Kolizijski napadi na oslabljene funkcije kriptografskog sažimanja

Katoliković, Ivan

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Ivan Katoliković

KOLIZIJSKI NAPADI NA OSLABLJENE FUNKCIJE KRIPTOGRAFSKOG SAŽIMANJA

ZAVRŠNI RAD

Varaždin, 2019.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Ivan Katoliković

Matični broj: 44012/15–R

Studij: Informacijski sustavi

KOLIZIJSKI NAPADI NA OSLABLJENE FUNKCIJE KRIPTOGRAFSKOG SAŽIMANJA

ZAVRŠNI RAD

Mentor:

Doc. dr. sc. Nikola Ivković

Varaždin, rujan 2019.

Ivan Katoliković

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

U ovom radu se generiraju poruke nasumičnog sadržaja koje se zatim provode kroz algoritme kriptografskog sažimanja. Algoritmi sažimanja koji se koriste za eksperimente su SHA-1, SHA-256 i SHA-3. Umjesto da se eksperimenti provode sa punim (neizmijenjenim) sažetcima, dobiveni sažetci se oslabljuju preklapanjem bitova koristeći XOR logička vrata. Tim postupkom se sažetci skraćuju (ovisno o razini oslabljenja, preciznije broju preklapanja). Nadalje se provodi traženje kolizija, a dodatni parametri su veličina poruka, vrsta sadržaja (alfanumerički i svi smisleni ASCII znakovi) i razine oslabljenja sažetaka.

Eksperimentima je prikazana pojava prve kolizije sažetaka dobivenih algoritmima sažimanja, te povećanja broja kolizija povećanjem broja generiranih poruka. Nadalje, koristi se tip poruka koji ima dinamički dio koji se mijenja tokom generiranja. Time dio poruke ostaje isti, a dio se izmjenjuje. Za takve poruke, traže se oni dinamički dijelovi koji ostvaruju koliziju.

Dobiveni rezultati pokazuju kako veličina poruka i njihov sadržaj, te algoritam sažimanja (kada je sažetak oslabljen na jednaku duljinu) nemaju utjecaj na pojavu kolizija, već o pojavi kolizije ovisi samo duljina sažetka.

Ključne riječi: kolizije, kolizijski napadi, kriptografsko sažimanje, oslabljenje sažetka, SHA-1, SHA-256, SHA-3, C++

Sadržaj

1. Uvod

U ovom radu ćemo provoditi eksperimente kolizijskih napada na oslabljene funkcije sažimanja. Algoritmi sažimanja koji se koriste su SHA-1, SHA-256 i SHA-3. Definirana su tri tipa poruka, poruke generirane od isključivo alfanumeričkih znakova, poruke generirane od svih ASCII znakova i poruke sa fiksnim i dinamičkim dijelovima. Oslabljenja sažetak se vrše na principu XOR logičkih vrata. Definirane su razine oslabljenja sažetka, ovisno o njegovoj duljini. Sažetak ima onoliko razina oslabljenja koliko njegova duljina ima djelitelja. Tako da SHA-1 sažetak koji je duljine 40 heksadekadskih znakova ima razine oslabljenja O = 1, 2, 4, 5, 8, 10, 20 i 40. Duljina oslabljenog sažetka je 40 / O. Ako je odabrana razina 10, sažetak se podijeli na 10 dijelova, svaki duljine 4. Zatim se prvi dio XOR vratima obradi sa drugim dijelom, dobiveni rezultat sa trećim dijelom, itd. Time se dobije oslabljeni sažetak duljine 40/O.

Aplikacija koja je izrađena u svrhu ovog rada može sažimati tekstualne poruke raznih veličina algoritmima sažimanja: SHA-1, SHA-256, SHA-3, Keccak, MD-5 i CRC-32. Koristi dretve radi bržeg izvođenja i zahtjevna je za memoriju. Radi se o aplikaciji za konzolu, a napisana je u jeziku C++, u okruženju Microsoft Visual Studio 2019.

Eksperimenti koji se vrše, gledaju prosjek pojavljivanja prve kolizije ovisno o raznim konfiguracijama. Također gledaju porast broja kolizija s porastom broja generiranih poruka i kod posebne vrste poruka (poruke s dinamičkim dijelom kod kojih se mijenja iz poruke u poruku brojčana vrijednost) koje poruke daju isti sažetak.

2. Teorijska osnova

U današnje doba gdje Internet postaje neophodan za poslovanje, komuniciranje, a tako i socijaliziranje, potreba za sigurnošću je sve veća. Kao odgovor na te potrebe dolazi kriptografija (eng. *cryptography*). Još od davnih dana ljudi su koristili metode kako bi važne poruke sakrili (pretvorili u nečitljive, nerazumljive) od onih kojima ista nije namijenjena. Jedna od ranijih, a vjerojatno najpoznatijih, je „Cezarova šifra“. Radi se o veoma jednostavnoj metodi koju danas nije teško dešifrirati, točnije jako je nesigurna. Kod nje se odredio pomak; brojčana vrijednost za koju su slova abecede bila pomaknuta. Na primjer, ako je pomak bio 5 i pisalo je slovo „F“, onda bi dekriptirano slovo bilo slovo „A“, i tako dalje. Danas se koriste puno složenije metode kriptografije kako bi se potencijalnom napadaču onemogućilo čitanje onoga što mu nije namijenjeno ili podmetanje zloćudnog sadržaja umjesto onoga kojega korisnik (žrtva) želi konzumirati.

Danas postoji veći broj algoritama kriptiranja koji imaju različite razine efektivnosti. Od najčešćih vrsta su sigurni algoritam sažimanja (eng. *Secure Hash Algorithm*, skraćeno SHA) koji imaju veći broj inačica od kojih su najkorištenije SHA-1 i SHA-256, a danas sve više i SHA-3. Također se koristi i MD5 (eng. *Message-digest Algorithm*) i njemu po sigurnosti slični ili slabiji algoritmi, no sve manje zbog njihove, danas, preslabe razine zaštite.

2.1. Kriptografski sažetak

Kada se datoteka, poruka ili bilo koja vrsta informacije provede kroz algoritam kriptografskog sažimanja, rezultat funkcije, dobiven matematičkim operacijama, je tekstualni niz od fiksnog broja znakova (ovisno o algoritmu kriptografskog sažimanja), najčešće kraćeg od izvorne poruke, točnije bitova. Tako, na primjer, tekst „Ovo je test kriptografskog sažetka.“ proveden kroz SHA-1 algoritam, daje niz od 40 heksadekadsnkih znakova (160 bitova): „f1b2bc7c6a266527b54b78ad95c5d8766c06d87d“. Kao što je vidljivo iz primjera, kriptografski sažetak nije ni približno sličan originalnoj poruci, te se iz njega izvorna poruka ne može zaključiti. Čak najmanja izmjena originalne poruke, recimo promjena kapitalizacije početnog slova „O“, će rezultirati potpuno izmijenjenim kriptografskim sažetkom. To je, naravno, i željeno ponašanje. U današnjem dobu interneta, gotovo svaka operacija (pretraživanje, prijavljivanje, razmjena poruka, certifikati i ostalo) koristi neku formu kriptografskog sažimanja radi sigurnosti i/ili prednosti koje s time dolaze. S obzirom da algoritmi kriptiranja rade samo u jednom smjeru, točnije, kriptografski sažetak se može dobiti provođenjem informacije kroz algoritam kriptiranja, ali informacija se ne može dobiti obrnutim postupkom, ovaj pristup sigurnosti služi primarno za autentifikaciju, no može služiti i ostalim

svrhama, kao na primjer ključevi u određenim strukturama podataka ili gitova upotreba sažetka za provjeru izmjena. Strukture podatka koje koriste sažetke imaju prednost u brzini dohvaćanja podataka. Sažimanje se također koristi kod digitalnih potpisa, gdje se sažetak kriptira privatnim ključem. Pojam rudarstvo (eng. *mining*) Bitcoin kripto valute u principu predstavlja provođenje SHA-256 algoritma sažimanja. Kriptografski sažetci imaju ulogu kod Rabin-Karp algoritma koji služi za detekciju plagijata.

Problem sa kriptografskim sažetcima je njihova nejedinstvenost. Veći broj informacija provedenih kroz isti algoritam kriptiranja, može, i bude rezultiralo istim sažetcima. Pretpostavimo da se dva korisnika prijavljuju na neki sustav, web stranicu. Oba korisnika imaju različite zaporke, ali njihove zaporke provedene kroz algoritam kriptiranja, koji taj sustav koristi, rezultiraju identičnim kriptografskim sažetkom. Nadalje, pretpostavimo da je razlika u korisničkim imenima minimalna, te drugi korisnik unese, zabunom (ili namjerno), korisničko ime prvog korisnika i SVOJU zaporku; time bi se prijavio u sustav kao prvi korisnik jer je, usprkos različitim zaporkama sustav proveo „krivu“ zaporku drugog korisnika kroz algoritam kriptiranja i dobio rezultat koji je pohranjen u bazi podataka kao ispravan sažetak zaporke za korisnički račun prvog korisnika. Međutim, namjerno traženje podataka koji imaju isti kriptografski sažetak nije lagano, ali postoje razne metode koje daju solidne rezultate.

2.2. Algoritmi kriptografskog sažimanja

Algoritmi kriptografskog sažimanja moraju zadovoljavati slijedeća svojstva [1]:

1. Računalna učinkovitost
    - Kao i kod mnogih računalnih funkcija, brzina izvođenja je uglavnom visoko prioritetna. Tako i algoritmi kriptografskog sažimanja moraju imati velike brzine izvođenja; to je bilo jako bitno prije četrdeset, pedeset godina, dok danas, zahvaljujući tehnološkom napretku sklopovlja i programske podrške, funkcije sažimanja jednostavne oblike podataka (npr. kraća poruka) provode u djeliću sekunde. No s obzirom da danas raspolažemo s puno većim podatcima, izračuni mogu potrajati duže vrijeme.
2. Determinističnost
    - Determinističnost je uvijek bitna kod programiranja, u suprotnom, programiranje ne bi imalo previše smisla. Informacija koja se sažima algoritmom kriptografskog sažimanja mora uvijek rezultirati istim kriptografskim sažetkom.

3. Otpornost na pred-sliku
    - Bitno je da algoritam kriptografskog sažimanja može primiti bilo kakav podatak, bilo to jedan znak, rečenica, itd. Nadalje, iz rezultata algoritma ne smije se moći zaključiti ulazni podatak. Kada bi izlaz bio ovisan o ulazu što se tiče duljine (kratak ulaz daje kratak izlaz, dugačak ulaz, dugačak izlaz), zaključiti o kakvoj duljini ulaznog podatka je riječ, bilo bi lako zaključiti na temelju izlaznog podatka. I najmanja izmjena ulaznog podatka, mora rezultirati drastično izmijenjenim izlaznim podatkom.

3. Otpornost na kolizije
    - Kada dva ulazna podatka daju identičan izlazni podatak, to nazivamo kolizija. S obzirom na otpornost na pred-sliku (rezultat mora biti fiksne duljine), dva različita ulazna podatka morati će dati identičan izlazni podatak. Nadalje, to je potkrijepljeno činjenicom da postoji beskonačno mnogo različitih ulaznih podataka, ali samo određen broj izlaznih podataka zbog fiksne duljine. Zato je bitno da je ta šansa što manja, te da je pronalaženje kolizija praktički neizvedivo u nekom realnom vremenu (na razumno dobro sklopovski opremljenom računalu). Što je bolji algoritam kriptografskog sažimanja, to je potreba za memorijom i obradom podataka veća, samim time se vrijeme traženja kolizija povećava.

2.2.1. SHA

SHA je akronim za *Secure Hash Algorithm* (hrv. Siguran Algoritam Sažimanja). To je obitelj algoritama sažimanja izrađenih od strane Nacionalnih Instituta Standarda i Tehnologije (eng. *National Institutes of Standards and Technology*) i drugih državnih (Sjedinjenih Američkih Država) i privatnih stranaka.

2.2.1.1. SHA-0

SHA-0, ranije zvan jednostavno SHA, je algoritam, baziran na MD5 obitelji kripto primitiva, koji kao rezultat sažimanja daje 160 bitni broj između 0 i 1.46×10^{48}. Šanse u odnosu na dva različita ulazna podatka da će doći do kolizije je približno 2^{80}. SHA je preimenovan u SHA-0 nakon što je uveden SHA-1. SHA-0 ima nepoznat problem koji lakše omogućava pronalaženje kolizija (navedeni problem/i nije/nisu objavljen/i iz sigurnosnih razloga), stoga je kao odgovor na to uveden SHA-1.

2.2.1.2. SHA-1

Iako je SHA-1 otklonio mane SHA-0, procedura rada je ostala ista. Izlazni podatak daje 160-bitni broj između 0 i 1.46×10^{48} sa šansom u donosu na dva različita ulazna podatka da

dođe do kolizije približno 2^{80}. Stoga je implementacija u postojeće sustave koji su koristili SHA-0, veoma lagana, te ne utječe na normalan rad. Od 2011. godine korištenje SHA-1 nije preporučljivo, te je sve lakše i isplativije pronalaženje kolizija.

2.2.1.3. SHA-2

Kao odgovor na sve češće i lakše probijanje SHA-1, osmišljena je obitelj SHA-2 algoritma sažimanja. U svom radu su primarno identični kao SHA-1 ali kao rezultat daju duži sažetak koji je samim time sigurniji. Primarne verzije SHA-2 su SHA-224, SHA-256, SHA-385, SHA-512, SHA-512/224 i SHA-512/256, no postoje i druge, manje korištene. Brojevi iza SHA predstavljaju dužinu izlaza i bitovima, dok brojevi iza kose crte predstavljaju verzije koje su otporne na napade produljenja dužine. Već je i najslabija verzija SHA-2, tehnički, otpornija na kolizije od SHA-1 (224 naprema 160). SHA-385 pa na dalje imaju pedeset posto bolje performanse na 64-bitnim računalima.

2.2.1.4. SHA-3

SHA-3 je u unutrašnjosti drugačiji od svojih prethodnika jer je baziran na keccak obitelji kripto primitiva i „spužvastoj strukturi“ što rezultira otpornošću na napade produljenja duljine. Dok je veoma sličan Keccak algoritmu sažimanja, nije potpuno identičan, već ima različito „ispunjavanje“ (eng. *padding*). Postoje različite verzije SHA-3 koje imaju različite duljine izlaza, slično kao i kod SHA-2.

2.2.2. MD

MD je akronim za Message-Digest (hrv. Sažimanje poruka). Najčešće se koristi za provjeru integriteta datoteka zbog, danas, niske razine sigurnosti. Poznatije verzije su MD3, MD4 i MD5. Zbog svojih kratkih duljina sažetaka (najdulji su kod MD5, 128 bitova) kolizijski napadi su veoma efektivni za probijanje, ali zbog svoje jednostavnosti su idealni za provjere integriteta datoteka gdje nije potrebna sigurnost od potencijalnih napadača (git).

2.3. Kolizijski napadi

Kolizijski napad je sličan matematičkom fenomenu rođendanskog paradoksa i na njemu se bazira. Ne radi se o pravom paradoksu, već je samo težak sa prihvatiti ljudskom mozgu. Ako u sobi imamo 23 osobe, šansa da barem dvije osobe imaju rođendan na isti dan je 50%. Ako imamo 75 osoba, šansa je 99.9% [2]. Kod „kolizija“ datuma rođenja ima 365 kombinacija, stoga kolizijski napadi nemaju tolike šanse za pronalazak kolizije (SHA-1 ima 16^{40} različitih kombinacija), no ovim znanjem, kolizijski napadi izgledaju optimističnije.

Kolizijski napadi su pokušaji pronalaženja dva (ili više) identična izlaza algoritma sažimanja (sažetka). Razni ulazi se provode kroz algoritam sažimanja, te se traže oni koji imaju iste izlaze. To je moguće postići iz razloga što ulazni podatci algoritma sažimanja imaju beskonačno mnogo kandidata, dok postoji točno određen broj izlaznih mogućnosti, što ovisi o vrsti algoritma sažimanja. S time na umu , lako je zaključiti kako postoji beskonačno mnogo ulaznih podataka koji daju isti izlaz. Međutim, u praksi nije lako pronaći takve podatke zbog velike količine mogućih sažetaka. Na primjer, SHA-256 algoritam daje sažetak dulje 64 heksadekadskih znakova, a to je 16^{64} različitih kombinacija. Također, zbog poveće količine različitih algoritama sažimanja, napadaču je bitno znati o kojem algoritmu je riječ.

3. Povezana istraživanja

3.1. SHAttered

2017. godine grupa istraživača iz CWI (Cryptology Group at Centrum Wiskunde & Informatica) iz Nizozemske u suradnji s istraživačkim timom iz Google-a su napravili dva različita PDF dokumenta s identičnim SHA-1 sažetkom [3]. Napad je trebao više od vrtoglavih 9,223,372,036,854,775,808 SHA-1 izračunavanja. Ovim pothvatom se poziva na prestanak korištenja SHA-1 algoritma, jer će ovim tempom biti lako probiv u skorijoj budućnosti.

3.2. Prva prijetnja SHA-1 algoritmu

2005. godine, istraživači (Xiaoyun Wang, Yiqun Lisa Yin i Hongbo Yu) su našli tehniku traženja kolizija punog 80-koračnog SHA-1 sažetaka sa manje od 2^{69} operacija sažimanja [4]. To je bio prvi uspjeh s kompleksnošću manjom od 2^{80}. Predviđali su da je uz pomoć modernih superračunala moguće pronaći prave kolizije na 70-koračni SHA-1 sažetak. Njihova analiza je bazirana na napadu skore kolizije SHA-0 algoritma, više-blokovske kolizijske tehnike i tehnike modifikacije poruka za kolizijske napade na MD-5.

3.3. Brzi kolizijski napadi na MD-5

2010. godine Marc Stevens sa tehnološkog sveučilište u Eindhovenu, Nizozemska, napravio je unaprijeđeni algoritam napada za pronalaženje kolizija dva bloka funkcije sažimanja MD-5. Prezentirana je nova tehnika koja omogućuje deterministično ispunjenje restrikcija za ispravno rotiranje diferencijala u prvoj rundi [5]. Prezentiran je novi algoritam, za pronalaženje prvog bloka, a za traženje drugog bloka korišten je Klimin algoritam. Napad je izvediv u roku oko jedne minute (prosječna kompleksnost $2^{32.3}$) na 3GHz Pentinum4 za nasumične preporučene početne vrijednosti. Za proizvoljne nasumične početne vrijednosti, prosječno vrijeme je oko 5 minuta (prosječna kompleksnost $2^{34.1}$). Sa razumnim vjerojatnošću za pronalazak kolizije unutar jedne sekunde, omogućen je, primjerice, napad tijekom izvođenja protokola.

3.4. Kolizijski napad jednim blokom na MD-5

2012. godine Marc Stevens, Kriptološka Grupa, CWI, Amsterdam. Napad je baziran na (barem tada) novom algoritmu traženja kolizija koji eksploatira nizak broj bitovnih uvjeta u

prvoj rundi [6]. Koristi način odabira blokova koji zadovoljava bitovne uvjete do 22. koraka i koristi 3 poznata tunela za ispravljanje bitovnih uvjeta do koraka 25. Napad ima prosječnu kompleksnost rada jednaku $2^{49.8}$ poziva MD-5 kompresijske funkcije.

3.5. Kolizijski napadi na NaSHA-384/512

NaSHA je bio jedan od kandidata za osnovu SHA-3 algoritma sažimanja.

2010. godine, grupa istraživača (Zhimin Li, Hongan Jiang i Cunhua Li) je prezentirala kolizijske napade na NaSHA (duljina izlaza 384 i 512 bitova) [7]. Napadi su bazirani na slabosti slijeda generiranja riječi stanja i činjenici da operacije kvazi grupe korištene u funkcijama kompresije su određene djelomičnim riječima stanja. Vremenska kompleksnost ovog napada je oko 2^{128} sa zanemarivom memorijom.

4. Aplikacija

Aplikacija za provođenje kolizijskih napada je izrađena u programskom jeziku C++. Radi se o aplikaciji za konzolu koja izvješća ispisuje u vanjske datoteke tekstualnog formata i/ili u samu konzolu. Napomena: jedan dio aplikacije koristi dretve uz pomoć vanjske biblioteke „future“ koja je uključena u standardni paket datoteka zaglavlja. Radno okruženje u kojem je aplikacija napisana je Microsoft Visual Studio 2019. Metoda traženja kolizija je sirova snaga (eng. *brute force*).

Biblioteka koja sadrži funkcije sažimanja je naziva „hash-library“ autora Stephann Brummea i dostupna je za preuzimanje na adresi: https://create.stephan-brumme.com/hash-library/. Napravljene su manje izmjene (nazivi imenskih prostora) u nekim datotekama biblioteke kako bi se omogućio rad sa više vrsta funkcija sažimanja u isto vrijeme. Algoritmi sažimanja koji su korišteni u aplikaciji i nalaze se u navedenoj biblioteci su SHA-1, SHA-256, SHA-3, Keccak, MD-5 i CRC-32.

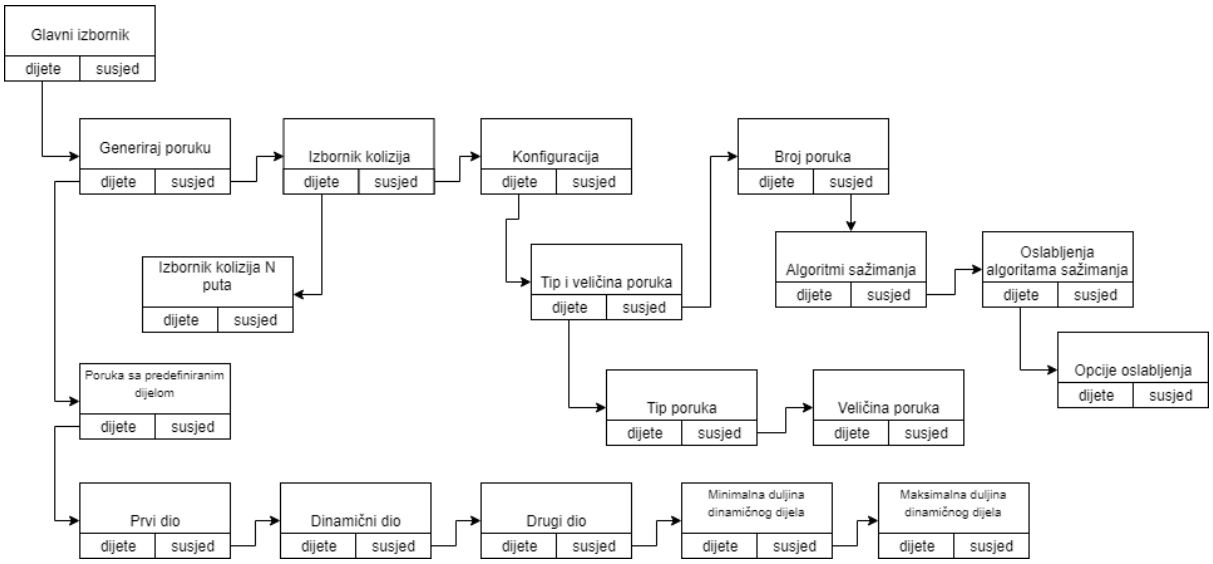
4.1. Izbornik

Izbornikom se služi tako da se upiše znak (broj ili znak „x“ koji je univerzalan za povratka u prethodni izbornik) ispred opcije koja se odabire. Izbornik aplikacije se sastoji od više podizbornika koji su strukturirani kao vrsta stabla. Dok nije najprohodnija struktura, za potrebe ovog izbornika je odgovarajuće. Korijen stabla je glavni izbornik koji ima opcije.

1. Generiraj poruku
    - Primarno služi za konstrukciju poruke sa dinamičkim dijelom (akko je tip poruke postavljen na dinamičku poruku) ali i u svrhu testiranja ispravnosti algoritama sažimanja ili oslabljenja istoga.
2. Traži kolizije
    - Izbornik nudi dvije opcije;
        1. Traži kolizije – generira se N poruka te se traže kolizije za one algoritme sažimanja koji su omogućeni u konfiguraciji. Pronađene kolizije se ispisuju u konzolu i vanjsku datoteku naziva „Collisions.txt“, te se sve generirane poruke zapisuju u vanjsku datoteku naziva „Messages.txt“
        2. Traži kolizije N puta – nakon unosa broja N, paralelno se, do 6 puta od jednom, generiraju poruke i traže kolizije. Dobiveni rezultati se zapisuju u konzulu i vanjsku datoteku, tekstualnog formata, naziva ovisnog o konfiguraciji aplikacije.

3. Konfiguracija
    - Postavke ponašanja aplikacije nalaze se u ovom izborniku. Po redu, mogućnosti su:
        1. Tip i veličina poruka – Tipovi poruka su alfanumerički, svi ASCII znakovi i dinamička poruka koju je potrebno, prije traženja kolizija, definirati u izborniku „Generiraj poruku“ u razini iznad.
        2. Broj poruka – predstavlja broj poruka generiranih po jednom izvođenju traženja kolizija.
        3. Algoritmi sažimanja – Pruža mogućnost odabira korištenog algoritama sažimanja (broj 0 u zagradi iza imena označava neaktiviranost, dok 1 aktiviranost).
        4. Oslabljenja – Oslabljenja kriptografskog sažetka su izvedena na sljedeći način: kandidati za oslabljenje su djelitelji duljine sažetka kojega algoritam sažimanja vrati kao svoj izlaz. Detaljnije o tome malo kasnije.

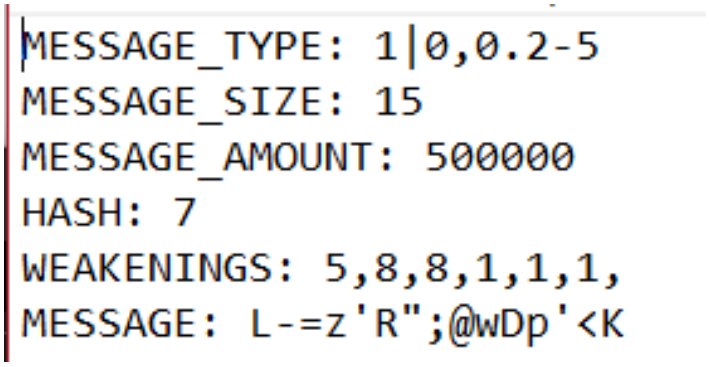
Detaljan prikaz strukture izbornika je prikazan na slici 1.



Slika 1: Struktura izbornika [autorski rad]

4.2. Konfiguracija

Konfiguracija aplikacije se sprema u vanjsku tekstualnu datoteku „CollAttConf.txt“, te primjer izgleda kao na slici 2.



Slika 2: Prikaz konfiguracije [autorski rad]

U prvom redu je redom (bitni su samo brojevi, ostali znakovi su separatori): tip poruke, indeks početka dinamičkog dijela, duljina dinamičkog dijela (bitno samo za poruku s dinamičkim dijelom), minimalna duljina generiranog dinamičkog dijela i maksimalna duljina dinamičkog dijela.

Drugi red ima samo duljinu poruka koje se generiraju, te nije bitan kod poruka sa dinamičkim dijelom.

Treći red predstavlja broj poruka koje se generiraju po jednom traženju kolizija.

Četvrti red predstavlja aktivirane algoritme sažimanja. U aplikaciji je moguće koristiti šest algoritama sažimanja u isto vrijeme (SHA-1, SHA-256, SHA-3, Keccak, MD-5, CRC-32), a ovaj sustav radi na sličnom principu kao i sustav dozvola za datotečne sustave, gdje se preko oktalnog sustava određuje što je moguće izvoditi sa datotekama. Ovdje se radi o sustavu sa bazom 64, gdje svaki algoritam sažimanja ima dodijeljenu vrijednost:

- SHA-1 – 1
- SHA-256 – 2
- SHA-3 – 4
- Keccak – 8
- MD-5 – 16
- CRC-32 – 32

Vrijednost 7 na slici 2 u četvrtom redu, stoga označava da su aktivirani algoritmi SHA-1, SHA-256 i SHA-3 (1+2+4). Kada bi uz navedena tri bio aktiviran i MD-5, onda bi vrijednost bila (1+2+4+16) 23. Funkcija koja provjerava je li algoritam aktiviran se vidi na slici 3.

```
bool CheckIfUsingHashAlg(int alg) {
    return ((settings[HASHES] >> (alg - SHA_1)) & 1) ? true : false;
}
```

Slika 3: Funkcija za provjeru korištenja algoritma sažimanja [autorski rad]

Ulazna varijabla „alg“ je enumeracija algoritma sažimanja, te označava pojedini algoritam; kada bi provjeravali ako je SHA-1 aktiviran, poziv funkcije bi izgledao: CheckIfUsingHashAlg(SHA_1). Funkcija vraća istinu (eng. *true*) ako je algoritam aktiviran i laž (eng. *false*) ako nije. „settings“ je numeričko polje sa konfiguracijom aplikacije, a „HASHES“ predstavlja element toga polja u kojemu je zapisana suma vrijednosti algoritama sažimanja ranije objašnjena. „>>“ se naziva bitovni pomak u desno, najlakše ga je interpretirati kao dijeljenje sa 2^x gdje je x redni broj algoritma sažimanja (0 – 6). Operator „&“ je logički operator AND. Najbolje je objasniti na primjeru; ako „HASH“ element polja „settings“ ima vrijednost 52 (bin. 110100) i provjeravamo ako je uključen Keccak algoritam (4. algoritma po redu, a poredak počinje od 0, tako da uzimamo vrijednost 3). Stoga, 1100100 >> 3 = 000110. 000110 & 1 = 0. Time zaključujemo kako Keccak nije aktiviran i funkcija vraća laž. Ako provjeravamo SHA-3 (redni broj 2) imamo: 110100 >> 2 = 001101, 001101 & 1 = 1, točnije SHA-3 je aktiviran i funkcija vraća istinu.

Peti red sadrži oslabljenja i to po redu SHA-1 (40 znaka), SHA-256 (64 znaka), SHA-3 (64 znaka), Keccak (64 znaka), MD-5 (32 znaka) i CRC-32 (8 znaka). Te vrijednosti su nadalje zapisane u polje „settings“.

Šesti red sadrži pohranjenu poruku generiranu prvom opcijom u izborniku „Generiraj poruku“, stoga nije potrebno ponovo generirati dinamičku poruku prilikom svakog pokretanja aplikacije.

Konfiguraciju je moguće mijenjati direktno preko datoteke, ali preporučljivo je izmjene raditi preko aplikacije, jer je na taj način predviđeno korištenje.

Funkcija koja učitava i pohranjuje konfiguraciju, naziva „Config()“, prima kao ulazni parametar broj 0 ili 1; 0 – učitaj konfiguraciju, 1 – pohrani konfiguraciju. Nećemo ulaziti u detalje funkcije, već je dovoljno reći kako ne vraća niti jednu vrijednost (eng. *void*; hrv. Praznina, ništa) i koristi ugrađenu biblioteku „fstream“ za rad sa datotekama. Prilikom pokretanja aplikacije, poziva se Config(0) kako bi se učitala konfiguracija, a kod izmjena se

poziva Config(1), s obzirom kako su sve postavke upisane u polje „settings“ (globalno polje), nije potrebno navoditi izmjene, već se samo postavke iz polja zapisuju u datoteku.

4.3. Traženje kolizija

Implementirane su dvije vrste traženja kolizija, obično traženje kolizija i traženje kolizija N puta, gdje je N broj ponavljanja. Kod prve vrste poziva se funkcija „Collide()“, a kod druge funkcija „CollideNTimes()“ koja N puta poziva funkciju „Collide()“. Funkcija „Collide()“ (slika 6) radi na slijedeći način. Prvo se deklariraju pripadne strukture podataka u koj će se spremati podatci generirani tijekom izvođenja. To su „mapa“, „collisions“ i „hits“. „mapa“ ima dualnu funkciju, prva je pohrana generiranih poruka i njihovih rednih brojeva kako bi ih se moglo ispisati nakon generiranja svih poruka; druga funkcija je osiguranje od pojave duplikata poruka. „hits“ sliži kao povratna vrijednost i u njoj su, nakon izvršenja glavnog dijela funkcije, zapisane oznake aktiviranih algoritama sažimanja, te redni brojevi poruka kod kojih je došlo do kolizije. „collisions“ je najopširnija od navedene tri strukture, a služi za ispis oznaka algoritama sažimanja, sažetaka i poruka koje su rezultirale tim sažetkom.

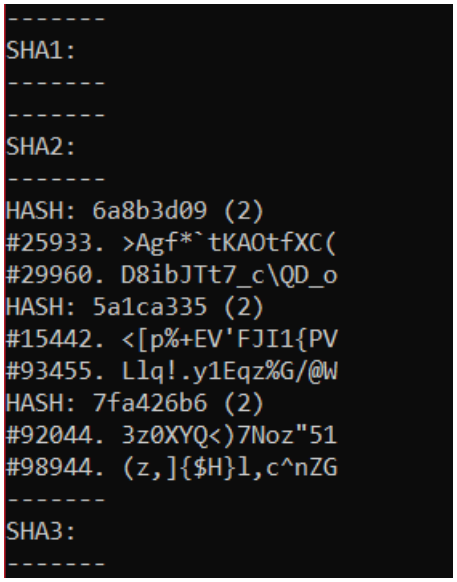
Princip rada:

1. Provjeri koji algoritmi sažimanja su u upotrebi i napravi oznake korištenih upiši u „hits“ i „collisions“ (u paru s oznakama su prazni vektori u koje će se kasnije upisivati podatci).
2. Petlja(brojač=0; Ako je brojač manji od broja poruka definiranih u konfiguraciji; brojač++)
    i. Deklariraj praznu varijablu tipa znakovni niz
    ii. Provjeri koji je tip poruke u konfiguraciji i pozovi funkciju koja takvu generira, a njen rezultat upiši u varijablu iz koraka a.
    iii. Pozovi funkciju za generiranje sažetaka svih aktiviranih algoritama sažimanja i njen rezultat upiši u novu varijablu tipa strukture koja ima šest varijabli znakovni niz, gdje svaka predstavlja jedan od šest algoritma sažimanja.
    iv. Probaj unijeti poruku u „mapa“, ako uspije idi dalje, ako ne brojač smanji za jedan (time se ponovo generira nova poruka i osigurava da se generira točno onaj broj poruka zadan u konfiguracji).
    v. Upiši u „collisions“ i „hits“ potrebne podatke
3. Ako je potreban ispis poruka i kolizija, ispiši ih u konzolu i u vanjske datoteke
4. Vrati „hits“

Vektor „hits“ nadalje ima svrhu samo funkciji „CollideNTimes()“ koja ga koristi za računanje aritmetičke sredine i medijana prve pojave kolizije.

4.3.1. Traženje kolizija prvom vrstom

Namjerno ne zovem ovo traženje kolizija „traženje kolizija jednom“ zbog toga što se to može napraviti i drugom vrstom, a rezultati nisu isti. Rezultat ove vrste traženja kolizija je popis poruka po oslabljenim sažetcima razvrstanih po algoritmima sažimanja. Na slici 4 je prikazan rezultat jednog takvog traženja; parametri su: tip poruke: svi ASCII, veličina poruka: 15 znakova (15 bajtova), broj generiranih poruka: 100,000, aktivni algoritmi: SHA-1, SHA-256, SHA-3, oslabljenja (redom): 5, 8, 8.



Slika 4: Primjer rezultata traženja kolizija [autorski rad]

SHA-1 i SHA-3 nisu imali kolizije, ali zato kod SHA-256 imamo 6 poruka sa istim sažetkom (oslabljenim). Važno je napomenuti kako za pojavu kolizije uzimamo redni broj druge poruke. Rezultat čitamo: sažetak 6a8b3d09 se pojavio 2 puta (broj u zagradi), a poruke koje imaju taj sažetak su poruke sa rednim brojevima 25,933 i 29,960. Prva kolizija pojavila se nakon 29,960 poruka. I tako, redom, za ostale sažetke.

Primijetimo kako rezultati nisu kronološki poredani, stoga za bolju analizu kolizija, druga vrsta generiranja poruka je preporučljiva.

4.3.1.1 Funkcija za traženje kolizija

Na slijedećoj slici je prikaz funkcije „Collide()“ koja služi traženju kolizija (Slika 5).

```
331 std::vector<std::pair<int, std::vector<int>>> Collide(bool disp) {
332     std::unordered_map<std::string, int> mapa;
333     std::vector<std::pair<int, std::unordered_map<std::string, std::vector<std::pair<int, std::string>>>>> collisions;
334     std::vector<std::pair<int, std::vector<int>>> hits;
335     for (int i = SHA_1; i < settSize; i++) {
336         if (CheckIfUsingHashAlg(i)) {
337             collisions.push_back(std::pair<int, std::unordered_map<std::string, std::vector<std::pair<int, std::string>>>>(i, std::unordered_map<std::string, std::vector<std::pair<int, std::string>>>()));
338             hits.push_back(std::pair<int, std::vector<int>>(i, std::vector<int>()));
339         }
340     }
341     for (int i = 1; i <= settings[NUM]; i++) {
342         if (i % 1000 == 0) std::cout << i << std::endl;
343         std::string nMsg;
344         switch (settings[TYPE]) {
345         case 0:
346             nMsg = GenPassLikeMsg();
347             break;
348         case 1:
349             nMsg = GenAllASCIIMsg();
350             break;
351         case 2:
352             std::string dynam = GenNumericMsg();
353             nMsg = msg;
354             nMsg.replace(settings[FIRST], settings[DYNAM], dynam);
355             break;
356         }
357         Hashes nHash = GenHashes(nMsg);
358         std::pair<std::unordered_map<std::string, int>::iterator, bool> rez = mapa.insert(std::pair<std::string, int>(nMsg, i));
359         if (!rez.second) i--;
360         else {
```

Slika 5: Funkcija koja traži kolizije [autorski rad]

Funkcija vraća vrijednost zbog druge vrste traženja kolizija (N puta). Vrijednost koju vraća je kolekcija rednih brojeva kolizija razvrstanih po algoritmima sažimanja u obliku: vektor(par(algoritam, vektor(redni brojevi)). Algoritam je numeričkog tipa zato što se radi o enumeraciji.

Ulazna vrijednost funkcije je tipa boolean (istina-laž, 1-0), a o toj vrijednosti ovisi ispis. Varijabla „mapa“ je tipa unordered_map (hrv. neuređena mapa) kojoj je ključ generirana poruka, a vrijednost njen redni broj. „mapa“ je toga tipa kako ne bi došlo do ponavljanja istih poruka, koje će, naravno, dati isti sažetak, te time stvoriti iluziju kolizije, iako zapravo to nije.

Varijabla „collisions“ služi za pohranu kolizija po algoritmima sažimanja. Strukturirana je kao: vektor(par(algoritam, mapa(sažetak, vektor(par(redni broj poruke, poruka))). Malo je kompleksnija struktura zbog činjenice da se može koristiti više algoritama sažimanja u isto vrijeme, te ih je potrebno razvrstati. Varijablu „hits“ funkcija vraća nakon obrade. Na liniji 335 se priprema varijabla „collisions“, točnije u vektor se upisuju algoritmi sažimanja koji se koriste. Stoga, ako koristimo tri algoritma sažimanja, veličina tog vektora biti će 3. Radi se o for (hrv. Za) petlji koja ima početnu varijablu „i“ postavljenu na vrijednost enumeracije SHA_1 koja je oznaka prvog po redu algoritma sažimanja. Varijabla „settSize“ je veličina polja „settings“ (globalno polje sa konfiguracijom), a algoritmi sažimanja su raspoređeni indeksima od SHA_1 do kraja polja. Petlja se izvršava sve dok varijabla „i“ više ne bude manja od varijable „settSize“ (dok se ne provjeri aktiviranost svakog algoritma sažimanja). Nakon svake iteracije petlje, vrijednost varijable „i“ se poveća za 1, točnije, ako je „i“ jednako SHA_1, kada se poveća za jedan postaje SHA_2 itd.

Svakom iteracijom se provjerava korištenje algoritma označenog varijablom „i“ preko funkcije CheckIfUsingHashAlg() ranije objašnjene. Ako ta funkcija vrati istinu, algoritam se dodaje u vektore „collisions“ i „hits“.

U for petlji na liniji 341 započinje sama generacija poruka i njihovo sažimanje i oslabljivanje sažetaka. Ovaj proces se ponavlja onoliko puta koliko je postavljeno u konfiguraciji. Grananje na liniji 342 omogućuje ispis svakih 1000 generiranih poruka, a ispisuje jednostavno redni broj poruke koja se trenutno obrađuje iz razloga što ovaj proces traj duže vrijeme, pa da se ne stvori privid „zamrznute“ aplikacije. Dalje se deklarira varijabla „nMsg“ koja će služiti za pohranu generirane poruke. Na liniji 344 počinje provjera tipa poruke.

Ovisno o tipu poruke, poruka se generira i dodjeljuje varijabli „nMsg“. Kada je tip poruke 0, onda se radi o alfanumeričkoj poruci i pokreće se funkcija GenPassLikeMsg() koja ju generira. Ako je tip poruke 1, onda se radi o poruci koja može sadržavati bilo koji ASCII znak, a generira se preko funkcije GenAllASCIIMsg() i pohranjuje u varijablu „dynam“ tipa znakovnog niza. Na posljetku, ako je tip poruke 2, onda se dinamički dio poruke generira preko funkcije GenNumericMsg() (kao što je zaključivo, rezultat je kombinacija brojeva kojoj može prethoditi znak minus (-)). Nadalje, kod ovog slučaja, varijabla „nMsg“ poprima vrijednost varijable „msg“ koja sadrži predložak za dinamičku poruku i definira se preko izbornika „Generiraj poruku“. Nakon toga se umeće u varijablu „nMSg“ generirani dinamički dio.

Funkcije za generiranje poruka koriste uniform_int_distribution, a rade na slijedeći način:

1. Uniform_int_distribution služi za nasumični odabir brojeva, koje kasnije, aplikacija, pretvara u znakove, ovisno koja funkcija ga koristi. Pripadni kôd je na slici 6. Prvo je

zadan „random_device“ koji služi za određivanje sjemena (eng. *seed*) mehanizam nasumičnog odabira. „mt19937_64“ (Mersenne_twister_engine) je mehanizam za nasumičan odabir brojeva. Od linije 27 do 31 se određuju intervali generiranja brojeva koje slijedeće funkcije koriste za kreaciju nasumično generiranih poruka.

```
25    std::random_device rd;
26    std::mt19937_64 gen(rd());
27    std::uniform_int_distribution<> dis(1, 60);
28    std::uniform_int_distribution<> intDis(48, 57);
29    std::uniform_int_distribution<> upCDis(65, 90);
30    std::uniform_int_distribution<> lowCDis(97, 122);
31    std::uniform_int_distribution<> allASCII(32, 126);
```

Slika 6: Postavke generacije nasumičnih brojeva [autorski rad]

2. Generiranje alfanumeričkih poruka

```
281  std::string GenPassLikeMsg() {
282      std::string nMsg;
283      for (int i = 0; i < settings[SIZE]; i++) {
284          int j = dis(gen);
285          if(j>=35) nMsg += (char)lowCDis(gen);
286          else if(j>=10) nMsg += (char)upCDis(gen);
287          else nMsg += (char)intDis(gen);
288      }
289      return nMsg;
290  }
```

Slika 7: Funkcija za generaciju alfanumeričkih poruka [autorski rad]

Funkcija GenPassLikeMsg() (slika 7) vraća znakovni niz, to jest generiranu alfanumeričku poruku. For petlja se vrti onoliko puta koliko je postavljeno u konfiguraciji kao veličina poruke (tu vrijednost dohvaća iz polja „settings“, elementa indeksa enumeracije „SIZE“). Za alfanumerički tip poruke, na raspolaganju je 60 ASCII znakova, 25 malih, 25 velikih slova i 10 brojeva. Zbog toga je „dis“ ograničen na interval od 1 do 60 ([1, 60]). Kada je varijabla „j“, koja je poprimila nasumičnu vrijednost generiranu preko „dis“, veća ili jednaka 35 onda se na kraj znakovnog niza „nMsg“ dodaje malo slovo koje je nasumično generirano preko „lowCDis“ koji je limitiran na interval [97, 122] (u ASCII tablici, to su numeričke vrijednosti malih slova), a zatim pretvoreno u znak. Ako je „j“ manji od 35 i veći ili jednak 10, onda se preko „upCDis“ generira broj na intervalu [65, 90] (u ASCII tablici vrijednosti velikih slova) i dodaje na kraj znakovnog niza „nMsg“. U suprotnom, „intDis“ generira broj u intervalu [48, 57],

točnije ASCII vrijednosti znakova 0 do 9. Nakon što se petlja izvrši, funkcija vraća generiranu alfanumeričku poruku.

3. Generiranje poruke od svih (smislenih) ASCII znakova (slika 8)

```
292  std::string GenAllASCIIMsg() {
293      std::string nMsg;
294      for (int i = 0; i < settings[SIZE]; i++) nMsg += (char)allASCII(gen);
295      return nMsg;
296  }
```

Slika 8: Generacija poruka sa svim ASCII znakovima [autorski rad]

Ovdje je generacija poruka jednostavnija jer su svi smisleni znakovi iz ASCII tablice na intervalu [32, 126]. Proces je sličan; for petlja se vrti onoliko puta koliko je zadana dužina poruke u konfiguraciji i svakom iteracijom se na kraj znakovnog niza „nMsg“ dodaje znak iz ASCII tablice s numeričkom vrijednosti iz intervala [32, 126].

4. Generiranje dinamičkog djela poruke (slika 9)

```
298  std::string GenNumericMsg() {
299      std::uniform_int_distribution<> d(settings[MIN], settings[MAX]);
300      std::string nMsg;
301      if (dis(gen) <= 30) nMsg = "-";
302      for (int j = d(gen), i= 0; i < j; i++) nMsg += (char)intDis(gen);
303      if (nMsg[0] == '-') while (nMsg[1] == '0') nMsg.erase(1, 1);
304      else while (nMsg[0] == '0') nMsg.erase(0, 1);
305      int msgSize = nMsg.length();
306      if (nMsg[0] == '-') msgSize = nMsg.length() - 1;
307      if (msgSize<settings[MIN]) nMsg = GenNumericMsg();
308      return nMsg;
309  }
```

Slika 9: Generacija dinamičkog dijela poruka [autorski rad]

Ovdje je uniform_int_distribution na lokalnoj razini funkcije jer radimo sa dinamičkim podatcima. „settings[MIN]“ i „settings[MAX]“ su minimalna i maksimalna vrijednost duljine dinamičkog djela poruke, te su kao takvi ovisni o konfiguraciji aplikacije koja se mijenja (uglavnom) za vrijeme rada aplikacije. Na liniji 301 se generira broj između 1 i 60, ako je manji ili jednak 30, onda se dodaje znak minus (-), time postižemo pedeset posto šanse za negativan i pedeset posto šanse za pozitivan broj. Na liniji 302 započinje generiranje brojeva gdje je: „j“ nasumično definirana vrijednost iz intervala [„settings[MIN]“, „settings[MAX]“] i „i“ jednako nula. Dok se izvršava for petlja, za nasumično generiranje brojeva koristi se ranije spomenuti „intDis“.

Na linijama 303 i 304 se vrši brisanje potencijalno generiranih početnih nula. Ako je broj započeo sa znakom minus, brišu se sve nule nakon znaka minus dok znak poslije minusa nije neki drugi broj osim nule. Ukoliko nakon brisanja potencijalnih višaka znakova 0, ako je generirana prekratka poruka, funkcija se poziva rekurzivno.

Sada kada je generirana poruka, slijedeće ju je potrebno provesti kroz algoritam/me sažimanja i oslabiti sažetak/ke. To radi funkcija „GenHashes()“, a njena povratna vrijednost pohranjuje se u strukturu „Hashes“ (linija 357 na slici 6) koja je sačinjena od šest varijabli tipa znakovni niz gdje svaka predstavlja oslabljeni sažetak jednog od algoritama sažimanja.

Dalje se generirana poruka upisuje u mapu „mapa“. Ako je uspješno upisana (nije duplikat) prelazi se na daljnju obradu, a u suprotnom se brojač „i“ smanjuje za 1 kako bi se generala nova poruka (ovaj slučaj je veoma rijedak).

```
367         for (int j = 0; j < collisions.size(); j++) {
368             switch (collisions[j].first) {
369             case SHA_1:
370                 if (collisions[j].second.find(nHash.sha_1) != collisions[j].second.end()) {
371                     collisions[j].second[nHash.sha_1].push_back(std::pair<int, std::string>(i, nMsg));
372                     hits[j].second.push_back(i);
373                 }
374                 else {
375                     collisions[j].second.insert(std::pair<std::string, std::vector<std::pair<int, std::string>>>(nHash.sha_1, std::vector<std::pair<int, std::string>>()));
376                     collisions[j].second[nHash.sha_1].push_back(std::pair<int, std::string>(i, nMsg));
377                 }
378                 break;
```

Slika 10: Funkcija Collide(), razvrstavanje potencijalnih kolizija po algoritmima [autorski rad]

Na liniji 367 pokreće se nova for petlja kojom se prolazi kroz svaki algoritam sažimanja u vektoru „collisions“. Na slici 10 prikazan je postupak za SHA-1 algoritam. Postupak se svodi na grananje; ako je sažetak već u mapi (algoritma SHA-1) unutar vektora „collisions“, upisuje se u mapu novo generirana poruka i njen redni broj (varijabla „i“), te se u vektor „hits“ algoritma SHA-1 upisuje redni broj (varijabla „i“). Primijetimo da se vektor „hits“ puni samo kada dođe do kolizije (broj poruka s istim sažetkom veći od 1). Ako se radi o sažetku koji nije u mapi, isti se u nju upisuje i kreira se prazan vektor, a zatim se u njega upisuje redni broj poruke (varijabla „i“).

Isti postupak se provodi za sve aktivirane algoritme. Postupak se dalje ponovo izvršava sve dok nije generirano onoliko poruka koliko je zadana u konfiguraciji. Kada su generirane i obrađene sve poruke, dalje slijedi ispis na ekran i u datoteku (samo kod prve vrste generiranja).

4.3.2. Traženje kolizija drugom vrstom (N puta)

Prije nego što započne traženje kolizija, korisnik upisuje u konzolu broj ponavljanja, a zatim se poziva funkcija „CollideNTimes()“ (slika 11) koja ima za ulazni parametar broj ponavljanja „n“ (vrijednost koju je korisnik upisao).

Ova vrsta traženja kolizija ima drugačiji ispis, ali nećemo objašnjavati prve tri linije, jer služe samo u svrhu ispisa u datoteku. Na liniji 515 se u varijablu „z“ posprema (u svrhu čuvanja) vrijednost varijable „n“ (broj ponavljanja). Potom su potrebne dvije strukture podataka, jedna u koji ćemo pospremati redne brojeve poruka (koje su rezultirale kolizijom) razvrstane po algoritmima sažimanja, a druga u koji ćemo pospremati aritmetičku sredinu i medijan prvih pogodaka. Za pospremanje svih pogodaka se koristi vektor „hitCollection“, a struktura mu je slijedeća: vektor(par(algoritam, vektor(vektor(redni broj kolizije)))). Vektor „u sredini“ je tu iz razloga što se radi o N ponavljanja.

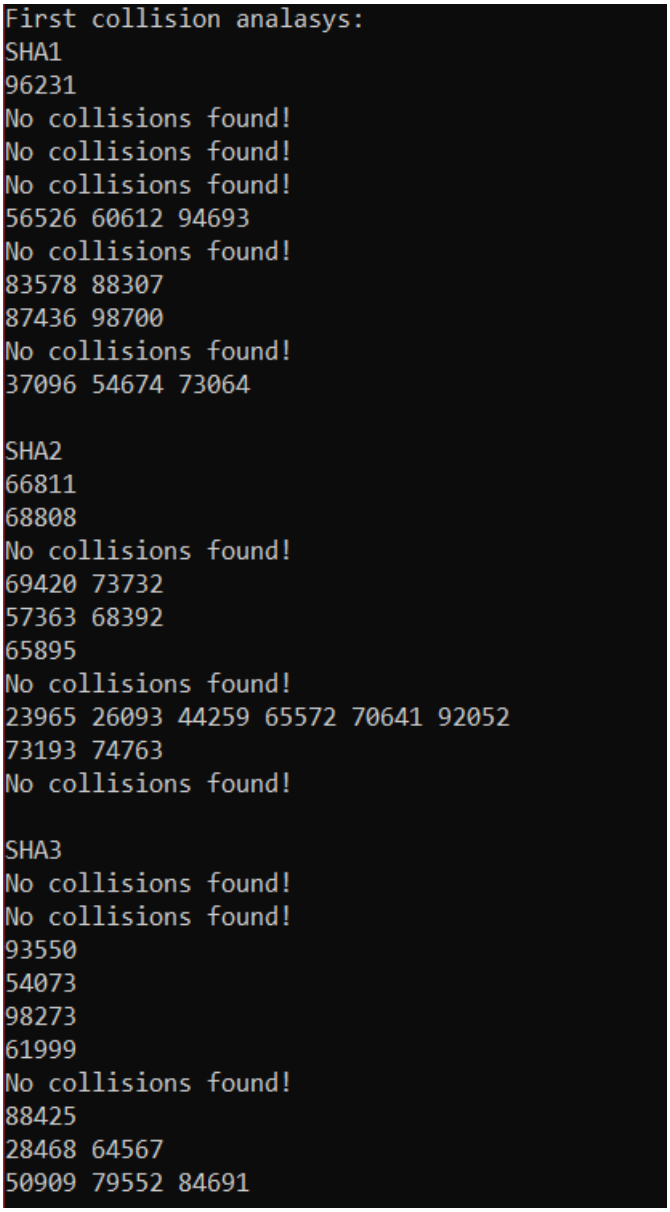
Petljom na liniji 518 se priprema vektor „hitCollection“ za unos podataka. Provjeravaju se aktivirani algoritmi sažimanja i njihove enumeracije upisuju u vektor. Dalje prelazimo na više-dretvenost; kreira se „vektor budućnosti“. Future (hrv. Budućnost) sadrži povratnu vrijednost funkcije koja se poziva asinkrono, a s obzirom da koristimo više od jedne dretve, potreban je vektor u koji će se vraćeno pohranjivati. Na liniji 520 je to definirano, ukratko se radi o strukturi: vektor(future(vraćeno iz funkcije „Collide()“)).

Petljom while (hrv. Dok) se N puta (varijabla „n“) poziva (asinkrono) funkcija „Collide()“. Ograničeno je na šest istovremenih izvršavanja, stoga ako je vrijednost varijable „n“ jednaka 10, prvo će se pozvati šest puta funkcija „Collide()“ i čekati dok se ne izvrši šesta (ne mora biti zadnja koja se izvršila), a zatim se poziva još četiri puta i čeka dok se ne izvrši zadnja (sada 10 po redu). Na liniji 524 se vraćena vrijednost funkcije „Collide()“ zapisuje u vektor. Potom se prolazi kroz vektor „fut“ i vrijednosti se zapisuju u vektor „hitCollection“.

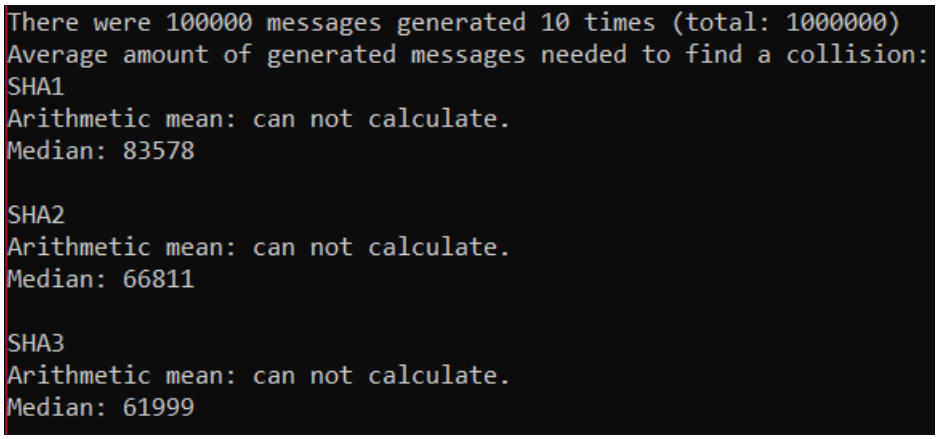
```
511  void CollideNTimes(int n) {
512      std::fstream analasysFile;
513      std::string fileName = std::to_string(settings[TYPE]) + ";SHA1-" + std::to_string(settings[SHA_1]) +";SHA2-"+ std::to_string(settings[SHA_2]
514      analasysFile.open(fileName, std::fstream::out | std::fstream::trunc);
515      int z = n;
516      std::vector<std::pair<int, std::vector<std::vector<int>>>> hitCollection;
517      std::vector<std::pair<double, double>> firstAvrgs;
518      for (int i = SHA_1; i < settSize; i++)
519          if (CheckIfUsingHashAlg(i)) hitCollection.push_back(std::pair<int, std::vector<std::vector<int>>>(i, std::vector<std::vector<int>>()));
520      std::vector<std::future<std::vector<std::pair<int, std::vector<int>>>>> fut;
521      while (n>0) {
522          int i = 0;
523          for (; i < 6 && n-i > 0; i++) {
524              fut.push_back(std::async(std::launch::async, Collide, false));
525          }
526          fut[fut.size() - 1].wait();
527          n -= 6;
528      }
529      for (int i = 0; i < fut.size(); i++) {
530          std::vector<std::pair<int, std::vector<int>>> oneRun = fut[i].get();
531          for (int j = 0; j < hitCollection.size(); j++) {
532              for (std::pair<int, std::vector<int>> hAlg : oneRun) {
533                  if (hAlg.first == hitCollection[j].first) hitCollection[j].second.push_back(hAlg.second);
534              }
535          }
536      }
```

Slika 11: Funkcija CollideNTimes() – dio sa obradom podataka [autorski rad]

Nakon što su generirani svi potrebni podatci, računa se aritmetička sredina i medijan prve pojave kolizije. Ako kolizije nije bilo u jednom setu traženja, aritmetičku sredinu nije moguće izračunati. Primjeri rezultata su na slikama 12 i 13.



Slika 12: Rezultat traženja kolizija 10 puta [autorski rad]



Slika 13: Aritmetičke sredine i medijani rezultata sa slike 13 [autorski rad]

Korišteni su algoritmi SHA-1, SHA-256 i SHA-3. Analiza SHA-1 rezultata: U prvom setu (100000 poruka) pronađena je samo jedna kolizija i ima redni broj 96231. U slijedeća tri seta nije bilo kolizija, kao ni u setovima 6 i 9. U petom setu su bile tri kolizije, u sedmom i osmom po dvije i u desetom tri. S obzirom da u pet setova nije bilo niti jedne kolizije, aritmetička sredina nije izračunata. Medijan je 83578.

4.4. Generiranje i oslabljenje sažetaka

Ovisno o algoritmu sažimanja, kriptografski sažetak ima razne duljine. Za SHA-1 to je 160 bitova (40 heksadecimalna znakova), za SHA-256, SHA-3 i Keccak 256 bitova (64 heksadecimalna znaka), MD-5 128 bitova (32 heksadekadska znaka) i za CRC-32 32 bitova (8 heksadekadska znaka).

Sažimanjem poruka upravlja funkcija „GenHashes()“ prikazana na slici 14.

```
186  Hashes GenHashes(std::string toHash) {
187      Hashes hashes;
188      for (int i = SHA_1; i < settSize; i++) {
189          if (CheckIfUsingHashAlg(i)) {
190              switch(i) {
191              case SHA_1: {
192                  SHA1 sha1;
193                  hashes.sha_1 = WeakenHash40(sha1(toHash), i);
194                  break;
195              }
196              case SHA_2: {
197                  SHA256 sha256;
198                  hashes.sha_2 = WeakenHash64(sha256(toHash), i);
199                  break;
200              }
```

Slika 14: Funkcija generacije svih sažetaka (prikazano: SHA-1 i SHA-256) [autorski rad]

Prikazan je postupak generiranja sažetaka za SHA-1 i SHA-256 algoritme, no slično vrijedi i za ostale algoritme sažimanja (zbog duljine funkcije i gotove identičnosti, ostali nisu prikazani). Funkcija ima ulazni parametar „toHash“ tipa znakovni niz koji predstavlja poruku koju sažimamo. Nadalje, struktura tipa „Hashes“ naziva hashes služi za vraćanje kolekcije sažetaka nakon sažimanja (i oslabljivanja) svim potrebnim (aktivniranim) algoritmima. For petljom se prolazi kroz sve algoritme sažimanja. U svakoj iteraciji se provjerava (funkcijom „ChechIfUsingHashAlg()“, ulazni parametar joj je oznaka algoritma) koristi li se trenutni algoritam, ako se koristi dalje se nastavlja na granje. Kada je slučaj SHA-1, koristi se SHA1 iz biblioteke „hash-library“. Na liniji 193 se u strukturu „hashes“, „sha_1“ znakovni niz dodjeljuje vrijednost funkcije „WeakenHash40()“ koja oslabljuje sažetak, a ulazni parametri su joj poruka

sažeta SHA-1 algoritmom i enumeracija SHA-1 algoritma koja predstavlja indeks polja „settings“ na kojemu je zapisana vrijednost oslabljenja sažetka za SHA-1 algoritam. Postupak je sličan za ostale algoritme sažimanja, a razlika je u tome što se poruka sažima ostalim algoritmima sažimanja i koristi se druga funkcija za oslabljivanje (ovisno o duljini sažetka). Tako da SHA-1 koristi funkciju „WeakenHash40()“, SHA-256, SHA-3 i Keccak koriste funkciju „WeakenHash64()“, MD-5 „WeakenHash32()“ i CRC-32 „WeakenHash8()“. Sve navedene funkcije oslabljivanje vrše XOR logičkim vratima.

4.4.1. WeakenHash40()

Na slici 15 je prikazana funkcija „WeakenHahs40()“ koja služi za oslabljivanje sažetka duljine 40 znakova (160 bitova).

```
77  std::string WeakenHash40(std::string hash, int alg) {
78      long long bin;
79      std::bitset<40> bits;
80      int part = 40 / settings[alg];
81      if (part <= 10) {
82          for (int i = 0; i < 40; i += part) {
83              std::stringstream ss;
84              ss << std::hex << hash.substr(i, part);
85              ss >> bin;
86              std::bitset<40> b(bin);
87              bits ^= b;
88
89          }
90          std::stringstream ss;
91          ss << std::hex << bits.to_ullong();
92          return ss.str();
93      }
94      else if (part == 40) hash;
95      else {
96          std::string weakHash;
97          for (int i = 0; i < 2; ++i) {
98              std::stringstream ss;
99              ss << std::hex << hash.substr(i * 10, 10);
100             ss >> bin;
101             std::bitset<40> b(bin);
102             ss.str(""); ss.clear();
103             ss << std::hex << hash.substr((i + 2) * 10, 10);
104             ss >> bin;
105             b ^= std::bitset<40>(bin);
106             ss.str(""); ss.clear();
107             ss << std::hex << b.to_ullong();
108             weakHash += ss.str();
109             while (weakHash.length() < (i+1) * 10) weakHash.insert(i * 10, "0");
110         }
111         return weakHash;
112     }
113 }
```

Slika 15: Funkcija oslabljenja sažetka veličine 160 bitova [autorski rad]

Funkcija vraća znakovni niz, a ulazni parametri su joj sažetak i indeks polja „settings“ u kojemu je zapisano oslabljenje algoritma kojim je stvoren sažetak. Funkcija se može koristit za oslabljivanje svih sažetaka, neovisno o algoritmu sažimanja, koji su duljine 40 znakova.

Sažetak (koji je u stvari heksadekadski broj zapisan u formatu znakovnog niza) se može interpretirati i kao običan (dekadski) broj (jako veliki broj u ovom slučaju), tako da varijabla „bin“ služi za realizaciju takve interpretacije. Na tip podataka „bitset“ (niz bitova) možemo gledati kao na polje istina i laži, jedinica i nula, a koristiti ćemo ga za logičku operaciju XOR (isključivo ILI).

Kod ove funkcije, kandidati za razinu oslabljivanja su 1, 2, 4, 5, 8, 10, 20 i 40 (djelitelji broja 40), gdje razina 1 predstavlja nikakvo oslabljivanje, a 40 maksimalno oslabljivanje kod kojega je rezultat samo jedan heksadekadski znak (4 bita).

Na liniji 81 provjerava se kolika je duljina oslabljenog sažetka („part“) jer se u varijablu „bin“ ne može pospremiti dekadska vrijednost svih 40 znakova. Maksimalna vrijednost je oko $(\pm)2^{64}$, ovisno o sustavu, stoga, ako imamo oslabljenje jednako 2, $\frac{40}{2} = 20$, a 20 heksadekadskih znakova (svaki po 4 bita) je $20 \cdot 4 = 80$, onda možemo očekivati brojeve do $(\pm)2^{80}$, a to je preveliko za varijablu „bin“. Stoga ovdje uvodimo grananje zbog takvih slučajeva.

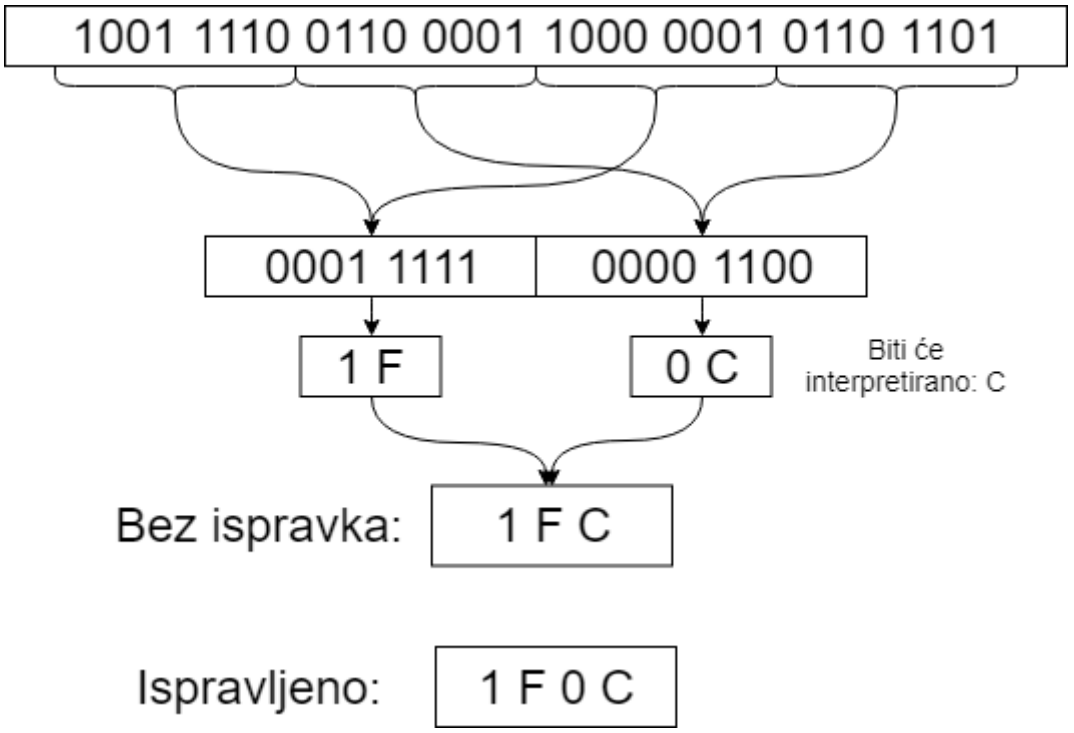
Ako je oslabljenje razine 4 do 40, koristi se prvi slučaj grananja („part“≤10). Preko for petlje se prolazi kroz cijeli 40-znakovni niz pretvarajući, s lijeva na desno, heksadekadske znakove (njih „part“) u decimalni broj „bin“ koristeći „stringstream“. Dobiveni „bin“ se nadalje predaje konstruktoru bitset-a „b“. Time se broj „bin“ pretvori u niz bitova „b“. Potom se XOR logičkim vratima niz bitova „bits“ (inicijalno sve nule) obradi sa nizom bitova „b“. U „bits“ varijabli se sada nalazi prvih „part“ znakova sažetka u binarnom formatu. Zatim se drugih „part“ znakova pretvaraju iz heksadekadskog zapisa u binarni i zapisuju u niz bitova „b“, nadalje se preko XOR vrata obradi „bits“ XOR „b“ i posprema u „bits“, u varijabli „bits“ sada imamo „prvi dio“ XOR „drugi dio“. Postupak se ponavlja dok nije obrađen cijeli sažetak. Točnije svaki dio se XOR vratima obradi sa „bits“ (ako imamo četiri dijela, onda je „bits“ jednako „prvi dio“ XOR „drugi dio“ XOR „treći dio“ XOR „četvrti dio“). Nakon obrade se niz bitova „bits“, koji sad sadrži „oslabljen“ sažetak u binarnom zapisu, pretvara u heksadekadski niz koji funkcija vraća.

Ako je „part“ jednak 40 (nema oslabljenja) funkcija jednostavno vraća sažetak kakav je i primila.

U trećem slučaju (40 > „part“ > 10), moramo koristit drugu metodu, zbog toga što je „part“ sažetka prevelik za varijablu „bin“. Iz kandidata za oslabljivanje sažetka, primjetimo da je jedina razina oslabljivanja koja rezultira ovim slučajem razina 2. Prvo deklariramo prazan

znakovni niz (nakon obrade, tu će pisati oslabljeni sažetak) „weakHash“. Zatim, for petljom (iterira se uvijek dva puta) zaobilazimo problem prevelikog broja. Umjesto da sažetak razdijelimo na dva djela (dio1 + dio2 = sažetak), razdijeliti ćemo ga na četiri. XOR vratima obradimo prvih deset znakova i trećih deset znakova sažetka zapisanih u nizove bitova. Nadalje taj rezultat pretvorimo u heksadekadski niz i dodamo na kraj (inicijalno praznog) znakovnog niza „weakHash“. Postupak se ponovi još jednom ali sada obrađujemo drugih deset i četvrtih deset znakova sažetka, te rezultat dodajemo na kraj znakovnog niza „weakHash“. Time smo dobili oslabljeni sažetak koji funkcija vraća.

Problem može nastati kada, zbog numeričke interpretacije, bude vodećih nula. S obzirom da dodajemo drugi dio oslabljenog sažetka na prvi, ako je drugi dio imao vodeće nule, a one nisu reprezentirane, rezultat neće biti ispravan. To se ispravlja na liniji 109 while (hrv. dok) petljom. Taj problem (sa manje bitova) je reprezentiran na slici 16.



Slika 16: problem spajanja dva dijela sažetka [autorski rad]

4.4.2. WeakenHash64()

Za algoritme (SHA-256, SHA-3, Keccak) koji daju sažetak duljine 64 znakova (256 bitova), koristi se funkcija „WeakenHash64()“ (slika 17).

```
115  std::string WeakenHash64(std::string hash, int alg) {
116      unsigned long long bin;
117      std::bitset<64> bits;
118      int part = 64 / settings[alg];
119      if (part <= 16) {
120          for (int i = 0; i < 64; i += part) {
121              std::stringstream ss;
122              ss << std::hex << hash.substr(i, part);
123              ss >> bin;
124              std::bitset<64> b(bin);
125              bits ^= b;
126          }
127          std::stringstream ss;
128          ss << std::hex << bits.to_ullong();
129          return ss.str();
130      }
131      else if (part == 64) return hash;
132      else {
133          std::string weakHash;
134          for (int i = 0; i < 2; ++i) {
135              std::stringstream ss;
136              ss << std::hex << hash.substr(i * 16, 16);
137              ss >> bin;
138              std::bitset<64> b(bin);
139              ss.str(""); ss.clear();
140              ss << std::hex << hash.substr((i + 2) * 16, 16);
141              ss >> bin;
142              b ^= std::bitset<64>(bin);
143              ss.str(""); ss.clear();
144              ss << std::hex << b.to_ullong();
145              weakHash += ss.str();
146              while (weakHash.length() < (i + 1) * 16) weakHash.insert(i * 16, "0");
147          }
148          return weakHash;
149      }
150  }
```

Slika 17: Funkcija oslabljenja sažetaka duljine 256 bitova [autorski rad]

Princip rada je gotovo identičan kao i kod funkcije „WeakenHash40()“ (slika 16). Kandidati za oslabljenja su 1, 2, 4, 8, 16, 32 i 64. Jedina razina oslabljenja koja bi rezultirala prevelikim brojem za varijablu „bin“ je razina 2 gdje bi mogli dobiti broj 2^{128}. Naravno kod razine oslabljenja 1, oslabljenje se ne vrši, te se samo vraća originalni sažetak.

4.4.3. WeakenHash32() i WeakenHash8()

Funkcije „Weakenhash32()“ (slika 18) koristi se za sažetke duljine 32 znaka, a „WeakenHash8()“ (slika 19) za sažetke duljine 8 znakova. U našem slučaju, MD-5, odnosno, CRC-32. Budući da barem prepolovljen sažetak ovih algoritama sažimanja stane u numeričku varijablu „bin“, ovdje nema komplikacija, te je postupak direktan.

```
152  std::string WeakenHash32(std::string hash, int alg) {
153      unsigned long long bin;
154      std::bitset<64> bits;
155      int part = 32 / settings[alg];
156      if (part == 32) return hash;
157      for (int i = 0; i < 32; i += part) {
158          std::stringstream ss;
159          ss << std::hex << hash.substr(i, part);
160          ss >> bin;
161          std::bitset<64> b(bin);
162          bits ^= b;
163      }
164      std::stringstream ss;
165      ss << std::hex << bits.to_ullong();
166      return ss.str();
167  }
```

Slika 18: Funkcija oslabljenja sažetaka duljine 128 bitova [autorski rad]

```
169  std::string WeakenHash8(std::string hash, int alg) {
170      long long bin;
171      std::bitset<32> bits;
172      int part = 8 / settings[alg];
173      if (part == 8) return hash;
174      for (int i = 0; i < 8; i += part) {
175          std::stringstream ss;
176          ss << std::hex << hash.substr(i, part);
177          ss >> bin;
178          std::bitset<32> b(bin);
179          bits ^= b;
180      }
181      std::stringstream ss;
182      ss << std::hex << bits.to_ullong();
183      return ss.str();
184  }
```

Slika 19: Funkcija oslabljenja sažetaka duljine 32 bita [autorski rad]

5. Analiza rezultata kolizija

Analize su izvođene za SHA-1, SHA-256 i SHA-3 algoritme sažimanja sa raznim razinama oslabljenja sažetka, a rezultati su slijedeći:

5.1. Analize prve pojave kolizije

1. **Eksperiment**: generirano 100,000 alfanumeričkih poruka, svaka veličine 128 bajta. Svaka poruka je sažimana algoritmima SHA-1, SHA-256 i SHA-3 čiji su sažetci oslabljeni na 8 bajta. Za te sažetke su se tražile kolizije, te je praćeno koliko se poruka mora generirati da bi došlo do prve kolizije. Postupak je ponovljen 100 puta i iz dobivenih podataka svakog od 100 setova izračunate su vrijednosti aritmetičke sredine i medijana. U tablici 1 su prikazani osnovni podatci o eksperimentu i dobiveni rezultati.

Tablica 1: Rezultat generiranja 100,000 alfanumeričkih poruka veličine 128 bajta 100 puta.

Algoritam	*Oslabljenje*	*Duljina sažetka*	*Aritmetička sredina*	*Medijan*
SHA-1	5	8 bajta	-	55257
SHA-256	8	8 bajta	-	58116
SHA-3	8	8 bajta	-	65968

   - Aritmetičku sredinu nije moguće izračunati jer nije došlo do kolizije u barem jednom od sto setova od 100,000 poruka.

2. **Eksperiment**: generirano 100,000 poruka generiranih od ASCII znakova, svaka veličine 128 bajta. Svaka poruka je sažimana algoritmima SHA-1, SHA-256 i SHA-3 čiji su sažetci oslabljeni na 8 bajta. Za te sažetke su se tražile kolizije, te je praćeno koliko se poruka mora generirati da bi došlo do prve kolizije. Postupak je ponovljen 100 puta i iz dobivenih podataka svakog od 100 setova izračunate su vrijednosti aritmetičke sredine i medijana. U tablici 2 su prikazani osnovni podatci o eksperimentu i dobiveni rezultati.

Tablica 2: Rezultat generiranja 100,000 poruka (svi ASCII) veličine 128 bajta 100 puta.

Algoritam	*Oslabljenje*	*Duljina sažetka*	*Aritmetička sredina*	*Medijan*
SHA-1	5	8 bajta	-	63317
SHA-256	8	8 bajta	-	62282
SHA-3	8	8 bajta	-	60048

- Razlike su postojane u odnosu na prvi eksperiment (druga vrsta znakova kojima su generirane poruke), ali zbog malog broja testiranja (100) možemo ih smatrati zanemarivima.

3. **Eksperiment**: generirano 500,000 alfanumeričkih poruka, svaka veličine 128 bajta. Svaka poruka je sažimana algoritmima SHA-1, SHA-256 i SHA-3 čiji su sažetci oslabljeni: SHA-1 na 10 bajta, SHA-256 i SHA-3 na 16 bajta. Za te sažetke su se tražile kolizije, te je praćeno koliko se poruka mora generirati da bi došlo do prve kolizije. Postupak je ponovljen 18 puta i iz dobivenih podataka svakog od 18 setova izračunate su vrijednosti aritmetičke sredine i medijana. U tablici 3 su prikazani osnovni podatci o eksperimentu i dobiveni rezultati.

Tablica 3: Rezultat generiranja 500,000 alfanumeričkih poruka veličine 128 bajta 18 puta.

Algoritam	*Oslabljenje*	*Duljina sažetka*	*Aritmetička sredina*	*Medijan*
SHA-1	4	10 bajta	-	353031
SHA-256	4	16 bajta	-	-
SHA-3	4	16 bajta	-	-

- Za SHA-256 i SHA-3 sažetke u svakom od 18 setova nije pronađena niti jedna kolizija.
- SHA-1 sažetak oslabljen za razinu 4, ima sažetak duljine 10 znakova (16^{10} različitih kombinacija). Vjerojatnost pojave kolizije u prvih 500,000 poruka je ~10.75%. Izračunato [8]:

$$k = 16^{10}$$

$$n = 500{,}000$$

$$x = \frac{-n^2}{2 \cdot k} = \frac{-500{,}000^2}{2 \cdot 16^{10}} = -0.11368683772$$

$$q = e^{-0.11368683772} = 0.89253742117$$

$$p = 1 - q = 1 - 0.89253742117 = 0.10746257882$$

$$vjerojatnost = 10.746257882\%$$

- Ponovljen je eksperiment sa parametrima boljim za analizu: generirano 500,000 alfanumeričkih poruka, svaka veličine 128 bajta. Svaka poruka je sažimana algoritmom SHA-1 čiji su sažetci oslabljeni na 10 bajta. Za te sažetke su se tražile kolizije, te je praćeno koliko se poruka mora generirati da bi došlo do prve kolizije. Postupak je ponovljen 180 puta i iz dobivenih podataka svakog od 180 setova izračunate su vrijednosti aritmetičke sredine. Na slici 20 su

prikazani osnovni podatci o eksperimentu (izuzev oslabljenja sažetaka i tipa i veličine poruka) i dobiveni rezultati.

```
There were 500000 messages generated 180 times (total: 90000000)
Average amount of generated messages needed to find a collision:
SHA1
Arithmetic mean: can not calculate.
```

Slika 20: Rezultat generiranja 500,000 poruka 180 puta (SHA-1: 10 bajta) [autorski rad]

Od 180 setova, u 20 setova je pronađena kolizija, to je prikazano na slici 21, gdje je prikazan zadnji set (set 20). $\frac{20}{190} = 0.111111$ što je 11.11%. To znači da je vjerojatnost pojave kolizije unutar prvih 500,000 poruka 11.11% što odgovara dobivenom rezultatu iz točke 2 ove liste.

```
20. set:
        50000: 0(0)
        100000: 0(0)
        150000: 0(0)
        200000: 0(0)
        250000: 0(0)
        300000: 0(0)
        350000: 1(1)
        400000: 2(1)
        450000: 2(0)
        500000: 2(0)
```

Slika 21: Zadnji set sa kolizijom od 180 setova [autorski rad]

- Kako bismo za SHA-1 sažetak oslabljen za 4 razine našli koliziju, dobro bi bilo generirati 1,500,000 poruka. Onda je vjerojatnost prve kolizije ~64%.
- Prateći metodu računanja; SHA-256 i SHA-3 sažetci imaju oslabljenje razine 4, što je sažetak duljine 16 znakova. Vjerojatnost kolizije u prvih 500,000 poruka je 6.77626355e-7%. Sa milijardom poruka, šanse za kolizijom su ~2.67%.

4. **Eksperiment**: generirano 10,000 alfanumeričkih poruka, svaka veličine 128 bajta. Svaka poruka je sažimana algoritmima SHA-1, SHA-256 i SHA-3 čiji su sažetci oslabljen: SHA-1 na 5 bajta, SHA-256 i SHA-3 na 4 bajta. Za te sažetke su se tražile kolizije, te je praćeno koliko se poruka mora generirati da bi došlo do prve kolizije. Postupak je ponovljen 1,200 puta i iz dobivenih podataka svakog od 1,200 setova

izračunate su vrijednosti aritmetičke sredine i medijana. U tablici 4 su prikazani osnovni podatci o eksperimentu i dobiveni rezultati.

Tablica 4: Rezultat generiranja 10,000 alfanumeričkih poruka veličine 128 bajta 1,200 puta.

Algoritam	*Oslabljenje*	*Duljina sažetka*	*Aritmetička sredina*	*Medijan*
SHA-1	8	5 bajta	1266.039167	1197
SHA-256	16	4 bajta	321.31	292
SHA-3	16	4 bajta	320.860833	302.5

- Broj pokušaja za oslabljene SHA-256 i SHA-3 sažetke je otprilike jednak (oko 300), što je vidljivo iz aritmetičkih sredina i medijana prve pojave kolizija. To je vjerojatno jer im je sažetak jednake duljine (4 bajta), a za oslabljene SHA-1 sažetke je potrebno malo više pokušaja (oko 1,200).
- Zbog kratkih duljina sažetka, kolizije nije teško naći, što je i vidljivo iz rezultata.

5. **Eksperiment**: generirano 10,000 alfanumeričkih poruka, svaka veličine 128 bajta. Svaka poruka je sažimana algoritmima SHA-1, SHA-256 i SHA-3 čiji su sažetci oslabljen: SHA-1 na 5 bajta, SHA-25 i SHA-3 na 4 bajta. Za te sažetke su se tražile kolizije, te je praćeno koliko se poruka mora generirati da bi došlo do prve kolizije. Postupak je ponovljen 1,200 puta i iz dobivenih podataka svakog od 1,200 setova izračunate su vrijednosti aritmetičke sredine i medijana. U tablici 5 su prikazani osnovni podatci o eksperimentu i dobiveni rezultati.

Tablica 5: Rezultat generiranja 10,000 poruka (svi ASCII) veličine 128 bajta 1,200 puta.

Algoritam	*Oslabljenje*	*Duljina sažetka*	*Aritmetička sredina*	*Medijan*
SHA-1	8	5 bajta	1272.649167	1184.5
SHA-256	16	4 bajta	316.826667	295
SHA-3	16	4 bajta	316.336667	298

- Rezultati su slični kao i u četvrtom eksperimentu, stoga mogli bismo zaključiti kako sadržaj nasumično generiranih poruka, nema utjecaja na pojavu prve kolizije.

6. **Eksperiment**: generirano 500,000 alfanumeričkih poruka, svaka veličine 64 bajta. Svaka poruka je sažimana algoritmima SHA-1, SHA-256 i SHA-3 čiji su sažetci oslabljeni na 8 bajta. Za te sažetke su se tražile kolizije, te je praćeno koliko se poruka mora generirati da bi došlo do prve kolizije. Postupak je ponovljen 20 puta i iz dobivenih

podataka svakog od 20 setova izračunate su vrijednosti aritmetičke sredine i medijana. U tablici 6 su prikazani osnovni podatci o eksperimentu i dobiveni rezultati.

Tablica 6: Rezultat generiranja 500,000 alfanumeričkih poruka veličine 64 bajta 20 puta.

Algoritam	*Oslabljenje*	*Duljina sažetka*	*Aritmetička sredina*	*Medijan*
SHA-1	5	8 bajta	75561.166667	75321.5
SHA-256	8	8 bajta	84263.033333	76313
SHA-3	8	8 bajta	81255.65	75320.5

- Zbog razlike u rezultatu SHA-1 i ostalih algoritama, za SHA-1 je generirano 360 puta 500,000 poruka (veličine oslabljenog sažetka i poruka su ostale iste) i dobiveni su slijedeći rezultati:
    - Aritmetička sredina: 84319.655556
    - Medijan: 80747

  Što je bliže druga dva algoritma (što se tiče aritmetičke sredine). S obzirom da su duljine oslabljenih sažetaka ova tri algoritma sažimanja identične, ovakav rezultat je i očekivan.

7. **Eksperiment**: generirano 500,000 poruka generiranih od ASCII znakova, svaka veličine 64 bajta. Svaka poruka je sažimana algoritmima SHA-1, SHA-256 i SHA-3 čiji su sažetci oslabljeni na 8 bajta. Za te sažetke su se tražile kolizije, te je praćeno koliko se poruka mora generirati da bi došlo do prve kolizije. Postupak je ponovljen 20 puta i iz dobivenih podataka svakog od 20 setova izračunate su vrijednosti aritmetičke sredine i medijana. U tablici 7 su prikazani osnovni podatci o eksperimentu i dobiveni rezultati.

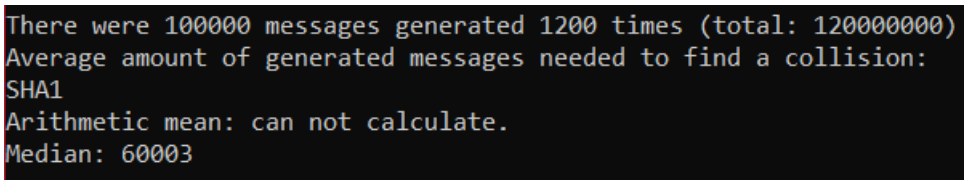
Tablica 7: Rezultat generiranja 500,000 poruka (svi ASCII) veličine 64 bajta 20 puta.

Algoritam	*Oslabljenje*	*Duljina sažetka*	*Aritmetička sredina*	*Medijan*
SHA-1	5	8 bajta	83957.65	69207.5
SHA-256	8	8 bajta	68479.15	66649.5
SHA-3	8	8 bajta	81845.4	83880

- Dobiveni rezultati su u skladu sa eksperimentom 6, te nadalje potvrđuju kako sadržaj nasumično generirane poruke nema utjecaja na pojavu prve kolizije.

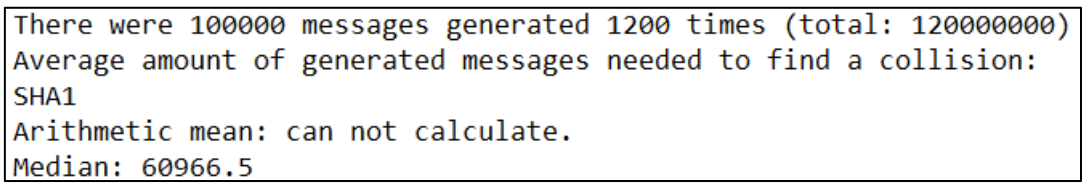
8. Proučavanje utjecaja veličine sažimane poruke na pojavu prve kolizije za SHA-1:
    8. **Eksperiment**: generirano 100,000 alfanumeričkih poruka, svaka veličine 64 bajta. Svaka poruka je sažimana algoritmom SHA-1, a dobiveni sažetci

su oslabljeni na 8 bajta. Za te sažetke su se tražile kolizije, te je praćeno koliko se poruka mora generirati da bi došlo do prve kolizije. Postupak je ponovljen 1,200 puta i iz dobivenih podataka svakog od 1,200 setova izračunate su vrijednosti aritmetičke sredine i medijana. Na slici 22 su prikazani osnovni podatci o eksperimentu (izuzev oslabljenja sažetaka i tipa i veličine poruka) i dobiveni rezultati.



Slika 22: (SHA-1) rezultat generiranja 100,000 poruka veličine 64 bajta 1,200 puta [autorski rad]

9. **Eksperiment**: generirano 100,000 alfanumeričkih poruka, svaka veličine 512 bajta. Svaka poruka je sažimana algoritmom SHA-1, a dobiveni sažetci su oslabljeni na 8 bajta. Za te sažetke su se tražile kolizije, te je praćeno koliko se poruka mora generirati da bi došlo do prve kolizije. Postupak je ponovljen 1,200 puta i iz dobivenih podataka svakog od 1,200 setova izračunate su vrijednosti aritmetičke sredine i medijana. Na slici 23 su prikazani osnovni podatci o eksperimentu (izuzev oslabljenja sažetaka i tipa i veličine poruka) i dobiveni rezultati.



Slika 23: (SHA-1) rezultat generiranja 100,000 poruka veličine 512 bajta 1,200 puta [autorski rad]

- Razlika u rezultatima 8. i 9. eksperimenta je minimalna, stoga veličina poruka koje se sažimaju nije faktor kod pronalaska prve kolizije oslabljenih SHA-1 sažetaka.

9. Proučavanje utjecaja veličine poruka na pojavu prve kolizije za SHA-256 i SHA-3.
    10. **Eksperiment**: generirano 100,000 alfanumeričkih poruka, svaka veličine 64 bajta. Svaka poruka je sažimana algoritmom SHA-256, a dobiveni sažetci su oslabljeni na 8 bajta. Za te sažetke su se tražile kolizije, te je praćeno koliko se

poruka mora generirati da bi došlo do prve kolizije. Postupak je ponovljen 1,200 puta i iz dobivenih podataka svakog od 1,200 setova izračunate su vrijednosti aritmetičke sredine i medijana. Na slici 24 su prikazani osnovni podatci o eksperimentu (izuzev oslabljenja sažetaka) i dobiveni rezultati.

```
There were 100000 messages generated 1200 times (total: 120000000)
Average amount of generated messages needed to find a collision:
SHA2
Arithmetic mean: can not calculate.
Median: 59136
```

Slika 24: (SHA-256) rezultat generiranja 100,000 poruka veličine 64 bajta 1,200 puta [autorski rad]

11. **Eksperiment**: generirano 100,000 alfanumeričkih poruka, svaka veličine 512 bajta. Svaka poruka je sažimana algoritmom SHA-256, a dobiveni sažetci su oslabljeni na 8 bajta. Za te sažetke su se tražile kolizije, te je praćeno koliko se poruka mora generirati da bi došlo do prve kolizije. Postupak je ponovljen 1,200 puta i iz dobivenih podataka svakog od 1,200 setova izračunate su vrijednosti aritmetičke sredine i medijana. Na slici 25 su prikazani osnovni podatci o eksperimentu (izuzev oslabljenja sažetaka i tipa i veličine poruka) i dobiveni rezultati.

```
There were 100000 messages generated 1200 times (total: 120000000)
Average amount of generated messages needed to find a collision:
SHA2
Arithmetic mean: can not calculate.
Median: 59734
```

Slika 25: (SHA-256) rezultat generiranja 100,000 poruka veličine 512 bajta 1,200 puta [autorski rad]

12. **Eksperiment**: generirano 100,000 alfanumeričkih poruka, svaka veličine 64 bajta. Svaka poruka je sažimana algoritmom SHA-3, a dobiveni sažetci su oslabljeni na 8 bajta. Za te sažetke su se tražile kolizije, te je praćeno koliko se poruka mora generirati da bi došlo do prve kolizije. Postupak je ponovljen 1,200 puta i iz dobivenih podataka svakog od 1,200 setova izračunate su vrijednosti aritmetičke sredine i medijana. Na slici 26 su prikazani osnovni podatci o eksperimentu (izuzev oslabljenja sažetaka i tipa i veličine poruka) i dobiveni rezultati.

```
There were 100000 messages generated 1200 times (total: 120000000)
Average amount of generated messages needed to find a collision:
SHA3
Arithmetic mean: can not calculate.
Median: 59505
```

Slika 26: (SHA-3) rezultat generiranja 100,000 poruka veličine 64 bajta 1,200 puta [autorski rad]

13. **Eksperiment**: generirano 100,000 alfanumeričkih poruka, svaka veličine 512 bajta. Svaka poruka je sažimana algoritmom SHA-3, a dobiveni sažetci su oslabljeni na 8 bajta. Za te sažetke su se tražile kolizije, te je praćeno koliko se poruka mora generirati da bi došlo do prve kolizije. Postupak je ponovljen 1,200 puta i iz dobivenih podataka svakog od 1,200 setova izračunate su vrijednosti aritmetičke sredine i medijana. Na slici 27 su prikazani osnovni podatci o eksperimentu (izuzev oslabljenja sažetaka i tipa i veličine poruka) i dobiveni rezultati.

```
There were 100000 messages generated 1200 times (total: 120000000)
Average amount of generated messages needed to find a collision:
SHA3
Arithmetic mean: can not calculate.
Median: 61256
```

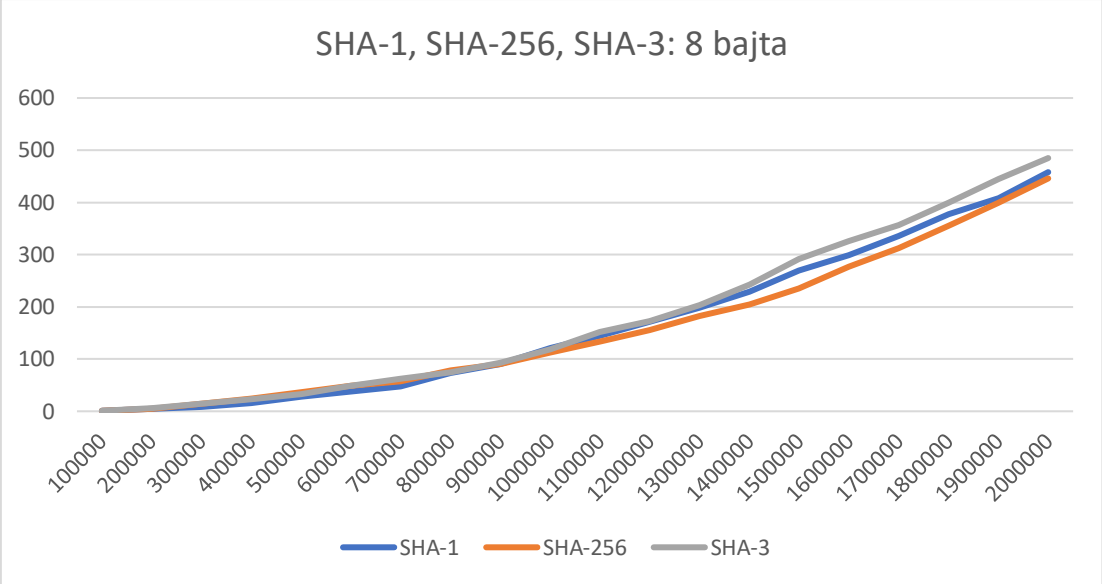
Slika 27: (SHA-3) rezultat generiranja 100,000 poruka veličine 512 bajta 1,200 puta [autorski rad]

- Iz dobivenih rezultata, zaključujemo kako veličina poruke koja se sažima ne utječe na pojavu prve kolizije, neovisno o algoritmu sažimanja (kada su sažetci oslabljeni na jednake duljine). Rezultati sva tri algoritma sažimanja (SHA-1 iz točke 8. i SHA-256 i SHA-3 iz ove točke) su podjednaka.

5.2. Analize rasta broja kolizija

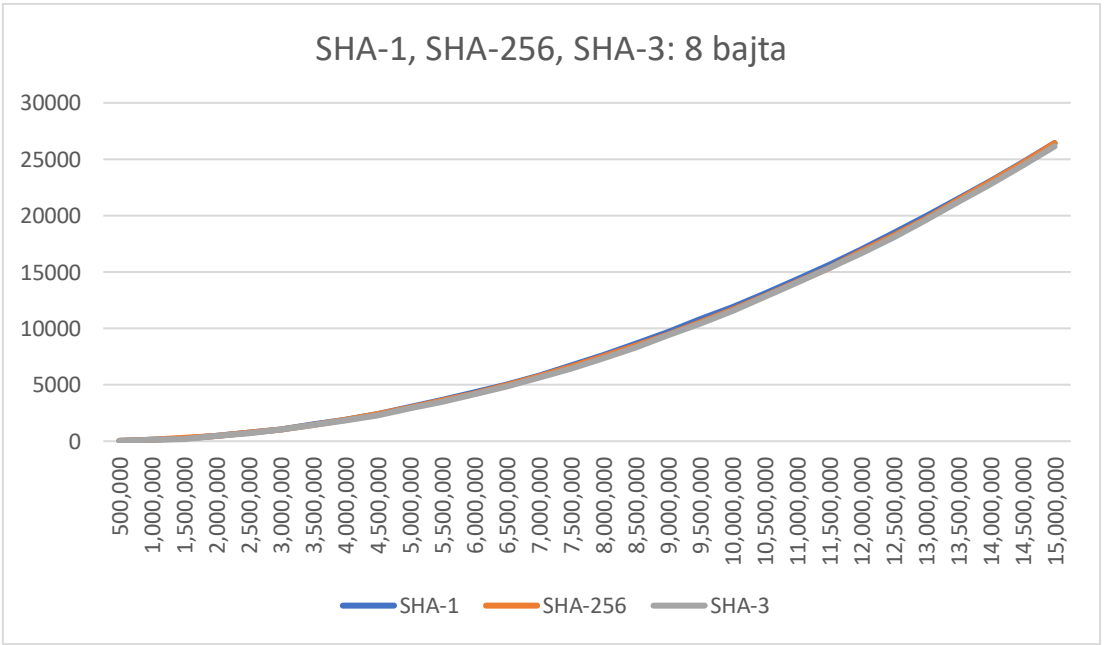
Na slijedećim grafovima je prikaz rasta broja kolizija (os apscisa) u odnosu na rast broja poruka (os ordinata).

1. **Generiranje**: generirano je 2,000,000 poruka koja je svaka sažimana algoritmima SHA-1, SHA-256 i SHA-3, a sažetci su oslabljeni na 8 bajta. Tokom generiranja se pratio porast kolizija porastom broja poruka. Rezultat je prikazan na slici 28.



Slika 28: graf broja kolizija sažetaka duljine 8 bajta u 2,000,000 poruka [autorski rad]

2. **Generiranje**: generirano je 15,000,000 poruka koja je svaka sažimana algoritmima SHA-1, SHA-256 i SHA-3, a sažetci su oslabljeni na 8 bajta. Tokom generiranja se pratio porast kolizija porastom broja poruka. Rezultat je prikazan na slici 29.

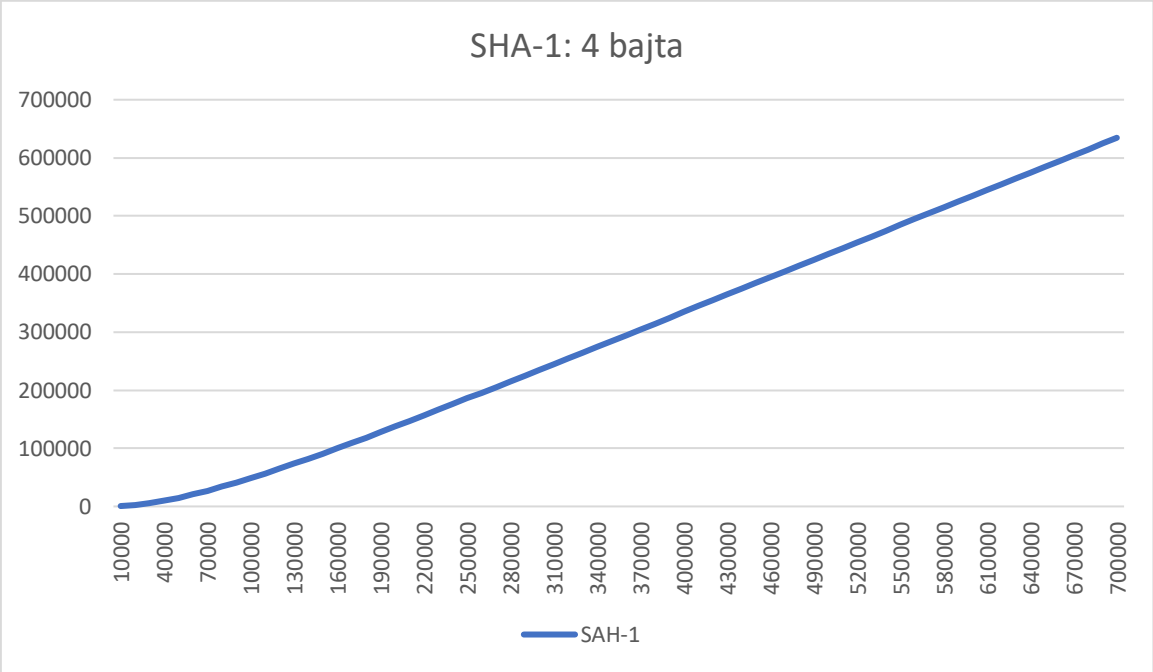


Slika 29: graf broja kolizija sažetaka duljine 8 bajta u 15,000,000 poruka [autorski rad]

- Iz rezultata je lako uočljivo ubrzanje rasta rastom broja poruka. S vremenom će doći do zasićenja, to jest, graf postaje linearan. To nastaje kada se generiraju poruke koje, u kolektivu, daju svaku moguću kombinaciju sažetaka.

Time će svaka nova poruka rezultirati kolizijom. Takav rezultat vidimo kod slijedećeg generiranja.

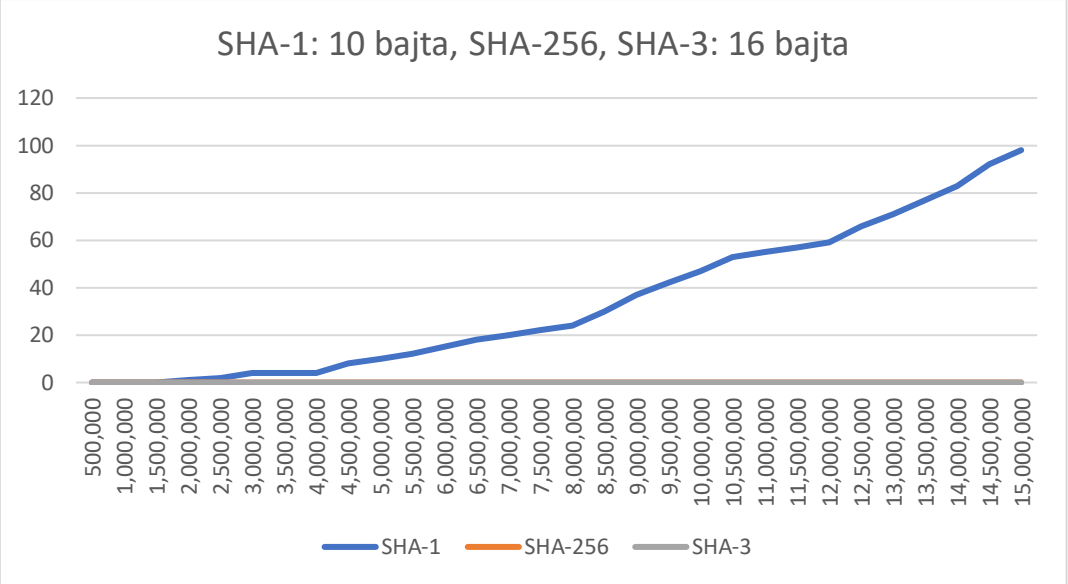
3. **Generiranje**: generirano je 700,000 poruka koja je svaka sažimana algoritmom SHA-1, a sažetci su oslabljeni na 4 bajta. Tokom generiranja se pratio porast kolizija porastom broja poruka. Rezultat je prikazan na slici 30.



Slika 30: graf broja kolizija SHA-1 sažetka duljine 4 bajta u 700,000 poruka [autorski rad]

- Ubrzo su generirani svi mogući sažetci duljine 4 bajta, zbog toga svaka nova poruka rezultira kolizijom, pa graf postaje linearan.

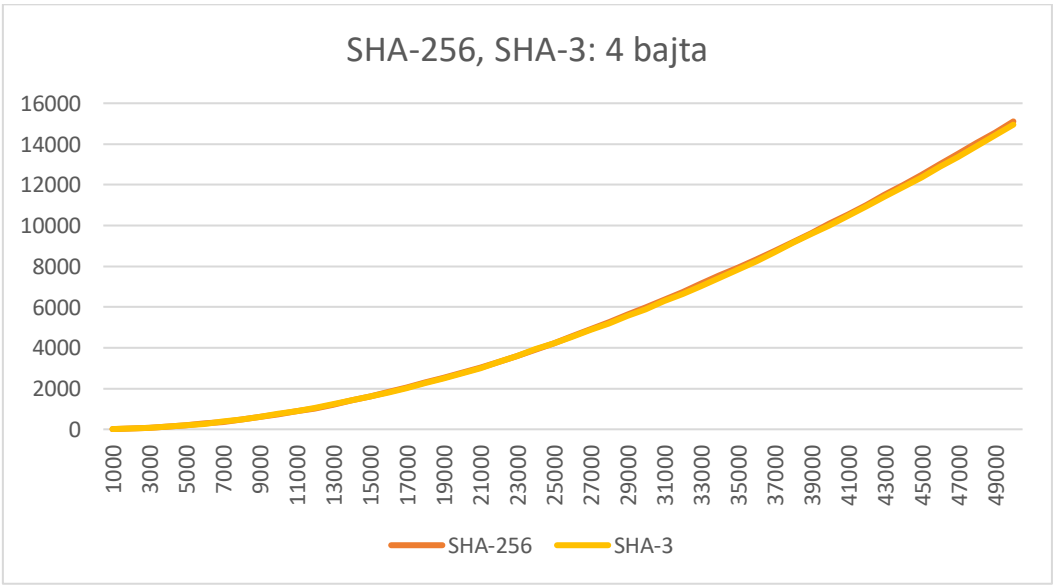
4. **Generiranje**: generirano je 15,000,000 poruka koja je svaka sažimana algoritmima SHA-1, SHA-256 i SHA-3, a sažetci su oslabljeni: SHA-1 na 10 bajta, SHA-256 i SHA-3 na 16 bajta. Tokom generiranja se pratio porast kolizija porastom broja poruka. Rezultat je prikazan na slici 31.



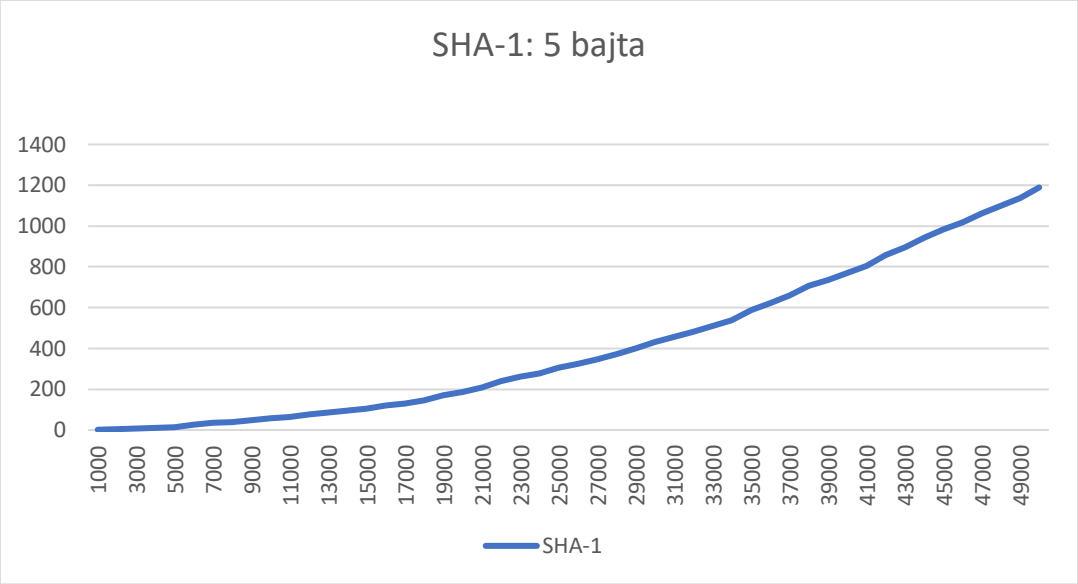
Slika 31: graf broja kolizija slabo oslabljenih sažetaka u 15,000,000 poruka [autorski rad]

- Nakon 15,000,000 poruka za SHA-1 sažetke (10 bajta) je pronađeno manje od 100 kolizija, dok za SHA-256 i SHA-3 (16 bajta) sažetke nije pronađena niti jedna.

5. **Generiranje**: generirano je 50,000 poruka koja je svaka sažimana algoritmima SHA-1, SHA-256 i SHA-3, a sažetci su oslabljeni: SHA-1 na 5 bajta, SHA-256 i SHA-3 na 4 bajta. Tokom generiranja se pratio porast kolizija porastom broja poruka. Rezultat generiranja za SHA-1 sažetke je prikazan na slici 31, a za SHA-256 i SHA-3 na slici 32.



Slika 32: graf broja kolizija SHA-256 i SHA-3 sažetaka oslabljenih na 4 bajta u 50,000 poruka [autorski rad]



Slika 33: graf broja kolizija SHA-1 sažetaka oslabljenih na 5 bajta u 50,000 poruka. [autorski rad]

5.3. Dinamičke poruke sa istim sažetkom

Kao predložak ćemo koristiti:

- Prvi dio: „Stanje racuna: „.
- Drugi dio: „HRK.“.
- Dinamički dio između prvog i drugog dijela: minimalno 2, maksimalno 5 (ne računajući minus) znakova.

1. **Generiranje**: generirano je 199,978 poruka sa zadanim predloškom. Poruke su sažimane algoritmom SHA-1, a sažetci oslabljeni na 8 bajta. Praćeni su sažetci koji su imali kolizije i rezultati su u tablici 8. Pronađena su četiri sažetka koja su imala kolizije.

Tablica 8: kolizije SHA-1 sažetaka oslabljenih na 8 bajta.

Sažetak	**Poruka**
54d9ae20	Stanje racuna: 59276HRK.
	Stanje racuna: 79489HRK.
43f0ad1c	Stanje racuna: -69053HRK.
	Stanje racuna: 79990HRK.
fccfc72b	Stanje racuna: -76107HRK.
	Stanje racuna: 30700HRK.
7802f3b0	Stanje racuna: -49216HRK.
	Stanje racuna: 49310HRK.

2. **Generiranje**: generirano je 199,978 poruka sa zadanim predloškom. Poruke su sažimane algoritmom SHA-256, a sažetci oslabljeni na 8 bajta. Praćeni su sažetci koji su imali kolizije i rezultati su u tablici 9. Pronađena su šest sažetka koja su imala kolizije.

Tablica 9: kolizije SHA-256 sažetaka oslabljenih na 8 bajta

Sažetak	Poruka
b4d0527d	Stanje racuna: 4415HRK.
	Stanje racuna: -69558HRK.
6773a161	Stanje racuna: -83588HRK.
	Stanje racuna: 73502HRK.
313808db	Stanje racuna: -83349HRK.
	Stanje racuna: -80878HRK.
84060aac	Stanje racuna: 11879HRK.
	Stanje racuna: -16986HRK.
e10f834e	Stanje racuna: 90719HRK.
	Stanje racuna: -52337HRK.
677e4335	Stanje racuna: -99315HRK.
	Stanje racuna: 56502HRK.

3. **Generiranje**: generirano je 199,978 poruka sa zadanim predloškom. Poruke su sažimane algoritmom SHA-3, a sažetci oslabljeni na 8 bajta. Praćeni su sažetci koji su imali kolizije i rezultati su u tablici 10. Pronađena su četiri sažetka koja su imala kolizije. Slijedeća četiri sažetka su imala kolizije:

Tablica 10: kolizije SHA-3 sažetaka oslabljenih na 8 bajtova.

Sažetak	Poruka
a691d7f8	Stanje racuna: 64424HRK.
	Stanje racuna: 64170HRK.
b47f862f	Stanje racuna: -83545HRK.
	Stanje racuna: 67731HRK.
5c695c84	Stanje racuna: 76498HRK.
	Stanje racuna: 55503HRK.
cbdefe92	Stanje racuna: 81784HRK.
	Stanje racuna: 33385HRK.

S obzirom da nema previše različitih poruka , točnije, razlika je samo u dinamičkom dijelu, mogući su samo dinamički dijelovi u rasponu od -99999 do 99999, s time da moraju

biti uvijek minimalno 2 broja, točnije ne može biti broj manji od 10 i veći od -10 (vodeća znamenka ne može biti nula). Formula:

$$BrojKombinacija = 2\left(10^{maks} - 10^{min-1} - 1\right)$$

U ovom slučaju:

$$maks = 5, min = 2$$
$$BrojKombinacija = 2(10^5 - 10^{2-1} - 1) = 199{,}978$$

Zbog toga, ako je u konfiguraciji zadan preveliki broj generiranja poruka, aplikacija „odbija“ duplikate, traženje kolizija se neće moći izvršiti do kraja (beskonačna petlja).

6. Zaključak

Izrađenom aplikacijom nije preporučeno tražiti kolizije sažetaka duljine veće od 10 znakova, dok je moguće pronaći kolizije i za takve sažetke, zahtjevno je za memoriju (pogotovo ako su poruke veće) i vrijeme.

Razlike između SHA-1, SHA-256 i SHA-3 oslabljenog sažetka reduciranog na 8 znakova su najviše promatrane. Dolazim do zaključka kako algoritam sažimanja nema ulogu u pojavi kolizija kada mu je sažetak oslabljen na istu duljinu kao i sažetak drugog algoritma sažimanja. Preciznije, sažetci sva tri algoritma sažimanja oslabljeni na istu duljinu, imaju gotovo identičnu pojavu prve kolizije, a krivulja rasta kolizija rastom broja generiranih poruka je gotovo identična. Prema tome, ako pretpostavimo da ovakvi oslabljeni sažetci i dalje imaju smisla u stvarnom svijetu (sažetci oslabljeni na ovaj način u praksi ne postoje), najbolja metoda zaštite od kolizija je duži sažetak (barem od traženja kolizija ovom metodom; metodom sirove snage).

Veličina generiranih poruka i vrsta algoritma sažimanja kojima se poruke sažimaju (akko su sažetci oslabljeni na istu duljinu), nemaju utjecaja na pojavu prve kolizije, već će do prve kolizije doći nakon približno jednakog broja generiranih poruka.

Kod poruka sa dinamičkim dijelom, pronađena je nekolicina poruka koje imaju iste sažetke (ne više od dvije poruke po sažetku). Očekivano, kolizije su se ponavljale iz testiranja u testiranje. Broj različitih poruka je u tom rasponu je 199,980, stoga, ne radi se o prevelikom broju različitih poruka.

Alfanumeričke poruke i poruke sa svim ASCII znakovima, imaju gotovo identične rezultate, tako da tip poruke (znakovi od kojih je poruka izgrađena) nema utjecaja na pojave kolizija.

Količina kolizija ovisi o duljini sažetka i broju generiranih poruka. Kraćim sažetcima je puno lakše pronaći kolizije, a rastom broja poruka eksponencijalno raste broj kolizija dok se ne pronađu svi mogući sažetci, gdje dalje broj kolizija raste linearno, preciznije, svaka poruka će rezultirati kolizijom.

Popis literature

[1] „Cryptographic Hash Functions Explained: A Beginner's Guide“ (14.08.2018.). Komodo [Na internetu] Dostupno na: https://komodoplatform.com/cryptographic-hash-function/ [pristupano 15.09.2019.]

[2] „Understanding the Birthday Paradox“ (bez dat.). Better Explained [Na internetu]. Dostupno na: https://betterexplained.com/articles/understanding-the-birthday-paradox/ [pristupano 16.09.2019.]

[3] „SHAttered“ (bez dat.). SHAttered [Na internetu]. Dostupno na: https://shattered.io/ [pristupano16.09.2019.]

[4] J. Liang, X.-J. Lai „Improved Collision Attack on Hash Function MD5“ (14.02.2007.) [Na internetu]. Dostupno na: https://link.springer.com/article/10.1007/s11390-007-9010-1 [pristupano 16.09.2019.]

[5] M. Stevens „Fast Collision Attack on MD5“. (2016.) [Na internetu]. Dostupno na: http://crppit.epfl.ch/documentation/Hash_Function/Examples/Code_Project/Documentation/104.pdf [pristupano 16.09.2019.]

[6] M. Stevens „Single-block collision attack on MD5“ (2012.) [Na internetu]. Dostupno na: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.400.7023&rep=rep1&type=pdf [pristupano 16.09.2019.]

[7] Z. Li, H. Jiang, C. Li „Collision attack on NaSHA-384/512“ (19.07.2010.) [Na internetu]. Dostupno na: https://ieeexplore.ieee.org/document/5508519 [pristupano 16.09.2019.]

[8] „Birthday problem for cryptographic hashing, 101.“ (27.08.2016.) [Na internetu]. Dostupno na: https://crypto.stackexchange.com/questions/39641/what-are-the-odds-of-collisions-for-a-hash-function-with-256-bit-output. [pristupano 15.09.2019.]

Popis slika

Popis tablica