тј. за сваку тачку скупа $\{1, \dots, n\}$. На тај начин добијен је скуп чији су елементи вредности локалног параметра за различите почетне тачке. На графику 4 представљена је расподела вредности локалних параметара тестиране функције $39x^2 + 257x + 11 \ (mod\ n) + 1$, за $n = 1048583$. Дати графици указују на степен распрострањености и асиметричности међу подацима.

Нека је $\{x_1, \dots, x_n\}$ скуп сортираних вредности локалног параметра добијеног за сваку од почетних тачака. Сваки од графика се састоји од правоугаоника у оквиру којег су приказани подаци од доњег квартила $q_L = x_i$, за $i = \left(\frac{n+1}{4}\right)$, до горњег квартила $q_U = x_i$, за $i = \left(\frac{3(n+1)}{4}\right)$. Линија која дели правоугаоник означава медијану. Вертикалне линије на графику су тзв. *whisker* линије и представљају најмању, односно највећу вредност унутар $1.5 * (q_L - q_U)$ посматрајући од доњег, односно горњег квартила. Све тачке које су ван ових граница приказане су засебно и сматрају се вредностима које одступају од осталих.

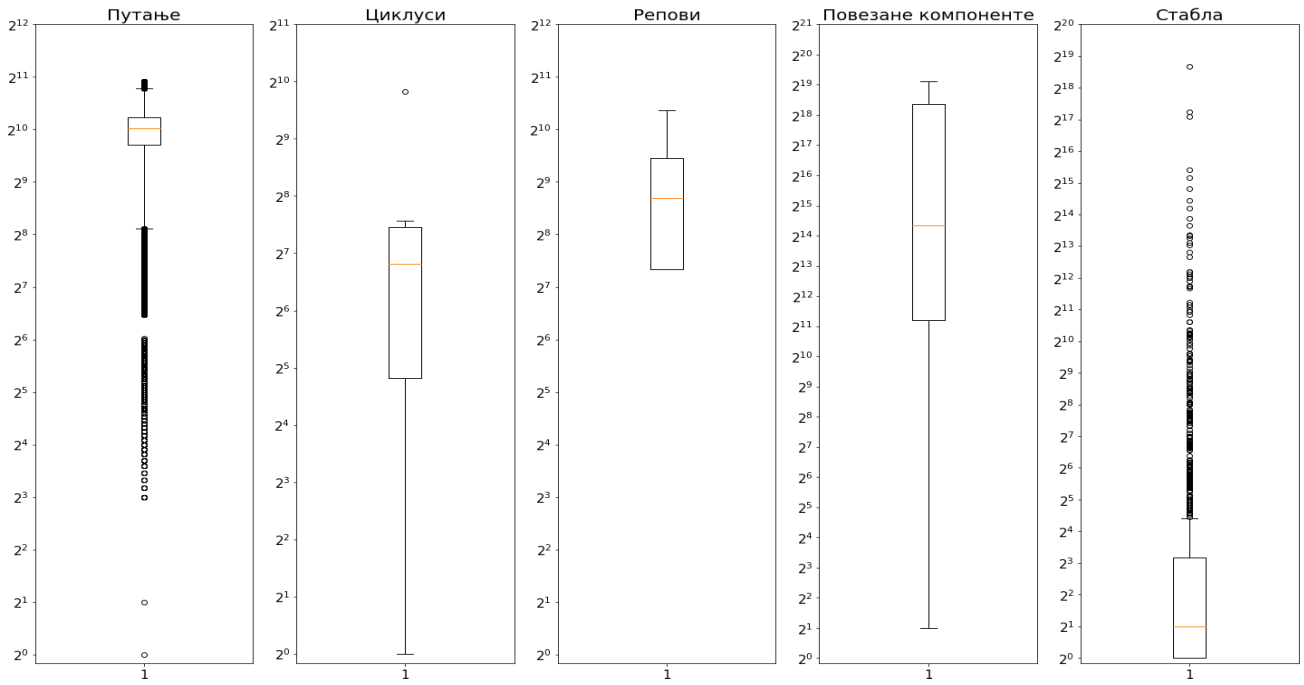


График 4: Расподела локалних параметара

График 5 даје приказ времена извршавања реализованих метода за одређивање параметара функционалног графа функције $39x^2 + 257x + 11 \ (mod\ n) + 1$, у зависности од величине улазног скупа n.

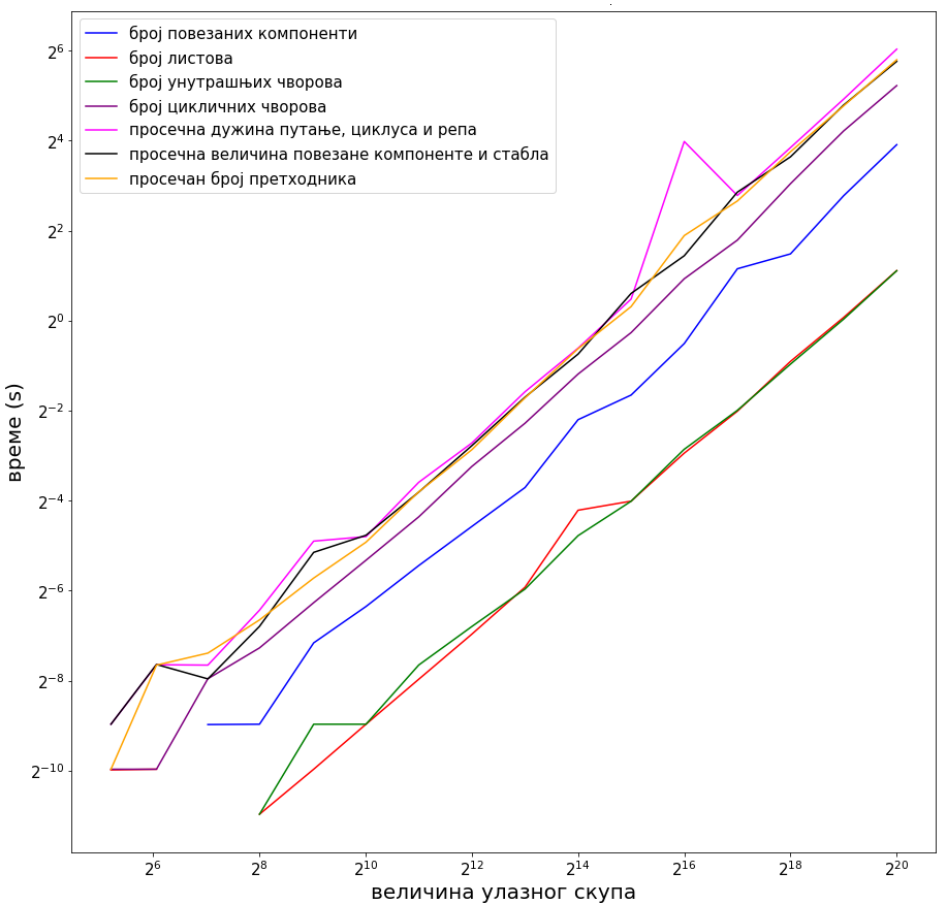


График 5: Време извршавања

8. Закључак

Анализом алгоритама за проналажење циклуса показано је да колизиони напади, базирани на овим алгоритмима, користе знатно мању количину меморије за чување података него уобичајени генерички напади. Основни циљ напада је проналажење колизије над низом објеката који су генерисани итерацијама неке фиксиране функције, према рекурзивном изразу $X_{i+1} = F(X_i), i \geq 0$. У циљу теоријског анализирања понашања оваквих низова, одговарајућа функција F се моделује као случајно пресликавање. У поглављу 7 изложени су резултати анализе општих, локалних и ексремалних својстава случајних пресликавања. Овом анализом потврђени су одговарајући асимптотски резултати дати у раду [3].

Показано је да представљени алгоритми за проналажење циклуса налазе примену у теорији бројева и да се могу користити за решавање проблема дискретног логаритма и факторизације. Још једна примена ових алгоритама илустрована је у вези са моделом сигурности блоковских шифри. Рођенданским нападима који су представљени илустрована је важност рођенданског парадокса у криптографији и показано је да је он кључ за разумевање сигурности многих криптографских протокола. Показано је да се при блоковском шифровању у одложеном CBC режиму изнад рођенданске границе, када се шифрат шаље блок по блок, коришћењем Брентовог алгоритма може извести напад са изабраним отвореним текстом, који ради много ефикасније него одговарајући напад на блоковско шифровање у обичном CBC режиму, када се шифрат шаље у целини. Као трећа област примене, представљен је колизиони напад на хеш функције и показано је како се технике за проналажење циклуса могу искористити за проналажење колизија између смислених порука. Показано је и да се у комбинацији са паралелним израчунавањима Нивашов алгоритам може искористити у циљу проналажења колизија.

У циљу даљег истраживања примене рођенданског парадокса било би интересантно анализирати проналажење колизија у општијој ситуацији, када наредни изабрани објекат није функција претходног, тј. када се технике из овог рада не могу применити. Криптоаналитички напади засновани на рођенданском парадоксу су у пракси често ограничени количином меморије коју захтевају, те се даље могу проучавати специфични случајеви напада за које се технике овог рада не могу применити, али који ипак користе мање меморије од генеричких метода. Ови алгоритми познати су као рођендански напади кроз квадрисекције и базирани су на подели почетног проблема на четири дела, уместо на два као код уобичајених рођенданских напада. Ове технике смањују потребну количину меморије без значајног повећања времена извршавања.

Литература

[1] A. Joux, *Algorithmic Cryptanalysis*, CRC Press, 2009.

[2] G. Nivasch, *Cycle detection using a stack*, Information Processing Letter, стр. 135-140, 2004.

[3] P. Flajolet, A. M. Odlyzko, *Random mapping statistics*, у J-J. Quisquater, J. Vandewalle, Advances in Cryptology - EUROCRYPT '89, том 434 књиге Lecture Notes in Computer Science, стр. 329-354, Springer-Verlag, април, 1989.

[4] P. L. Montgomery, *A FFT Extension of the Elliptic Curve Method of Factorization*, PhD thesis, University of California, Los Angeles, 1992.

[5] R. P. Brent, J. M. Pollard, *Factorization of the eight Fermat number*, Mathematics of Computation, том 36, стр. 627-360, април 1981.

[6] J-J. Quisquater, J-P. Delescaille, *How easy is collision search? New results and applications to DES*, у G. Brassard, Advances in Cryptology – CRYPTO '89, том 435 књиге Lecture Notes in Computer Science, стр. 408-413, Springer-Verlag , август 1990.

[7] D. E. Knuth, *The Art of Computer Programming*: *Sorting and Searching*, 3. издање, Addison Wesley, 1973.

[8] A. Joux, G. Martinet, F. Valette, *Blockwise-adaptive attackers: Revisiting the (in)security of some provably secure encryption models: CBC, GEM, IACBC*, у M. Yung, Advances in Cryptology - CRYPTO 2002, том 2442 књиге Lecture Notes in Computer Science, стр. 17-30, Springer-Verlag, август 2002.

[9] M. Bellare, T. Kohno, C. Namprempre, *Authenticated Encryption in SSH: Provably Fixing the SSH Binary Packet Protocol,* ACM Transactions on Information and System Security, том 7, стр. 206-241, мај 2004.

[10] R. Montenegro, P. Tetali, *How long does it take to catch a wild kangaroo?*, у M. Mitzenmacher, 41st ACM Symposium on Theory of Computing, стр 553-559, ACM Press, мај 2009.

[11] P-A. Fouque, G. Martinet, G. Poupard*, Practical symmetric on-line encryption*, у T. Johansson, Advances in Cryptology - FSE 2003, том 2887 књиге Lecture Notes in Computer Science, стр. 362-375, Springer-Verlag, фебруар 2003.

[12] G. Yuval, *How to swindle Rabin*, Cryptologia, том 3, стр. 187-189, 1979.

[13] J. M. Pollard, *A Monte Carlo method for factorization*, BIT Numerical Mathematics, том 15, стр. 331-334, 1975.

[14] М. Живковић, *Криптографија*, скрипта, 2017.

9. Прилог

```
# broj povezanih komponenti
def __os_broj_povezanih_komponenti(self):
    # vreme pocetak
    pocetno_vreme = time.time()
    self.__broj_povezanih_komponenti = nx.number_connected_components(self.__G.to_undirected())
    # vreme kraj
    self.__broj_povezanih_komponenti_vreme = round(time.time() - pocetno_vreme, 5)

# broj listova
def __os_broj_listova(self):
    # vreme pocetak
    pocetno_vreme = time.time()
    self.__broj_listova = len([x for x in self.__G.nodes() if self.__G.in_degree(x) == 0])
    # vreme kraj
    self.__broj_listova_vreme = round(time.time() - pocetno_vreme, 5)

# broj unutrasnjih cvorova
def __os_broj_unutrasnjih_cvorova(self):
    # vreme pocetak
    pocetno_vreme = time.time()
    self.__broj_unutrasnjih_cvorova = len([x for x in self.__G.nodes() if self.__G.in_degree(x) > 0])
    # vreme kraj
    self.__broj_unutrasnjih_cvorova_vreme = round(time.time() - pocetno_vreme, 5)

# broj ciklicnih cvorova
def __os_broj_ciklicnih_cvorova(self):
    # vreme pocetak
    pocetno_vreme = time.time()
    self.__broj_ciklicnih_cvorova = 0
    skup_povezanih_komponenti = nx.connected_component_subgraphs(self.__G.to_undirected())
    for povezana_komponenta in skup_povezanih_komponenti:
        grane_ciklusa = nx.find_cycle(self.__G, list(povezana_komponenta.nodes())[0])
        self.__broj_ciklicnih_cvorova += len(grane_ciklusa)
    # vreme kraj
    self.__broj_ciklicnih_cvorova_vreme = round(time.time() - pocetno_vreme, 5)

# prosecna duzina putanje, ciklusa i repa u grafu
# svaka putanja racuna se iz pocetne tacke, gde se kao skup pocetnih tacaka razmatra ceo skup cvorova grafa
# svaki ciklus racuna se iz pocetne tacke, gde se kao skup pocetnih tacaka razmatra ceo skup cvorova grafa
# svaki rep racuna se iz pocetne tacke, gde se kao skup pocetnih tacaka razmatra ceo skup cvorova grafa
# putanja je oblika 'Rho', tj  putanja = ciklus + rep
def __ls_prosecna_duzina_putanje_ciklusa_i_repa(self):
    # vreme pocetak
    pocetno_vreme = time.time()
    self.__putanje = []
    self.__najduza_putanja = 0
    self.__prosecna_duzina_putanje = 0
    self.__ciklusi = []
    self.__najduzi_ciklus = 0
    self.__prosecna_duzina_ciklusa = 0
    self.__repovi = []
    self.__najduzi_rep = 0
    self.__prosecna_duzina_repa = 0
    skup_povezanih_komponenti = nx.connected_component_subgraphs(self.__G.to_undirected())
    for povezana_komponenta in skup_povezanih_komponenti:
        grane_ciklusa = nx.find_cycle(self.__G, list(povezana_komponenta.nodes())[0])
        velicina_ciklusa = len(grane_ciklusa)
        self.__ciklusi.append(velicina_ciklusa)
        # pamcenje najduzeg ciklusa
        if velicina_ciklusa > self.__najduzi_ciklus:
            self.__najduzi_ciklus = velicina_ciklusa
        povezana_komponenta_bez_ciklusa = nx.Graph(povezana_komponenta)
        povezana_komponenta_bez_ciklusa.remove_edges_from(grane_ciklusa)
        for ciklicni_cvor, _ in grane_ciklusa:
            velicina_stabla = len(nx.ancestors(povezana_komponenta_bez_ciklusa, ciklicni_cvor)) + 1
            self.__prosecna_duzina_ciklusa += velicina_stabla * velicina_ciklusa
            duzine_repova =
                    nx.shortest_path_length(povezana_komponenta_bez_ciklusa, target=ciklicni_cvor).values()
            self.__prosecna_duzina_repa += sum(duzine_repova)
```

```
            self.__repovi += duzine_repova
            # pamcenje najduzeg repa
            if max(duzine_repova) > self.__najduzi_rep:
                self.__najduzi_rep = max(duzine_repova)
            duzine_putanja = [rep + len(grane_ciklusa) for rep in duzine_repova]
            self.__prosecna_duzina_putanje += sum(duzine_putanja)
            self.__putanje += duzine_putanja
            # pamcenje najduze putanje
            if max(duzine_putanja) > self.__najduza_putanja:
                self.__najduza_putanja = max(duzine_putanja)
    self.__prosecna_duzina_ciklusa /= len(self.__G.nodes())
    self.__prosecna_duzina_repa /= len(self.__G.nodes())
    self.__prosecna_duzina_putanje /= len(self.__G.nodes())
    # vreme kraj
    self.__prosecna_duzina_putanje_ciklusa_i_repa_vreme = round(time.time() - pocetno_vreme, 5)

# prosecna velicina povezane komponente i stabla grafa
# svaka povezana komponenta racuna se kao komponenta koja sadrzi pocetnu tacku, gde se kao skup pocetnih
# tacaka razmatra ceo skup cvorova grafa
# svako stablo racuna se kao stablo za pocetnu tacku, gde se kao skup pocetnih tacaka razmatra ceo skup
# cvorova grafa
# stablo za datu pocetnu tacku x_0: svi cvorovi koji na istom mestu ulaze u isti ciklus kao i x_0
def __ls_prosecna_velicina_povezane_komponente_i_stabla(self):
    # vreme pocetak
    pocetno_vreme = time.time()
    self.__povezane_komponente = []
    self.__najveca_povezana_komponenta = 0
    self.__prosecna_velicina_povezane_komponente = 0
    self.__stabla = []
    self.__najvece_stablo = 0
    self.__prosecna_velicina_stabla = 0
    skup_povezanih_komponenti = nx.connected_component_subgraphs(self.__G.to_undirected())
    for povezana_komponenta in skup_povezanih_komponenti:
        velicina_komponente = len(povezana_komponenta.nodes())
        self.__prosecna_velicina_povezane_komponente += velicina_komponente * velicina_komponente
        self.__povezane_komponente.append(velicina_komponente)
        # pamcenje najvece povezane komponente
        if velicina_komponente > self.__najveca_povezana_komponenta:
            self.__najveca_povezana_komponenta = velicina_komponente
        grane_ciklusa = nx.find_cycle(self.__G, list(povezana_komponenta.nodes())[0])
        povezana_komponenta_bez_ciklusa = nx.Graph(povezana_komponenta)
        povezana_komponenta_bez_ciklusa.remove_edges_from(grane_ciklusa)
        for ciklicni_cvor, _ in grane_ciklusa:
            velicina_stabla = len(nx.ancestors(povezana_komponenta_bez_ciklusa, ciklicni_cvor)) + 1
            self.__prosecna_velicina_stabla += velicina_stabla * velicina_stabla
            self.__stabla.append(velicina_stabla)
            # pamcenje najveceg stabla
            if velicina_stabla > self.__najvece_stablo:
                self.__najvece_stablo = velicina_stabla
    self.__prosecna_velicina_povezane_komponente /= len(self.__G.nodes())
    self.__prosecna_velicina_stabla /= len(self.__G.nodes())
    # vreme kraj
    self.__prosecna_velicina_povezane_komponente_i_stabla_vreme = round(time.time() - pocetno_vreme, 5)

# pomocna funkcija
# funkcija vraca ukupnu sumu broja prethodnika za svakog od prethodnika zadate tacke x_0 u zadatom grafu G
# suma se racuna rekurzivno, bazirano na BFS algoritmu
def __suma_prethodnika_svih_prethodnika_zadate_tacke(self, G, x_0, nivo=1):
    if len(list(G.predecessors(x_0))) == 0:
        return 0
    suma = 0
    for prvi_prethodnik in G.predecessors(x_0):
        suma += nivo + self.__suma_prethodnika_svih_prethodnika_zadate_tacke(G, prvi_prethodnik, nivo + 1)
    return suma

# prosecan broj prethodnika u grafu
# broj prethodnika racuna se za svaku od pocetnih tacaka, gde se kao skup pocetnih tacaka razmatra ceo skup
# cvorova grafa
# prethodnici date pocetne tacke x_0: skup svih tacaka koje se kroz jednu ili vise iteracija preslikavaju u
# x_0
def __ls_prosecan_broj_prethodnika(self):
    # vreme pocetak
    pocetno_vreme = time.time()
    self.__prosecan_broj_prethodnika = 0
```

```
G_bez_ciklusa = nx.DiGraph(self.__G)
ciklusi = []
skup_povezanih_komponenti = nx.connected_component_subgraphs(self.__G.to_undirected())
for povezana_komponenta in skup_povezanih_komponenti:
    grane_ciklusa = nx.find_cycle(self.__G, list(povezana_komponenta.nodes())[0])
    cvorovi_ciklusa = list(x for x, _ in grane_ciklusa)
    ciklusi.append(cvorovi_ciklusa)
    self.__prosecan_broj_prethodnika += len(povezana_komponenta.nodes()) * len(cvorovi_ciklusa)
    G_bez_ciklusa.remove_edges_from(grane_ciklusa)
for ciklus in ciklusi:
    for cvor in ciklus:
        self.__prosecan_broj_prethodnika +=
               self.__suma_prethodnika_svih_prethodnika_zadate_tacke(G_bez_ciklusa, cvor) -
               len(nx.ancestors(G_bez_ciklusa, cvor))
self.__prosecan_broj_prethodnika /= len(self.__G.nodes())
# vreme kraj
self.__prosecan_broj_prethodnika_vreme = round(time.time() - pocetno_vreme, 5)
```