Primjena kriptografskih hash funkcija pri očuvanju integriteta podataka

Marić, Mislav

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

PRIMJENA KRIPTOGRAFSKIH *HASH* FUNKCIJA PRI OČUVANJU INTEGRITETA PODATAKA

Diplomski rad

Mislav Marić

Osijek, 2019.

Sadržaj

1. UVOD 1

2. RAČUNALNA SIGURNOST 2

2.1. Sigurnosne usluge 3

2.2. Sigurnosni mehanizmi 4

2.3. Model mrežne sigurnosti 5

3. KRIPTOGRAFSKE *HASH* FUNKCIJE 7

3.1. Klasifikacija *hash* funkcija 8

3.2. Konstrukcija *hash* funkcija 10

3.3. Jednostavne *hash* funkcije 12

3.4. Sigurnosni zahtjevi kriptografskih *hash* funkcija 13

3.5. Napadi na *hash* funkcije 15

3.6. Primjena *hash* funkcija 17

3.7. Sigurnosni *hash* algoritam (SHA) 21

3.8. Kodovi za autentifikaciju poruke (MAC) 28

3.8.1. Sigurnost MAC kodova 30

3.8.2. MAC kodovi temeljeni na *hash* funkcijama (HMAC) 31

3.8.3. Sigurnost HMAC koda 34

4. PRAKTIČNI PRIMJERI PRIMJENE *HASH* FUNKCIJA 35

4.1. Primjeri primjene *Cryptool 1.4.41* aplikacije pri određivanju *hash* funkcija 36

4.1.1. Prvi primjer - Rad sa *hash* funkcijama 37

4.1.2. Drugi primjer - Rad sa HMAC funkcijama 41

4.1.3. Treći primjer - Digitalni potpis 43

4.2. Primjeri primjene *Cryptool 2.1* aplikacije pri određivanju *hash* funkcija 52

4.2.1. Četvrti primjer - Rad sa *hash* funkcijama 52

4.2.2. Peti primjer - Rad sa HMAC kodovima 55

5. ANALIZA REZULTATA 58

6. ZAKLJUČAK 62

LITERATURA 63

SAŽETAK 64

ABSTRACT 64

ŽIVOTOPIS 65

1. UVOD

U današnje vrijeme kada primjena različitih tehnoloških rješenja i društvenih medija postaje sve značajnija, nužno je očuvanje integriteta, tajnosti i autentičnosti podataka. Razmjena povjerljivih podataka, primjerice pri provedbi financijskih transakcija ili pri slanju elektroničke pošte, zahtijeva visok stupanj zaštite tih podataka, osobito kada se njihova razmjena provodi putem nesigurnih komunikacijskih kanala. Zbog prethodno navedenih razloga, nastala je potreba za primjenom kriptografskih rješenja čiji je cilj omogućiti sigurnu razmjenu informacija koje se razmjenjuju putem nesigurnih komunikacijskih kanala. Iako za veliki broj poznatih sigurnosnih prijetnji postoje odgovarajuća kriptografska rješenja koja povećavaju sigurnost, potreba za novim rješenjima i poboljšanjima postojećih raste svakodnevno. Cilj ovog rada je dati uvid u postojeće kriptografske tehnike i algoritme te prikazati osnovna načela koja se primjenjuju u području kriptografije i mrežne sigurnosti. Rad se fokusira na dva temeljna područja: primjenu kriptografskih *hash* funkcija te povećanje razine sigurnosti pri prijenosu podataka kroz komunikacijsku mrežu zasnovano na primjeni implementiranih kriptografskih rješenja. Kriptografske *hash* funkcije mogu se primijeniti u brojnim postojećim kriptografskim sustavima i alatima. Idealne *hash* funkcije se ponašaju kao funkcije koje za svaki novi ulaz daju slučajne izlaze, no u praksi to nije tako. U ovom radu dan je prikaz osnovnih problema zbog kojih se primjenjuju kriptografski postupci, pri čemu su objašnjena rješenja tih problema, a u literaturi navedena istraživanja u kojima se mogu pronaći opsežniji opisi pojedinih tema. U prvom dijelu rada analizira se mrežna sigurnost i definiraju sigurnosni ciljevi, dok se ostatak rada bavi analizom postojećih *hash* funkcija i njihovom primjenom u praktičnim primjerima.

2. RAČUNALNA SIGURNOST

Postupci vezani uz povećavanje razine sigurnosti pri razmjeni podataka u komunikacijskim mrežama sastoje se od mjera za sprječavanje, otkrivanje ili smanjivanje sigurnosnih rizika koji se pojavljuju pri prijenosu podataka. Prema [1], pojam 'računalna sigurnost' odnosi se na razinu zaštite koja se pruža automatiziranom informacijskom sustavu kako bi se postigli primjenjivi ciljevi očuvanja integriteta, dostupnosti i povjerljivosti informacijskog sustava. Definicija računalne sigurnosti obuhvaća tri temeljna sigurnosna cilja vezana uz podatke, informacije i računalne usluge:

- Integritet: Sprečavanje nepravilnog mijenjanja i uništavanja informacija, uključujući autentičnost informacija. Gubitak integriteta predstavlja neovlaštenu izmjenu ili uništavanje informacija.
- Dostupnost: Osiguravanje pravovremenog i pouzdanog pristupa i korištenja informacija. Gubitak dostupnosti predstavlja ometanje pristupa informacijama ili korištenja informacija i informacijskog sustava.
- Povjerljivost: Očuvanje odobrenih ograničenja za pristup informacijama i njihovo otkrivanje, uključujući sredstva za zaštitu osobne privatnosti i osobnih informacija. Gubitak povjerljivosti predstavlja neovlašteno otkrivanje informacija.

Zahtjevi mrežne i računalne sigurnosti su složeni, stoga su složeni i mehanizmi koji se koriste za ispunjavanje tih zahtjeva. Prilikom razvoja određenog sigurnosnog mehanizma posebno se moraju razmotriti potencijalni napadi na mehanizam, jer su u mnogim slučajevima uspješni napadi osmišljeni korištenjem neočekivanih slabosti u mehanizmu. Dizajner sigurnosnog mehanizma mora pronaći i eliminirati sve slabosti za postizanje sigurnosti, dok napadač mora pronaći samo jednu slabost i ugroziti sigurnost. Osoba koja je u određenoj organizaciji odgovorna za računalnu sigurnost treba sustavan način definiranja zahtjeva za sigurnošću kako bi se efikasno procijenile sigurnosne potrebe organizacije. Jedan od načina je OSI sigurnosna arhitektura koja osobi odgovornoj za sigurnost omogućuje organizaciju zadataka za pružanje sigurnosti. Prema [1], OSI arhitektura se fokusira na:

- Sigurnosni napadi: Sve akcije koje kompromitiraju sigurnost informacija u vlasništvu organizacije. Sigurnosni napadi se dijele na pasivne napade, one čiji cilj je prikupiti i

iskoristiti informaciju određenog sustava bez utjecaja na postojeće resurse sustava, te aktivne napade, one koji pokušavaju modificirati podatke ili stvoriti lažne podatke.

- Sigurnosni mehanizmi: Procesi koji imaju za cilj otkrivanje, sprečavanje ili oporavak od sigurnosnog napada.
- Sigurnosne usluge: Komunikacijske usluge koje poboljšava sigurnost sustava za obradu podataka i prijenos informacija s ciljem odbijanja mogućih napada, pri čemu se pružanje sigurnosne usluge može koristiti kao jedan od sigurnosnih mehanizama.

2.1. Sigurnosne usluge

Sigurnosna usluga se, prema [1], definira kao usluga koju pruža protokolni sloj komunikacijskog sustava, te omogućuje sigurnost sustava i prijenos podataka. Sigurnosne usluge se dijele na pet kategorija: autentifikacija, kontrola pristupa, povjerljivost podataka, integritet podataka i neporecivost.

Autentifikacija osigurava autentičnost komunikacije. Može se promotriti kroz slučaj jedne poruke prilikom čega je funkcija usluge autentičnosti osigurati primatelju poruke uvjerenje da poruka dolazi iz izvora iz kojeg tvrdi da dolazi. Također, može se promotriti kroz slučaj interakcije kao što je povezivanje terminala sa *host*-om, pri čemu ova usluga osigurava autentičnost subjekata u komunikaciji i sprječava neovlašten prijenos ili prijem podataka.

Kontrola pristupa predstavlja ograničavanje i kontrolu pristupa aplikacijama i uređajima putem komunikacijskih veza. Kako bi se ovo postiglo, svaki pojedini subjekt koji pokušava dobiti pristup, prvo mora biti ovjeren tako da se prava pristupa mogu prilagoditi pojedinom subjektu.

Povjerljivost podataka predstavlja zaštitu podataka od pasivnih napada. Postoji nekoliko razina zaštite, a najšira usluga je zaštita podataka koji se u određenom vremenu prenose između dva korisnika. Povjerljivost se također odnosi i na zaštitu podataka od analize, što onemogućava napadaču promatranje izvora, odredišta, duljine i ostalih karakteristika prometa.

Integritet podataka predstavlja zaštitu podataka i osigurava prijem podataka u obliku u kakvom su poslani od strane pošiljatelja, odnosno prijem poruka bez modificiranja, promjene redoslijeda, ponavljanja ili uništavanja podataka. Usluge integriteta se odnose na aktivne napade, stoga se više

pažnje posvećuje detekciji napada, dok za prevenciju napada postoje softveri ili ljudi vrše oporavak sustava od napada.

Neporecivost sprečava pošiljatelja ili primatelja da opovrgne poslanu, odnosno primljenu poruku. Dakle, kada je poruka poslana, primatelj može dokazati da je pošiljatelj poslao poruku (neporecivost izvora), odnosno kada je poruka primljena, pošiljatelj može dokazati da je primatelj primio poruku (neporecivost odredišta).

2.2. Sigurnosni mehanizmi

Za pružanje bilo koje sigurnosne usluge koristi se jedan ili više sigurnosnih mehanizama. Razlikuju se reverzibilni i ireverzibilni mehanizmi šifriranja. Reverzibilni mehanizmi su algoritmi šifriranja koji omogućuju da podaci budu šifrirani te naknadno dešifrirani. Ireverzibilni mehanizmi uključuju *hash* algoritme i kodove za provjeru autentičnosti poruka koji se koriste za digitalni potpis i u aplikacijama za provjeru autentičnosti poruka[1].

Sigurnosni mehanizmi definirani prema [1] su sljedeći:

- Šifriranje: Korištenje matematičkih algoritama za transformaciju podataka u naizgled nečitljiv oblik. Kako bi se podaci vratili u izvorni oblik, potrebno je poznavati algoritam i enkripcijski ključ.
- Digitalni potpis: Kriptografska transformacija podataka koja omogućuje primatelju potvrdu izvora i integriteta podataka.
- Kontrola pristupa: Pristup resursima kroz strogo definirana pravila.
- Integritet podataka: Mehanizmi za osiguravanje integriteta podatkovnih tokova ili podatkovnih blokova.
- Autentifikacijska razmjena: Utvrđivanje identiteta subjekata u komunikaciji.
- Nadopunjavanje prometa: Umetanje bitova unutar praznina u podatkovnom toku kako bi se otežali pokušaji analize prometa.
- Kontrola usmjeravanja: Odabir osiguranih ruta za određene podatke i omogućavanje izmjene usmjeravanja kada se sumnja na kompromitaciju.

- Ovjera: Korištenje pouzdane treće strane za osiguranje određenog sigurnosnog svojstva podatkovnu razmjenu.

USLUGE/ MEHANIZMI	Šifriranje	Digitalni potpis	Kontrola pristupa	Integritet podataka	Autentifikacijska razmjena	Nadopunjavanje prometa	Kontrola usmjeravanja	Ovjera
Autentifikacija subjekata	•	•			•			
Autentifikacija podataka	•	•						
Kontrola pristupa			•					
Povjerljivost prometa	•						•	
Integritet podataka	•	•		•				
Neporecivost		•		•				•
Dostupnost				•	•			

Tab. 2.1. Odnos između sigurnosnih usluga i mehanizama [1]

2.3. Model mrežne sigurnosti

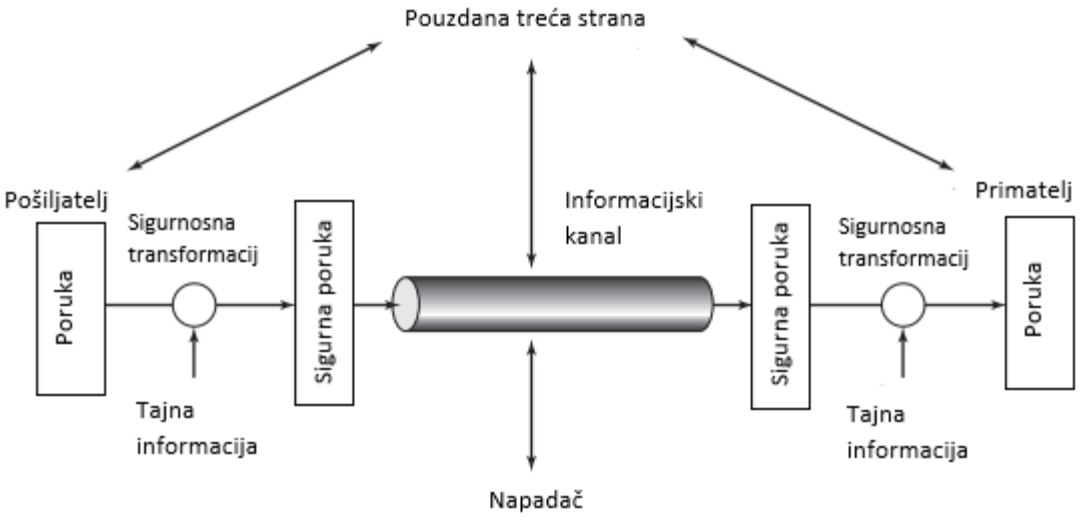
Poruka se šalje od pošiljatelja do primatelja putem određenog Internet sustava. Komunikacijski kanal se uspostavlja definiranjem rute putem interneta od izvora do odredišta i uz korištenje komunikacijskih protokola subjekata koji sudjeluju u komunikaciji. Ukoliko je potrebno zaštititi prijenos informacija, tada na snagu nastupaju sigurnosni aspekti koji se sastoje od dvije osnovne komponente:

- Sigurnosna transformacija informacije koja se šalje, kao što je šifriranje poruke i dodavanje koda koji omogućava provjeru identiteta pošiljatelja

- Tajna informacija koju dijele subjekti u komunikaciji i koja bi trebala biti nepoznata napadaču. Primjer tajne informacije je ključ za šifriranje koji se koristi sa transformacijom.

Pouzdana treća strana, kojoj „vjeruju“ sudionici u komunikaciji, je potrebna kako bi se postigao siguran prijenos podataka i tajne informacije zadržavajući ih od bilo kojeg napadača. Prema [1] model na slici 2.1. prikazuje četiri osnova zadatka u dizajniranju sigurnosne usluge.

1. Dizajniranje algoritma za provođenje sigurnosne transformacije.
2. Generiranje tajne informacije koja će se koristiti zajedno sa algoritmom.
3. Razvoj metode za distribuciju i dijeljenje tajnih informacija.
4. Specificiranje protokola koji koristi sigurnosne algoritme i tajne informacije za postizanje određene sigurnosne usluge.

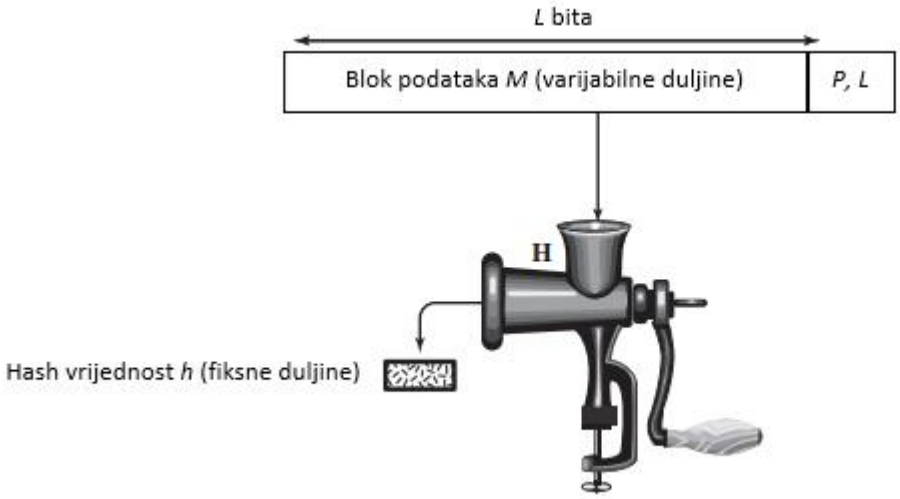


Sl. 2.1. Model mrežne sigurnosti [1]

3. KRIPTOGRAFSKE *HASH* FUNKCIJE

Prema [2], kriptografske *hash* funkcije, odnosno tzv. sažetci poruka (engl. *hash functions*), predstavljaju temelj moderne kriptografije. Ovaj rad opisuje kriptografske *hash* funkcije, u daljnjem tekstu *hash* funkcije, i njihovu primjenu pri očuvanju integriteta podataka i pri provjeri autentičnosti poruke.

Hash funkcija H koristi podatkovni blok M promjenjive veličine kao ulaz i proizvodi *hash* vrijednost sa fiksnom veličinom $h=H(M)$, odnosno *hash* funkcija preslikava nizove podataka proizvoljne konačne duljine u nizove fiksne duljine. Ako *hash* funkcija proizvodi *hash* vrijednosti duljine n-bita, tada je vjerojatnost da će se proizvoljni ulazni podaci preslikati u određenu *hash* vrijednost jednaka 2^{-n}. Dobra *hash* funkcija će svojom primjenom na veliki skup ulaznih podataka proizvesti ravnomjerno raspoređenu *hash* vrijednost koja je prividno slučajna. Mala promjena bita ulaznog podatka M, rezultira promjenom pripadne *hash* vrijednosti, stoga je zaštita integriteta podataka glavni cilj *hash* funkcija. Algoritam kriptografske *hash* funkcije posjeduje svojstvo da je računski neizvedivo pronaći objekt podataka koji se preslikava na unaprijed određeni *hash* rezultat ili dva objekta podataka koji se preslikavaju na isti *hash* rezultat. Zbog navedenih karakteristika, *hash* funkcije se često koriste za određivanje promjene podataka. Kako bi se utvrdila promjena podataka, prilikom slanja poruke y računa se *hash* vrijednost navedene poruke u određenom vremenu T_1, pri čemu je zagarantiran integritet ove *hash* vrijednosti. Potom se u određenom vremenu T_2 provodi test kojim se utvrđuje je li poruka y' jednaka originalnoj poruci. Test se provodi tako što se računa *hash* vrijednost poruke y', koja se potom uspoređuje sa zaštićenom *hash* vrijednosti originalne poruke. Ako su vrijednosti jednake, može smatrati da su podaci poruke također jednaki, odnosno da poruka nije izmijenjena. Na ovaj način problem očuvanja integriteta potencijalno velike poruke je sveden na manju *hash* vrijednost fiksne veličine [1, 2].



Sl. 3.1. Kriptografska *hash* funkcija; $h=H(M)$; P = dopuna (*padding*) i L = duljina polja (*length field*) [1]

Slika 3.1. opisuje način rada kriptografske *hash* funkcije. Ulaz je zadan kao cijeli broj neke fiksne vrijednosti, npr. 1024 bita. *P* predstavlja *padding*, odnosno dopunu koja sadrži vrijednost duljine originalne poruke u bitovima, dok *length field*, odnosno duljina polja predstavlja sigurnosnu mjeru koja napadaču otežava proizvodnju poruke sa istom *hash* vrijednosti. Ostatak poglavlja će se baviti klasifikacijom *hash* funkcija i njihovom konstrukcijom, nakon čega će se razmotriti sigurnost kriptografskih *hash* funkcija [1].

3.1. Klasifikacija *hash* funkcija

Hash funkcije se općenito mogu podijeliti u dvije klase, *hash* funkcije bez ključa i *hash* funkcije sa ključem. *Hash* funkcije bez ključa specificiraju jedan ulazni parametar - poruku, dok *hash* funkcije sa ključem specificiraju dva ulazna parametra - poruku i tajni ključ [2].

Prema [3] skup *hash* funkcija sa ključem sastoji se od četiri elementa:

1) X = skup mogućih poruka
2) Y = konačni skup mogućih sažetaka poruka
3) K = konačan skup ključeva

4) H = za svaki $k \in K$, postoji *hash* funkcija $h_k \in K$, $h_k : x \rightarrow y$

Skup x može biti konačan ili beskonačan, dok je skup y uvijek konačan. Za svaki par $(x, y) \in X$ x Y kaže se da je valjan sa ključem K ako je $h_K(x) = y$.

Hash funkcija bez ključa je funkcija $h: X \rightarrow Y$, gdje je X skup mogućih poruka, a Y konačni skup mogućih sažetaka poruka. Skup *hash* funkcija bez ključa jest skup u kojem postoji samo je mogući ključ, $|K| = 1$.

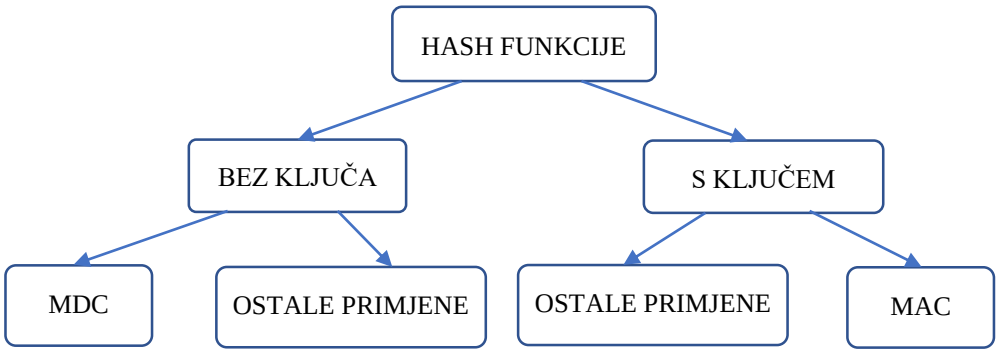
Hash funkcija h posjeduje svojstvo kompresije i jednostavnog proračuna. Kompresija označava svojstvo funkcije h da uzima ulaz x koji je proizvoljne konačne duljine te ga preslikava u izlaz $h(x)$, odnosno *hash* vrijednost koja je fiksne duljine, dok svojstvo jednostavnog proračuna opisuje da je relativno jednostavno izračunati $h(x)$ uz danu funkciju h i ulaz x

Hash funkcije je za stvarnu upotrebu potrebno klasificirati na temelju svojstava koje pružaju i zahtjeva određenih aplikacija u kojima se primjenjuju. Na temelju ovakve klasifikacije, u ovom poglavlju će se detaljnije razmatrati dva tipa *hash* funkcija:

1. Kodovi za otkrivanje modifikacije MDC (engl. *modification detection codes*)
   MDC kodovi se koriste u svrhu očuvanja integritet podataka prema zahtjevima za specifičnu primjenu. Bitno je napomenuti da MDC kodovi ne koriste tajni ključ.

2. Kodovi za autentifikaciju poruke MAC (engl. *message authentication codes*)
   Svrha MAC kodova jest osigurati autentičnost i integritet poruke. MAC kodovi su podklasa *hash* funkcija sa ključem jer se sastoje od poruke i tajnog ključa

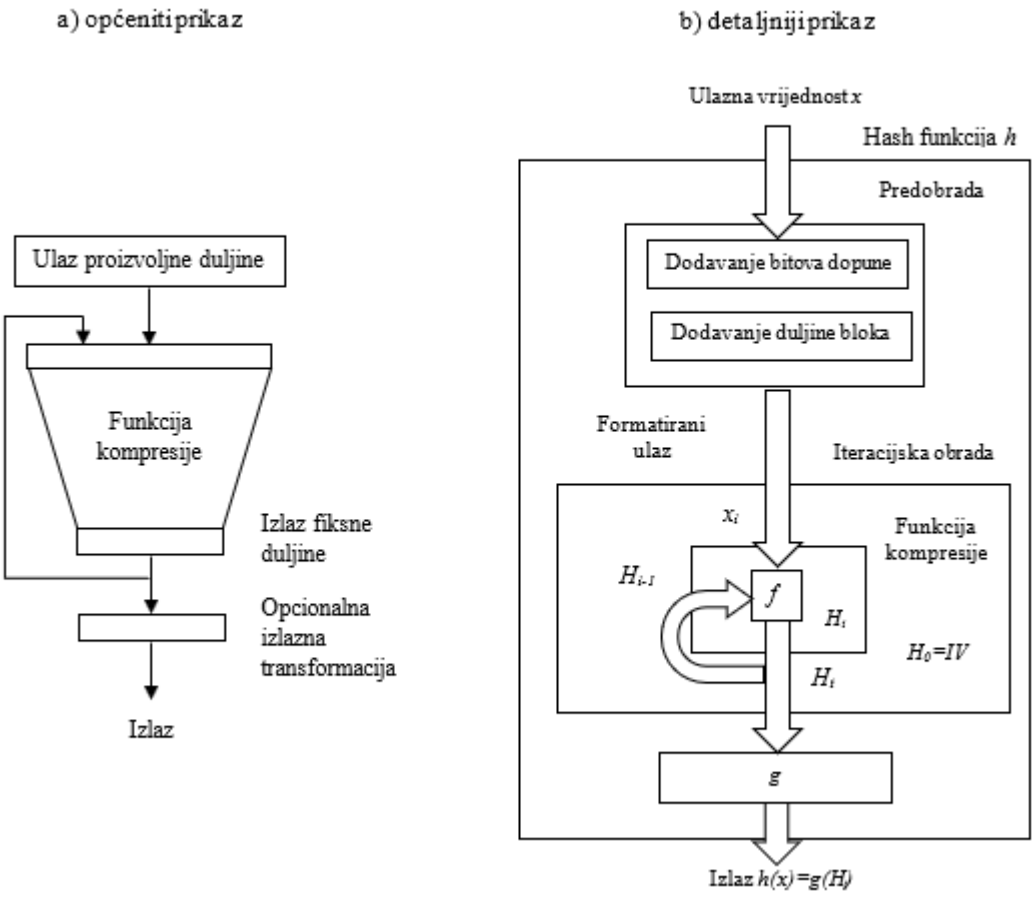
Pojednostavljena klasifikacija *hash* funkcija je prikazana na slici 3.2..



Sl. 3.2. Klasifikacija *hash* funkcija [2]

3.2. Konstrukcija *hash* funkcija

Konstrukcija *hash* funkcija obično je modularna, a funkcije su dizajnirane pomoću funkcije kompresije. Funkcija kompresije predstavlja algoritam koji može komprimirati samo blokove poruke fiksne duljine, a temelji se na blok šifri ili permutaciji [4]. Stoga su *Hash* funkcije bez ključa dizajnirane kao iterativni proces koji obrađuje uzastopne ulazne blokove fiksne veličine kako bi proizveo *hash* vrijednost, što je prikazano na slici 3.3. [2].



Sl. 3.3. Konstrukcija *hash* funkcija [2]

Ulaz *x* proizvoljne duljine dijeli se na blokove x_i fiksne duljine, pri čemu se po potrebi dodaju dodatni bitovi kako bi se postigla ukupna potrebna duljina bloka. Svaki blok x_i se koristi kao ulaz u *hash* funkciju *f*, koja je fiksne duljine i predstavlja funkciju kompresije, a koristi se za proračun međurezultata fiksne duljine pomoću prethodnih međurezultata i idućeg ulaznog bloka x_i. Prema (3-1) [2] H_i označava međurezultat faze *i*, proces iterativne *hash* funkcije sa ulazom *x* može se prikazati na sljedeći način [2]:

$$H_0 = IV$$

$$H_i = f(H_{i-1}, x_i),\ 1 \leq i \leq t$$

$$h(x) = g(H_t) \qquad (3\text{-}1)$$

gdje je

- H_{i-1} – varijabla duljine *n*-bita između stanja *i-1* i stanja *i*

- H_0 – predefinirana početna vrijednost inicijalizacijske vrijednosti *IV*

- *g* – konačni korak dodavanja varijable H_{i-1} rezultatu $g(H_t)$

3.3. Jednostavne *hash* funkcije

Hash funkcije koriste ulaz predstavljen kao niz n-bitnih blokova koji se obrađuju redom jedan po jedan kako bi se proizvela n-bitna *hash* funkcija [1]. Jedna od najjednostavnijih *hash* funkcija je isključivo ILI (XOR) funkcija pojedinog bloka koja je prema (3-2) [1] izražena na sljedeći način:

$$C_i = b_{i1} \oplus b_{i2} \oplus \ldots \oplus b_{im} \qquad (3\text{-}2)$$

gdje je

- C_i = i-ti bit *hash* vrijednosti

- m = broj n-bitnih blokova ulaza

- b_{ij} = i-ti bit i j-tom bloku

- $\oplus$ = XOR operator

Funkcija proizvodi paritete za pojedine pozicije bitova i zove se longitudinalna provjera redundancije. Svaka n-bitna *hash* vrijednost je jednako vjerojatna, a vjerojatnost pogreške podataka rezultirat će nepromijenjenom *hash* vrijednošću 2^{-n}. Učinkovitost funkcije se smanjuje sa povećanjem vjerojatnosti podataka. Kako bi se povećala učinkovitost, izvodi se rotacija jednog bita na *hash* vrijednost nakon obrade svakog bloka. Postupak se izvodi na sljedeći način:

1. n-bitna *hash* vrijednost se postavlja na nulu
2. svaki sljedeći n-bitni blok se obrađuje na sljedeći način:
   1) rotiraj ulijevo trenutnu *hash* vrijednost za jedan bit
   2) izvedi XOR bloka u *hash* vrijednost

Ovim postupkom se postiže veća mjera integriteta podataka, no korištenje jednostavnog ili rotiranog XOR-a je nedovoljno za postizanje veće sigurnosti podataka ako je samo *hash* vrijednost šifrirana [1].

3.4. Sigurnosni zahtjevi kriptografskih *hash* funkcija

Jednostavan način analiziranja *hash* funkcije je pokušati izvršiti napad na nju i otkriti otvoreni tekst. Ovaj način je poznat kao kriptoanalitički pristup, a osnovna ideja je razmotriti *hash* funkcije i pokušati pronaći kolizije i preslike [4]. Za *hash* vrijednost $h = H(x)$, kaže se da je x preslika funkcije h, tj. x je blok podataka čija *hash* funkcija je h. Kolizija se pojavljuje ukoliko postoje preslike x i y koje su različite, a njihove *hash* funkcije su jednake, odnosno $H(x) = H(y)$. Kolizije nisu poželjne zato što *hash* funkcije koristimo za integritet podataka. Svaka *hash* vrijednost može imati više preslika, a broj preslika predstavlja potencijalni broj kolizija za danu *hash* vrijednost. Broj preslika za pojedinu *hash* vrijednost u prosjeku iznosi 2^{b-n} preslika, gdje je b broj ulaznih blokova podataka, a n duljina *hash* vrijednosti izražena u bitima. U ovom i ostalim poglavljima se spominju pojmovi „jednostavno je izračunati“ i „računski nije moguće“. Pojam „jednostavno je izračunati“ može označavati vrijeme, odnosno određeni broj operacija ili vremenskih jedinica (sekundi ili milisekundi) koje su potrebne za izračun određene funkcije, a pojam „računski nije moguće“ označava napore koji nadilaze razumljive resurse i eksponencijalno malu vjerojatnost da je sigurnosno svojstvo prekršeno [1, 2].

Zahtjev	Opis
Promjenjiva veličina ulaza	H se može primijeniti na ulazni blok podataka proizvoljne veličine
Fiksna veličina izlaza	H proizvodi izlaze fiksne veličine
Efikasnost	$H(x)$ je jednostavno izračunati za bilo koji zadani blok podataka x
Jednosmjernost	Za proizvoljnu *hash* vrijednost h, računski nije moguće pronaći y tako da vrijedi $H(y) = h$
Slaba otpornost na kolizije	Za proizvoljni blok podataka x, računski nije moguće pronaći $y \neq x$ tako da vrijedi $H(y) =$

	H(x)
Otpornost na kolizije	Računski nije moguće pronaći dva različita bloka podataka (x,y) tako da vrijedi H(x) = H(y)
Pseudoslučajnost	Izlaz od H zadovoljava standardna testiranja slučajnosti

Tab. 3.1. Zahtjevi kriptografske funkcije H [1]

Prema [1] tablica 3.1. prikazuje prihvaćene zahtjeve kriptografskih *hash* funkcija. Prva tri zahtjeva se odnose na praktičnu primjenu *hash* funkcija. Jednosmjernost navodi kako je lako generirati *hash* vrijednost sa danom porukom, a nemoguće generirati poruku sa danom *hash* vrijednošću. Napadač koji pokušava napasti zahtjev jednosmjernosti za cilj ima pronaći ulazni podatak pomoću kojeg dobije danu *hash* vrijednost. Napadač definira raspon *hash* vrijednosti *h* te uspijeva pronaći *y* tako da vrijedi H(y) = *h*. Napadač koji pokušava napasti zahtjev slabe otpornosti na kolizije ima za cilj pronaći ulazni podatak s kojim dobije *hash* vrijednost od drugog danog ulaza. Napadač definira raspon ulaznih vrijednosti *y,* te uspijeva pronaći ulaz *x* tako da vrijedi H(x) = H(y) [4]. Peti zahtjev, odnosno slaba otpornost na kolizije, garantira da je nemoguće pronaći alternativnu poruku sa istom *hash* vrijednošću kao i dana poruka. Na ovaj način se sprečava krivotvorenje prilikom korištenja šifrirane *hash* vrijednosti.

Hash funkcija koja zadovoljava prvih pet sigurnosnih zahtjeva naziva se slaba *hash* funkcija. Ukoliko je zadovoljen i šesti zahtjev, odnosno otpornost na koliziju, tada se kaže da je *hash* funkcija jaka. Kako bi napadač uspješno pronašao koliziju, njegov cilj je pronaći dva različita ulaza u *hash* funkciju koji komprimiraju u istu vrijednost, odnosno napadač uspješno pronalazi koliziju ako pronađe dva različita ulazna bloka podataka *x* i *y* tako da vrijedi H(x) = H(y). Jaka *hash* funkcija štiti od napada prilikom kojeg jedna strana šalje drugoj strani poruku koju treba potpisati, npr. ako Bob pošalje poruku Alice i ona ju potpiše, tada Bob otkriva dvije poruke sa istom *hash* vrijednošću prilikom čega prva poruka zahtijeva Alice plaćanje manjeg iznosa, a druga poruka zahtijeva Alice plaćanje većeg iznosa. Alice potpiše prvu poruku, prilikom čega Bob može tvrditi da je druga poruka autentična.

Posljednji zahtjev, pseudoslučajnost, govori da se kriptografske *hash* funkcije obično koriste za generiranje ključeva i pseudoslučajnih brojeva i da se tri svojstva otpornosti, koja ovise o izlazu *hash* funkcije, pojavljuju slučajno [1, 4].

3.5. Napadi na *hash* funkcije

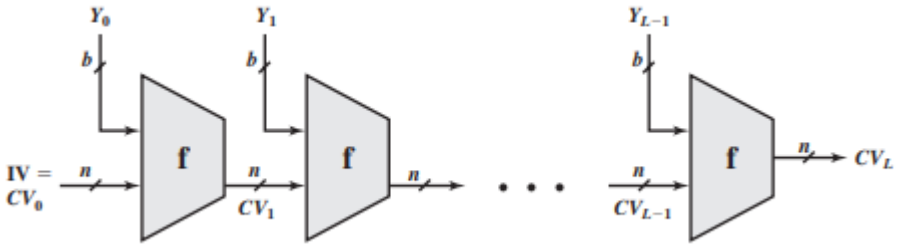
Postoje dvije vrste napada na *hash* funkcije: napad grubom silom i kriptoanaliza.

Napad grubom silom (engl. *Brute-force attack*) ovisi o duljini bita i *hash* vrijednosti, a prilikom napada napadač isprobava sve moguće ključeve na zadanom šifratu, sve dok ne otkrije otvoreni tekst.

Prilikom napada na jednosmjernost i slabu otpornost na kolizije, napadač želi pronaći vrijednost y takvu da je H(y) jednaka danoj *hash* vrijednosti h, te se metodom napada grube sile nasumce odabiru vrijednosti y sve dok se ne dogodi kolizija. Za m-bitnu *hash* vrijednost razina pokušaja je proporcionalna 2^m.

Prilikom napada otpornosti na koliziju, napadač želi pronaći blokove podataka, x i y, koji daju istu *hash* funkciju: H(x) = H(y). Ovaj napad zahtijeva značajno manje napora nego napad na presliku. Razina napora se objašnjava matematičkim izrazom nazvanim paradoks rođendana koji navodi da prilikom odabira slučajne varijable u rasponu od 0 do N-1, vjerojatnost ponovnog odabira iste varijable prelazi 0.5 nakon $\sqrt{N}$ odabira. Prema tome, za m-bitne *hash* vrijednosti vrijedi da je vjerojatnost odabira dva bloka podataka sa istom *hash* vrijednosti unutar $2^{m/2}$ pokušaja. Vrijednost $2^{m/2}$ određuje jačinu *hash* koda protiv napada grube sile.

Kriptoanaliza nastoji iskoristiti određene osobine algoritma za izvođenje određenog napada. Mjerenje otpora *hash* algoritma na kriptoanalizu predstavlja usporedbu snage algoritma sa naporom potrebnim za napad grube sile, odnosno idealan *hash* algoritam će zahtijevati napor kriptoanalize veći ili jednak naporu napada grube sile. Kako bi se razumjeli napadi kriptoanalize, prema slici 3.4. potrebno je proučiti osnovnu strukturu sigurnih *hash* funkcija koja predstavlja temelj gotovo svih *hash* funkcija koje se danas koriste, uključujući SHA o kojoj će se raspravljati u idućim poglavljima.



Sl. 3.4. Struktura sigurne *hash* funkcije [1]

Gdje je:

- IV = inicijalna vrijednost
- CV_i = varijabla lanca
- Y_i = i-ti ulazni blok
- f = algoritam kompresije
- L = broj ulaznih blokova
- n = duljina *hash* vrijednosti
- b = duljina ulaznog bloka

Hash funkcija uzima ulazne poruke i dijeli ih u L bokova fiksne veličine koji sadrže b bita. Zadnji blok također sadrži vrijednost duljine ulaza u *hash* funkciju. *Hash* algoritam uključuje korištenje algoritma kompresije f koji uzima dva ulaza i proizvodi n-bitni izlaz. Prema izrazu (3-3) [1] *hash* funkcija se može predstaviti kao:

$$CV_0 = IV$$

$$CV_i = f(CV_{i-1}, Y_{i-1}) \quad 1 \leq i \leq L$$

$$H(M) = CV_L \tag{3-3}$$

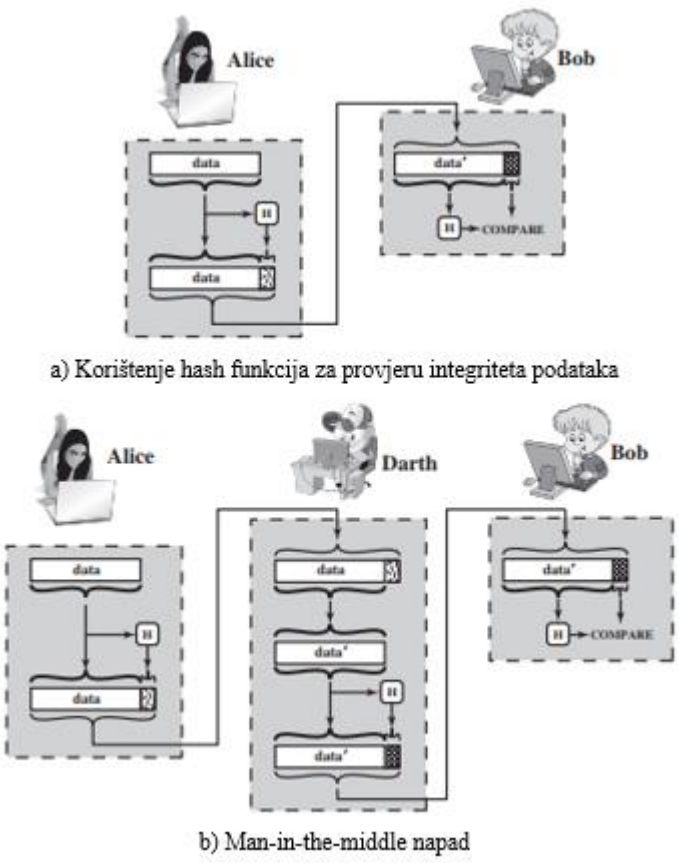
gdje ulaz u *hash* funkciju predstavlja poruku M koja se sastoji od blokova $Y_0, Y_1, \ldots, Y_{L-1}$.

Kriptoanaliza *hash* funkcije se bavi unutarnjom strukturom algoritma f i pokušajima pronalaženja učinkovitih tehnika za izazivanje sudara prilikom izvođenja algoritma f. Napad na algoritam f ovisi

o iskorištavanju njegove unutarnje strukture. Algoritam *f* se sastoji od niza ciklusa obrade tako da napad uključuje analizu uzoraka promjene bita iz runde u rundu [1].

3.6. Primjena *hash* funkcija

Hash funkcije imaju široku primjenu u raznim sigurnosnim aplikacijama i internetskim protokolima. *Hash* funkcije se koriste prilikom autentifikacije poruke koja predstavlja mehanizam za provjeru integriteta poruke. Kada se *hash* funkcija koristi za autentifikaciju poruke, tada se *hash* vrijednost često naziva sažetak poruke. Slika 3.5.a) prikazuje upotrebu *hash* funkcije za provjeru integriteta poruke. Pošiljatelj (Alice) računa *hash* vrijednost te ju šalje zajedno sa porukom do primatelja, potom primatelj (Bob) također računa *hash* vrijednosti i uspoređuje izračunatu vrijednost sa vrijednosti koju je primio od pošiljatelja. Ako izračunata i primljena vrijednost nisu jednake, primatelj tada može zaključiti da primljena poruka nije jednaka poruci koja je poslana, odnosno da se dogodila izmjena izvorne poruke [1].



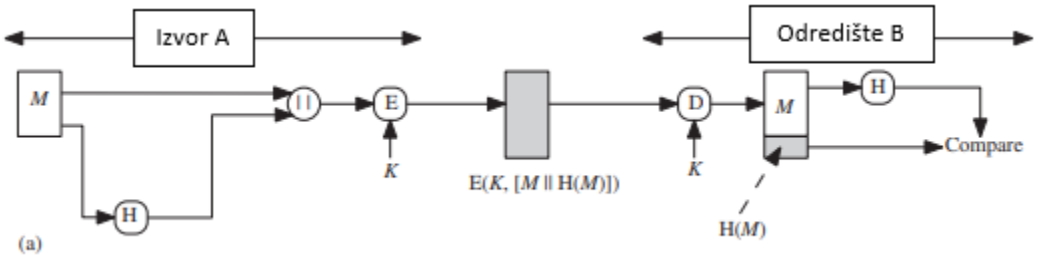
Sl. 3.5. Napad na *hash* funkciju [1]

Hash vrijednost se mora zaštititi kako u slučaju napada i promjene poruke napadač ne bi bio u mogućnosti promijeniti *hash* vrijednost te na taj način zavarati primatelja. Ovakav napad se zove *Man-in-the-middle* napad. Prema slici 3.5.b) prikazan je ovakav scenarij napada, Alice šalje poruku sa pripadajućom *hash* vrijednosti, potom Darth presreće poruku i vrši izmjenu poruke te računa novu *hash* vrijednost. Bob prima izmijenjenu poruku sa novom *hash* vrijednosti, potom računa *hash* vrijednosti, ali ne otkriva promjenu vrijednosti. Kako bi se ovakav napad spriječio, potrebno je zaštiti *hash* vrijednost.

Hash vrijednost može koristiti više metoda za osiguranje autentičnosti poruke:

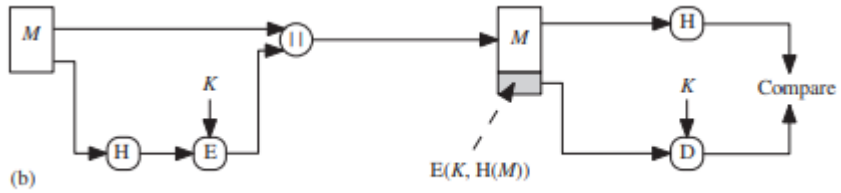
a) prema slici 3.7. poruka se zajedno sa dodanim *hash* kodom šifrira pomoću simetrične šifre, a enkripcija se primjenjuje na cijelu poruku, uključujući i *hash* vrijednost, stoga je osigurana povjerljivost poruke

b)



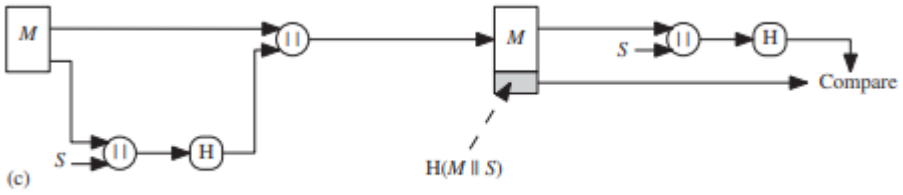
Sl. 3.7. Primjer korištenja enkripcije na cijelu poruku [1]

c) prema slici 3.8. simetrična enkripcija se primjenjuje samo na *hash* vrijednost, čime se pojednostavljuje obrada što je korisno za aplikacije koje ne zahtijevaju povjerljivost



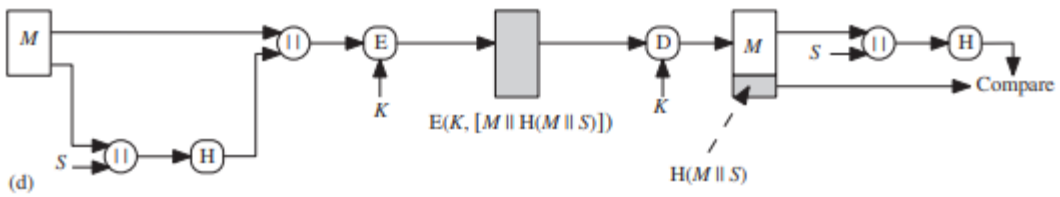
Sl. 3.8. Primjer korištenja enkripcije na *hash* vrijednost [1]

d) prema slici 3.9. *hash* funkcija se koristi za autentifikaciju, ali se ne koristi enkripcija



Sl. 3.9. Primjer korištenja *hash* funkcije koristi za autentifikaciju [1]

e) prema slici 3.10. *hash* funkcija se koristi za autentifikaciju, ali se koristi enkripcija čime se osigurava povjerljivost poruke

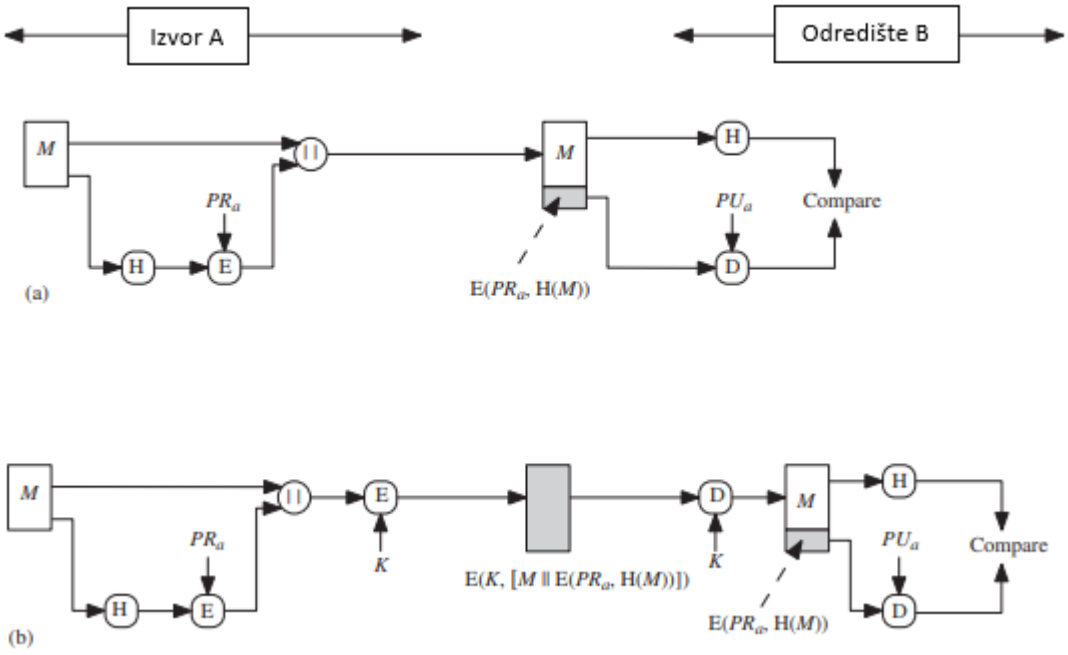


Sl. 3.10. Primjer osiguravanja povjerljivosti poruke [1]

Ako povjerljivost poruke nije nužna, tada metoda b) ima prednost nad metodama a) i d) jer je potrebno manje izračuna budući da se ne šifrira poruka zajedno sa *hash* vrijednosti.

Provjera autentičnosti poruke se najčešće vrši pomoću koda za autentifikaciju poruke (MAC), koji se koristi između pošiljatelja i primatelja koji dijele tajni ključ za provjeru autentičnosti poruka. O MAC kodovima će se više raspravljati u poglavlju 3.8.

Hash funkcije se također koriste za digitalni potpis, što je slično provjeri autentičnosti. *Hash* vrijednost poruke se šifrira sa privatnim ključem, dok se javni ključ koristi za provjeru integriteta poruke. Napadač mora posjedovati privatni ključ ukoliko želi presresti poruku.



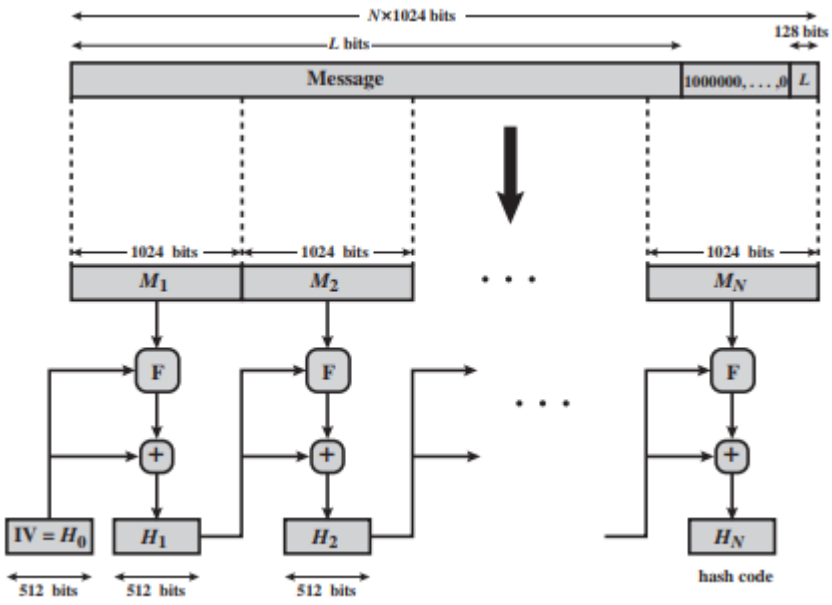
Sl. 3.11. Primjeri digitalnog potpisa [1]

Slika 3.11. prikazuje način na koji se *hash* vrijednost koristi za omogućavanje digitalnog potpisa. *Hash* vrijednost se šifrira koristeći šifriranje sa privatnim ključem pošiljatelja, što je prikazano na slici 3.11.a. Digitalni potpis je omogućen jer jedino pošiljatelj može izračunati šifriranu *hash* vrijednost. Ako je uz digitalni potpis potrebno osigurati i povjerljivost, tada se poruka zajedno sa šifriranom *hash* vrijednosti šifrira pomoću simetričnog tajnog ključa, što je prikazano na slici 3.11.b [1].

3.7. Sigurnosni *hash* algoritam (SHA)

Sigurnosni *hash* algoritam SHA je u posljednje vrijeme najraširenija *hash* funkcija. SHA je razvijen od strane *National Institue of Standards and Technology* (NIST) 1993. godine. SHA se bazira na *hash* funkciji MD4, a dizajn je sličan MD4 modelu. Kada su otkrivene slabosti u SHA, poznatom pod nazivom SHA-0, izdana je nova verzija algoritma pod nazivom SHA-1. SHA-1 kao izlaz daje *hash* vrijednost duljine 160 bita. 2002. godine NIST preporučuje zamjenu SHA-1 algoritma sa tri nove verzije SHA sa pripadajućim *hash* vrijednostima duljine 256, 384 i 512 bita, a ova tri algoritma su poznata pod nazivom SHA-2. NIST također najavljuje poziv za izradu novog SHA-3 algoritma [1, 4].

Verzija SHA-512 će se koristiti kao primjer za opisivanje SHA algoritma, dok su sve druge varijante algoritma vrlo slične. Algoritam koristi ulaznu poruku koja može biti maksimalne duljine 2^{128} bita i procesira ju u blokovima duljine 1024 bita, a potom proizvodi izlaz u obliku sažetka poruke duljine 512 bita.



Sl. 3.12. Postupak izrade sažetka poruke koristeći SHA-512 algoritam [1]

Slika 3.12. prikazuje postupak obrade poruke za izradu sažetka. Postupak se prema [1] sastoji od sljedećih koraka:

1. Dodavanje nadopune

   Poruka se nadopunjunje pomoću nadopune čija duljina iznosi 896(*mod* 1024), prilikom čega se nadopuna uvijek dodaje čak i kada je poruka već željene duljine. Nadopuna se sastoji od prvog bita koji je 1, a svi sljedeći bitovi su 0.
2. Dodavanje duljine poruke

   Svakoj poruci je potom dodan blok podataka duljine 128 bita koji sadrži duljinu originalne poruke bez nadopune. Prema slici 3.12. izlaz prva dva koraka je prikazan u obliku poruke duljine 1024 bita. Proširena poruka je prikazana kao niz blokova duljine 1024 bita M_1, M_2, ..., M_n, pri čemu ukupna duljina proširene poruke iznosi N x 1024 bita.
3. Inicijalizacija *hash* međuspremnika

Hash međuspremnik se koristi za pohranjivanje međurezultata i krajnjih rezultata *hash* funkcije, a sastoji od 8 64-bitnih registara (a, b, c, d, e, f, g, h) čiji sadržaj je sljedeći (izražen u heksadecimalnom zapisu):

a = 6A09E667F3BCC908

b = BB67AE8584CAA73B

c = 3C6EF372FE94F82B

d = A54FF53A5F1D36F1

e = 510E527FADE682D1

f = 9B05688C2B3E6C1F

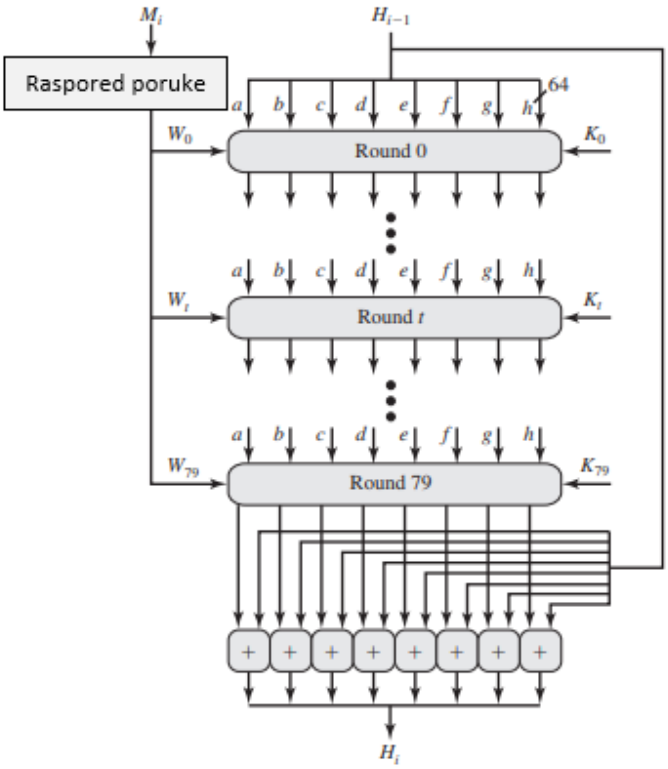
g = 1F83D9ABFB41BD6B

h = 5BE0CD19137E2179

Navedene vrijednosti sadržaja registara proračunate su tako što se uzimaju prva 64 bita decimalnih vrijednosti kvadratnih korijena prvih 8 brojeva, a pohranjene su u *big-endian* format pri čemu je krajnji lijevi bit registra najznačajniji bit.

4. Procesiranje poruke u 1024-bitnim blokovima

   Modul prikazan na slici 3.13. predstavlja jezgru algoritma te se sastoji od 80 rundi.



Sl. 3.13. Procesiranje pojedinačnog 1024-bitnog bloka [1]

U svakoj rundi *t* uzima se 512-bitna vrijednost međuspremnika te se ažurira njegov sadržaj. Također se u svakoj rundi uzima 64-bitna vrijednost W_t od 1024-bitnog bloka M_t koji se obrađuje u pojedinoj rundi. Prilikom svake runde koristi se aditivna konstanta K_t koja označava početna 64 bita decimalnog dijela kubnih korijena prvih 80 brojeva. Vrijednost H_i dobije se zbrojem (*modulo* 2^{64}) izlaza iz osamdesete runde sa ulazom u početnu rundu $H_{i=1}$.

5. Izlaz

   512-bitni sažetak poruke koji se pojavljuje na izlazu *N*-tog stupnja, dobije se nakon obrade svih *N* 1024-bitnih blokova.

Prema (3-4) [1] SHA-512 algoritam se može prikazati na sljedeći način:

$$H_0 = IV$$

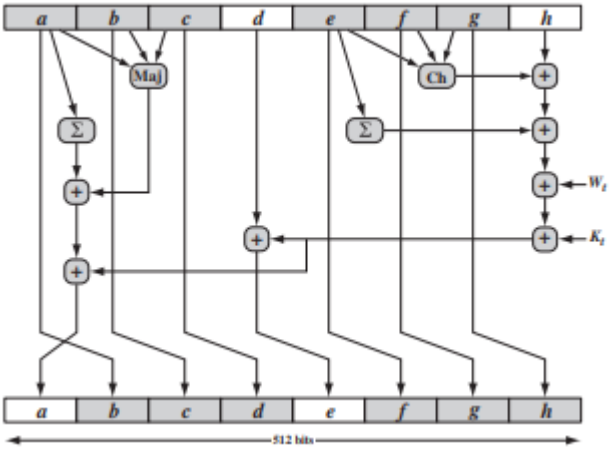
$$H_i = \mathrm{SUM}_{64}(H_{i-1}, \mathrm{abcdefgh}_i)$$

$$MD = H_N \qquad (3\text{-}4)$$

gdje je:

- *IV* – inicijalna vrijednost abcdefgh međuspremnika
- $abcdefgh_i$ – izlaz zadnje runde procesiranja *i*-tog bloka poruke
- *N* – broj blokova u poruci
- SUM_{64} – zbrajanje (*modulo* 2^{64}) odvojeno za svaku riječ
- *MD* – konačna vrijednost sažetka poruke

SHA-512 funkcija runde



Sl. 3.14. Elementarna SHA-512 operacija [1]

Slika 3.14. prikazuje pojedinačnu rundu SHA-512 operacije, pri čemu se prema (3-5) [1] svaka runda može definirati na sljedeći način:

$$T_1 = h + \mathrm{Ch}(e, f, g) + \left(\sum_1^{512} e\right) + W_t + K_t$$

$$T_2 = \left(\sum_0^{512} a\right) + \mathrm{Maj}(a, b, c)$$

$$h = g$$

$$g = f$$

$$f = e$$

$$e = d + T_1$$

$$d = c$$

$$c = b$$

$$b = a$$

$$a = T_1 + T_2 \qquad (3\text{-}5)$$

gdje je:

- t = broj koraka; $0 \leq t \leq 79$
- Ch(e, f, g) = (e AND f) $\oplus$ (NOT e AND g)
- Maj(a, b, c) = (a AND b) $\oplus$ (a AND c) $\oplus$ (b AND c)
- $(\sum_0^{512} a) = \text{ROTR}^{28}(a) \oplus \text{ROTR}^{34}(a) \oplus \text{ROTR}^{39}(a)$
- $(\sum_1^{512} e) = \text{ROTR}^{14}(e) \oplus \text{ROTR}^{18}(e) \oplus \text{ROTR}^{41}(e)$
- $\text{ROTR}^n(x)$ = kružna rotacija 64-bitnog argumenta x udesno za n bita
- Wt = 64-bitna poruka izvedena iz trenutnog 1024-bitnog ulaznog bloka
- Kt = a 64-bitna aditivna konstanta
- + = zbrajanje modulo 2^{64}

Izvođenje 64-bitnih riječi W_t iz 1024-bitne poruke prikazano je na slici 3.15. Prvih 16 vrijednosti W_t se uzima direktno iz 16 riječi trenutnog bloka, a prema (3-6) [1] ostale vrijednosti su definirane formulom:

$$W_t = \sigma_1^{512}(W_{t-2}) + W_{t-7} + \sigma_0^{512}(W_{t-15}) + W_{t-16} \qquad (3\text{-}6)$$

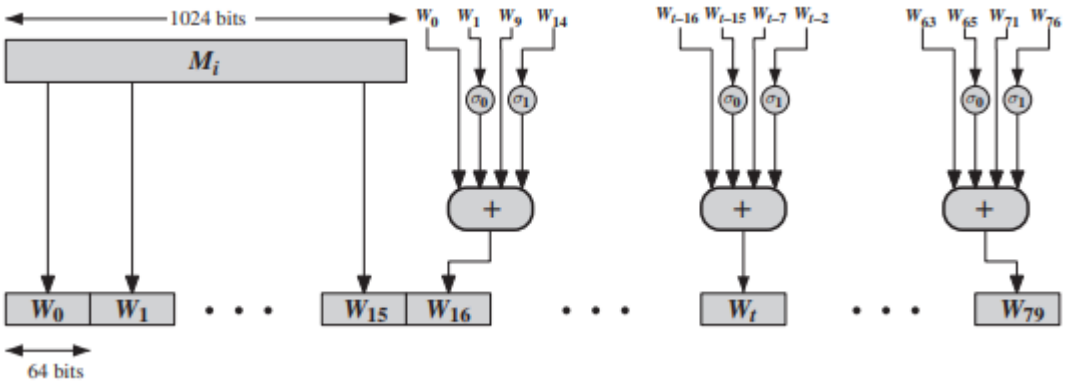
gdje je:

- $\sigma_1^{512}(x) = \text{ROTR}^1(x) \oplus \text{ROTR}^8(x) \oplus \text{SHR}^7(x)$

- $\sigma_0^{512}(x) = \text{ROTR}^{19}(x) \oplus \text{ROTR}^{61}(x) \oplus \text{SHR}^6(x)$

- $\text{SHR}^n(x)$ = pomicanje 64-bitnog argumenta x ulijevo za n bitova sa dodavanjem nula s desne strane

- + = zbrajanje *modulo* 2^{64}



Sl. 3.15. Stvaranje ulazne sekvence za SHA-512 procesiranje jednog bloka [1]

Vrijednost W_t u prvih 16 koraka procesa predstavlja odgovarajuću riječ u bloku podataka, a u ostalih 64 koraka vrijednost W_t se dobije kružnim pomicanjem 1 bita ulijevo dobivenog XOR-om četiri prethodne vrijednosti W_t.

Svaki bit *hash* vrijednosti u SHA-512 algoritmu je funkcija svakog bita na ulazu, stoga je malo vjerojatno da će dvije slučajno odabrane poruke imati jednaku *hash* vrijednost.

3.8. Kodovi za autentifikaciju poruke (MAC)

Jedno od najsloženijih područja kriptografije je autentifikacija poruke. Bilo bi gotovo nemoguće izdvojiti sve kriptografske funkcije i protokole koji su implementirani ili predloženi u svrhu autentifikacije poruke. Ovo poglavlje će se baviti temeljnim pristupom za autentifikaciju poruke pod nazivom MAC (eng. *Message Authentication Code*), odnosno kod za autentifikaciju poruke.

Prema [1] autentifikacija poruke je procedura pomoću koje se utvrđuje da primljena poruka dolazi iz navedenog izvora i da nije izmijenjena tijekom prijenosa. Funkcije koje se koriste u svrhu autentifikacije mogu se podijeliti u tri kategorija:

- *Hash* funkcije: funkcije koje mapiraju poruku proizvoljne duljine na *hash* vrijednost koja je fiksne duljine i koja služi kao autentifikator
- Šifriranje poruke: šifrat cijele poruke služi kao autentifikator
- Kodovi za autentifikaciju poruke: MAC kod je funkcija poruke i tajnog ključa koja daje vrijednost fiksne duljine koja služi kao autentifikator

MAC kod, poznat i pod nazivom kriptografska kontrolna suma, predstavlja blok podataka fiksne veličine koji se dodaje poruci, a generira se pomoću tajnog ključa uz pretpostavku da sudionici u komunikaciji dijele zajednički tajni ključ *K*. Jedan od načina konstrukcije MAC funkcija jest pridružiti tajni ključ *hash* funkciji bez ključa, tako što će ga se pridružiti kao dio poruke koja će se šifrirati [3]. Prema (3-7) [1] prilikom slanja poruke pošiljatelj računa MAC kao funkciju poruke i tajnog ključa:

$$\mathrm{MAC} = \mathrm{C}(K, M) \tag{3-7}$$

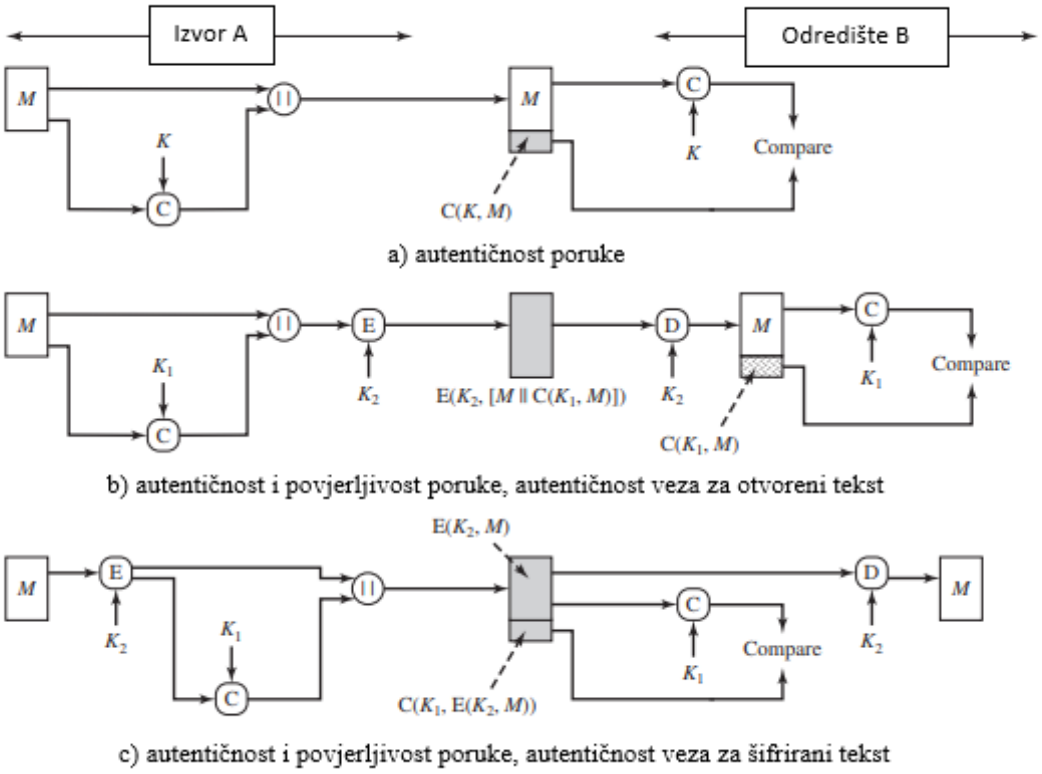
gdje je:

- M = ulazna poruka
- C = MAC funkcija
- K = zajednički tajni ključ
- MAC = kod za autentifikaciju poruke

Poruka se zajedno sa MAC kodom šalje primatelju koji nakon primitka poruke računa novi MAC kod pomoću navedene formule. Izračunati MAC se uspoređuje sa primljenim MAC-om, a ukoliko se izračunata i primljena MAC vrijednost podudaraju tada se može utvrditi sljedeće:

1) Primatelj je uvjeren da primljena poruke nije izmijenjena tijekom prijenosa
2) Primatelj je uvjeren da poruka dolazi od navedenog pošiljatelja
3) Ukoliko poruka uključuje broj sekvence, primatelj se može uvjeriti u pravilan broj sekvence primljene poruke

Razlika između MAC funkcije i enkripcije jest u tome što MAC algoritam ne treba biti reverzibilan dok enkripcija mora, jer inače ne bi bilo moguće dekriptirati poruku. MAC funkcije su generalno više naprema jedan funkcije.



Sl. 3.16. Osnovne primjene MAC koda [1]

Slika 3.16.a prikazuje autentičnost, no ne i povjerljivost MAC koda jer se cijela poruka prenosi do primatelja. Povjerljivost se može ostvariti enkripcijom poruke nakon MAC algoritma, prikazano na slici 3.16.b ili prije MAC algoritma, prikazano na slici 3.16.c. U prvom slučaju MAC se računa zajedno sa porukom kao ulaz, a potom se povezuje sa porukom te se cijeli blok šifrira. Na ovaj način autentifikacija je povezana sa otvorenim tekstom. U drugom slučaju je najprije šifrirana poruka, a potom se računa MAC pomoću dobivene šifrirane poruke te se spaja sa šifriranom porukom i kao takav čini blok podataka koji se prenosi. U ovom slučaju je autentifikacija povezana sa šifriranim tekstom. U oba slučaja su potrebna dva odvojena ključa koji moraju biti dijeljeni između pošiljatelja i primatelja.

3.8.1. Sigurnost MAC kodova

Napadi na MAC kodove mogu se podijeliti u dvije kategorije: napadi grubom silom i kriptoanaliza.

Napadi grubom silom

MAC funkcija generira MAC kod prema izrazu (3-8) [1]

$$T = \mathrm{MAC}(K, M) \tag{3-8}$$

gdje je

- M = ulazna poruka
- K = zajednički tajni ključ
- MAC(K, M) = autentifikator fiksne duljine poznat pod nazivom oznaka (engl. *tag*)

Napad grubom silom na MAC algoritam ovisi o relativnoj veličini tajnog ključa i oznake. MAC algoritam posjeduje sigurnosno svojstvo pod nazivom otpornost izračuna koje navodi sljedeće: uzimajući u obzir jedan ili više tekstualnih-MAC parova [x_i, MAC(K, x_i)], nije moguće izračunati proizvoljni tekstualni-MAC par [x, MAC(K, x)] za koji vrijedi da je proizvoljni novi ulaz $x \neq x_i$, odnosno napadač prilikom napada mora otkriti važeći MAC kod za dane poruke x. Napadač koristi dvije vrste napada:

1) Napad na tajni ključ: ako napadač može odrediti MAC ključ, tada može generirati važeći MAC kod za bilo koju ulaznu poruku *x*. Neka je veličina MAC ključa *k* bita i neka napadač posjeduje jedan par tekstualne-oznake, tada napadač može izračunati *n*-bitnu oznaku na danom tekstu za sve moguće MAC ključeve.
2) Napad na MAC vrijednost: napadač pokušava odrediti važeću oznaku za danu poruku kako bi pronašao poruku koja se podudara sa danom oznakom.

Kriptoanaliza

Kriptoanaliza nastoji iskoristiti određena svojstva MAC algoritma kako bi se izveo određeni napad. Mjerenje otpora MAC algoritma na kriptoanalizu predstavlja usporedbu snage algoritma sa naporom potrebnim za napad grube sile na MAC algoritam, odnosno idealan MAC algoritam će zahtijevati napor kriptoanalize veći ili jednak naporu napada grube sile [1].

3.8.2. MAC kodovi temeljeni na *hash* funkcijama (HMAC)

MAC kodovi su se većinom konstruirali na temelju simetričnih blokovnih šifri, no posljednjih godina se povećao interes u razvoj MAC kodova izvedenih iz kriptografskih *hash* funkcija. Porast interesa je proizašao zbog sljedećih karakteristika *hash* funkcija:

1. Kriptografske *hash* funkcije općenito brže izvršavaju zadatke u usporedbi sa simetričnim blokovnim šiframa
2. Biblioteka koda kriptografskih *hash* funkcija je široko dostupna

Kriptografske *hash* funkcije poput SHA funkcije nisu namijenjene za upotrebu u MAC kodovima jer se ne oslanjaju na upotrebu tajnog ključa, no pojavio se niz prijedloga za ugradnju tajnog ključa u postojeći *hash* algoritam. Prijedlog koji je stekao najviše podrške je HMAC, koji je izdan kao RFC 2104 i odabran kao obavezan pristup za implementaciju u MAC.

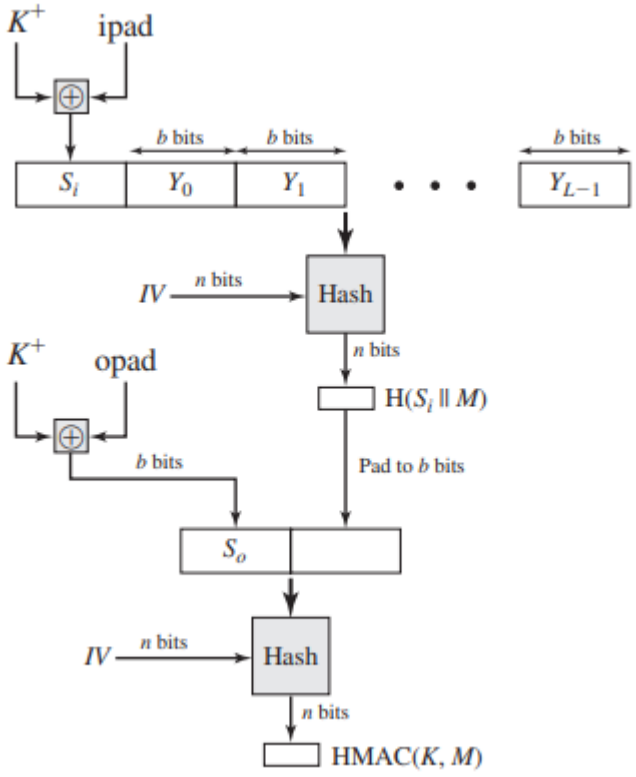
RFC 2104 navodi sljedeće ciljeve dizajna HMAC koda:

- Upotreba *hash* funkcija koje dobro funkcioniraju u softveru i čiji kod je besplatan i dostupan

- Omogućiti jednostavne zamjene ugrađenih *hash* funkcija ako su potrebne sigurnije ili brže *hash* funkcije
- Očuvati izvorne performanse *hash* funkcije
- Koristiti ključeve na jednostavan način
- Imati dobro shvaćenu kriptografsku analizu snage mehanizma autentičnosti na temelju razumnih pretpostavki o ugrađenoj *hash* funkciji

HMAC tretira *hash* funkciju kao *crnu kutiju*, što omogućava korištenje postojeće implementacije *hash* funkcije kao modul pri implementiranju HMAC-a. Ako je potrebno zamijeniti postojeću *hash* funkciju u HMAC implementaciji, sve što treba učiniti jest zamijeniti postojeći modul *hash* funkcije sa novim modulom [1].

HMAC algoritam



Sl. 3.17. Struktura HMAC koda [1]

Slika 3.17. prikazuje način rada HMAC koda, pri čemu je:

- H = ugrađena *hash* funkcija (npr. MD5, SHA-1)

- IV = početna vrijednost ulaza u *hash* funkciju

- M = poruka na ulazu u *hash* funkciju

- Y_i = i-ti blok od M

- L = broj blokova u M

- b = broj bita u pojedinom bloku

- n = duljina *hash* koda

- K = tajni ključ

- K^+ = K dopunjen nulama sa lijeve strane kako bi rezultat bio b bita u duljini

HMAC algoritam se prema (3-9) [1] može izraziti:

$$\mathrm{HMAC}(K, M) = \mathrm{H}[(K^+ \oplus opad) \} \mathrm{H}[(K^+ \oplus ipad) \} \mathrm{M}]] \qquad (3\text{-}9)$$

pri čemu je :

- *ipad* = 00110110 (36 u heksadecimalnom zapisu) ponovljen b/8 puta

- *opad* = 01011100 (5C u heksadecimalnom zapisu) ponovljen b/8 puta

Algoritam je moguće opisati na sljedećom metodom:

1. Dodaj nule na lijevi kraj K kako bi se kreirao b-bitni niz K^+
2. XOR K^+ sa *ipad* kako bi se kreirao b-bitni blok S_i
3. M dodaj na S_i
4. Primijeni H na niz generiran u koraku 3.
5. XOR K^+ sa *opad* kako bi se kreirao b-bitni blok S_0
6. *Hash* rezultat iz koraka 3. dodaj na S_0
7. Primijeni H na niz kreiran u prethodnom koraku te ispiši rezultat

HMAC bi trebao izvršiti zadatak otprilike u isto vrijeme kao i ugrađeni *hash* algoritam za dugačke poruke.

3.8.3. Sigurnost HMAC koda

Sigurnost HMAC koda ovisi o kriptografskoj snazi ugrađene *hash* funkcije. Dokazano je da za poruke koje je kreirao izvor te ih je vidio napadač, vjerojatnost uspješnog napada ovisi o sljedećim napadima na ugrađenu *hash* funkciju:

1. Napadač može izračunati izlaz iz funkcije kompresije
2. Napadač pronalazi kolizije u *hash* funkciji čak i bez poznavanja *IV*.

Prilikom prvog napada, *IV hash* funkcije je zamijenjena tajnom, a napad na *hash* funkciju zahtijeva napad grubom silom na ključ. Prilikom drugog napada napadač traži dvije poruke M i M' koje proizvode istu *hash* vrijednost. Ovaj napad zahtijeva razinu napora od $2^{n/2}$ za *hash* vrijednost duljine n [1].

4. PRAKTIČNI PRIMJERI PRIMJENE *HASH* FUNKCIJA

U svrhu analize *hash* funkcija, u radu će se koristiti aplikacije *Cryptool 1.4.41* i *Cryptool 2.1*. *Cryptool* je besplatan Windows program za kriptografiju i kripoanalizu, te je najrašireniji softver e-učenja takve vrste. Izvorno je dizajniran kao poslovna aplikacija za obuku informacijske sigurnosti, no razvio se u *open-source* projekt u području kriptografije i informatičke sigurnosti [5].

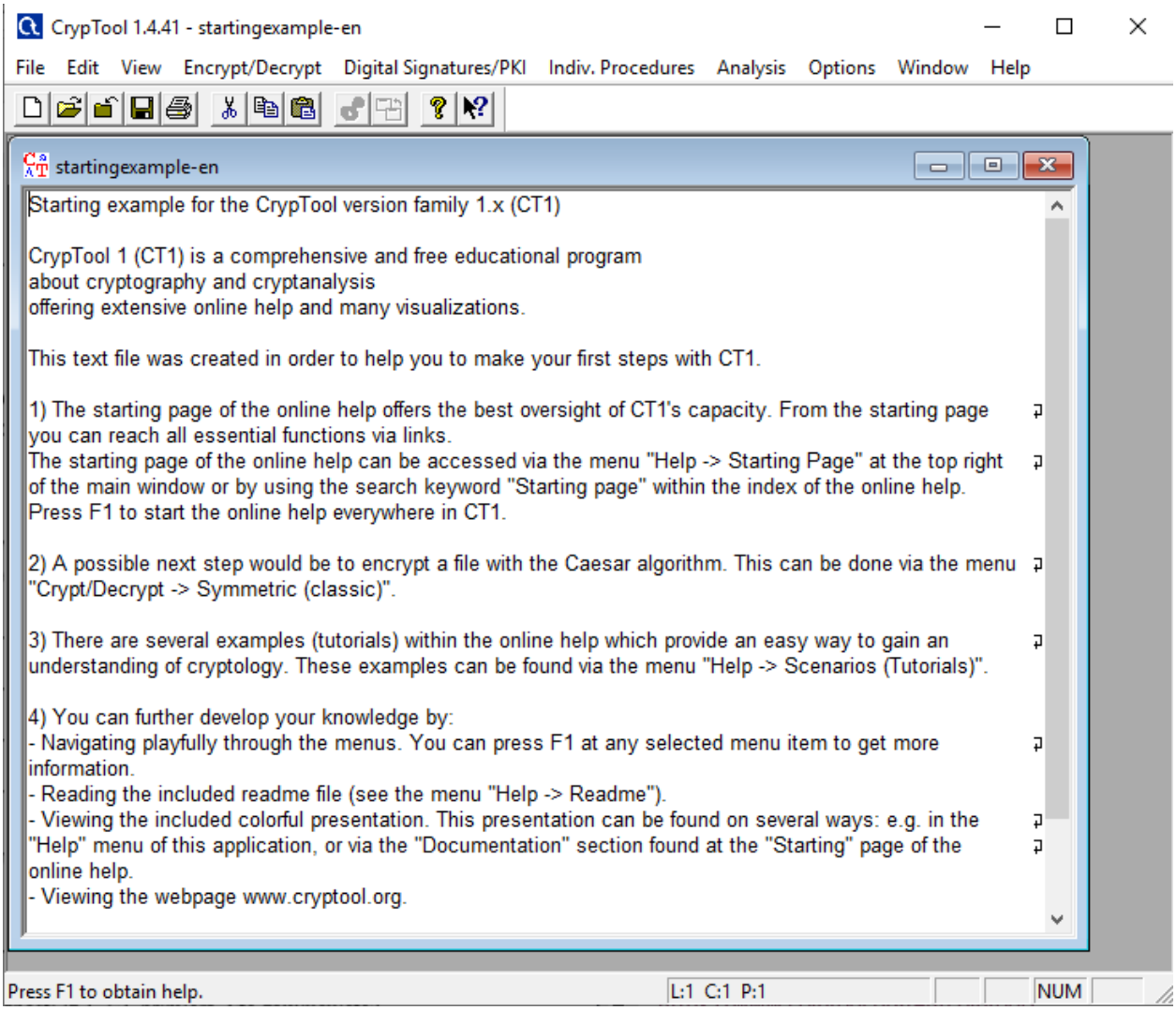
Cryptool implementira više od 400 algoritama koje korisnici mogu prilagoditi vlastitim parametrima. Program sadrži većinu klasičnih šifri kao i moderne simetrične i asimetrične metode kriptografije koje uključuju *Rivest–Shamir–Adleman* (RSA), digitalni potpis, hibridno šifriranje i razmjenu ključeva *Diffie-Hellman*. *Cryptool 2.1* također sadrži grafičko sučelje, alate za analizu i algoritme koji upoznavaju korisnika s područjem kriptografije.

Grafičko sučelje omogućuje vizualno programiranje, odnosno tijek rada se može vizualizirati i kontrolirati kako bi se omogućila intuitivna manipulacija kriptografskih funkcija. Također je moguće vizualizirati unutarnje operacije algoritma, što korisniku omogućuje shvaćanje svih detalja proizvoljno odabranog kriptografskog algoritma i primjenu određenog algoritam u stvarnom scenariju. Vektorski orijentirano grafičko korisničko sučelje omogućuje korisniku skaliranje trenutnog prikaza po želji. Program također sadrži integriranu online pomoć koja obuhvaća upute o korištenju funkcije i teorijsku podlogu sa referencama [6].

Hash funkcije će se prvo analizirati u programu *Cryptool 1.4.41*, a potom u programu *Cryptool 2.1*

4.1. Primjeri primjene *Cryptool 1.4.41* aplikacije pri određivanju *hash* funkcija

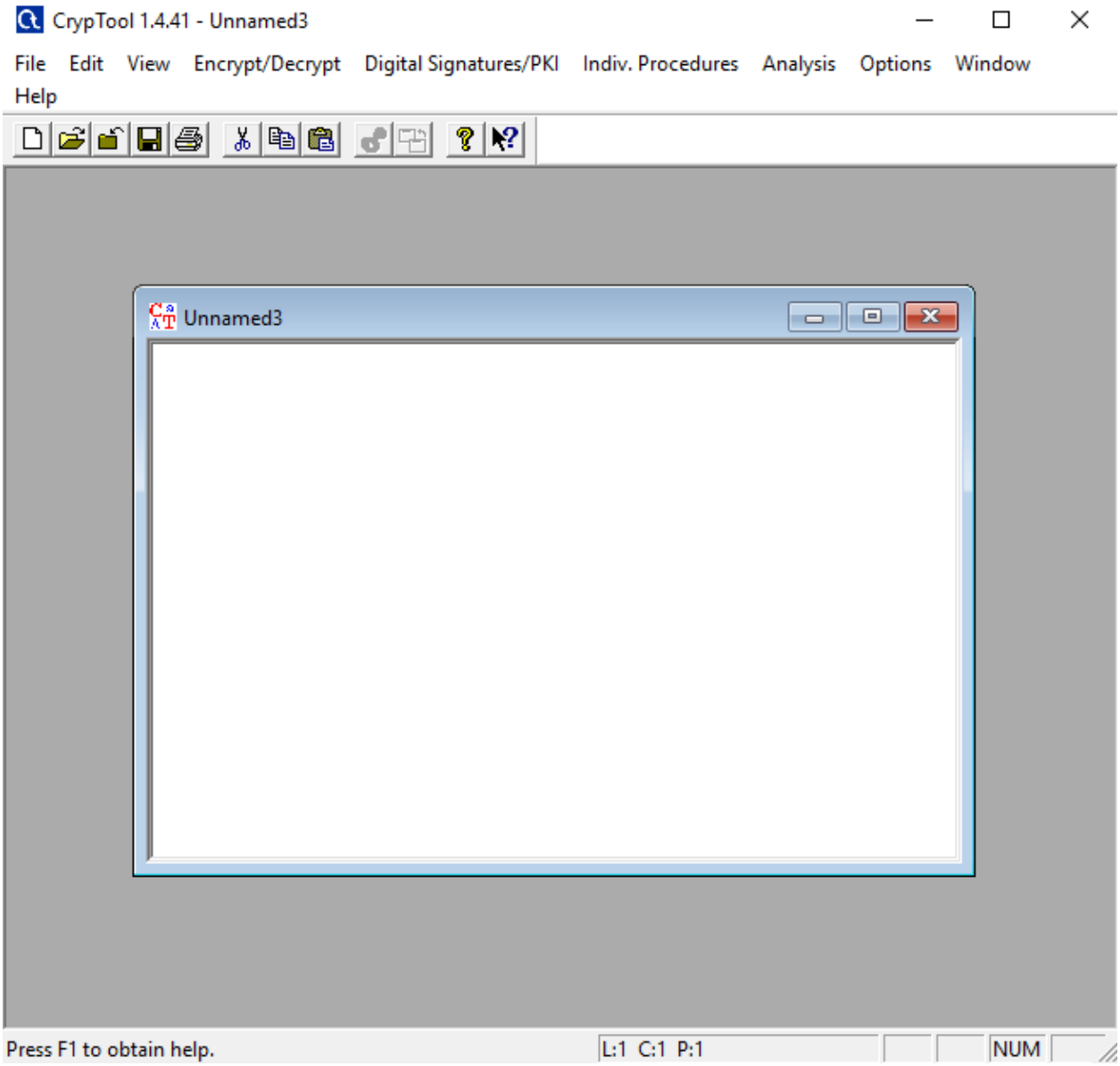
Potrebno je pokrenuti program *Cryptool 1.4.41* nakon čega bi se trebalo otvoriti grafičko sučelje, slika 4.1.



Sl. 4.1. Grafičko sučelje *Cryptool 1.4.41*

4.1.1. Prvi primjer - Rad sa *hash* funkcijama

Potrebno je napraviti novu datoteku odabirom naredbe *File* → *New*, nakon čega se otvara novi skočni prozor u koji je moguće unijeti tekst, prikazano na slici 4.2.

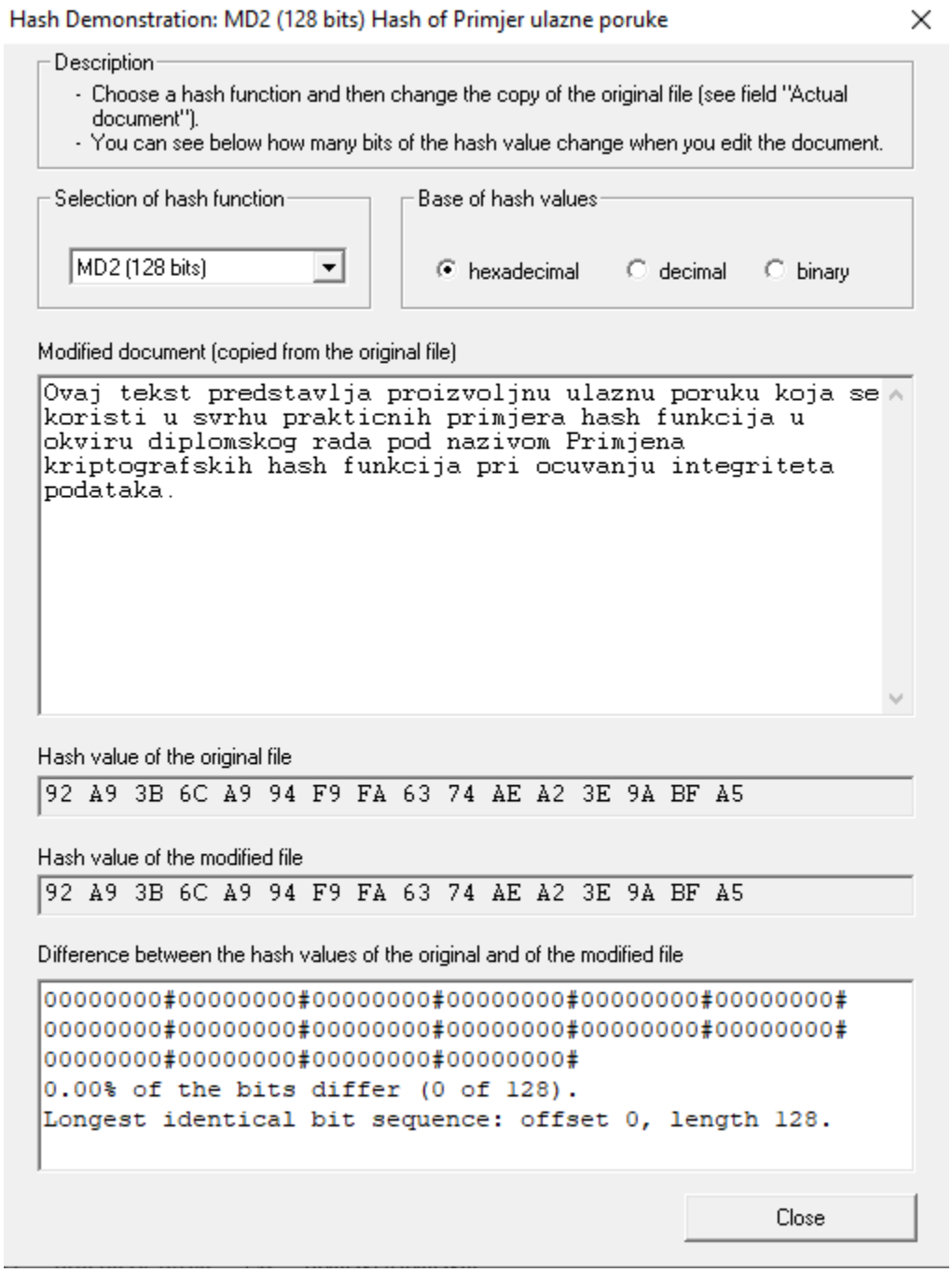


Sl. 4.2. Skočni prozor nove datoteke

Potrebno je unijeti proizvoljnu ulaznu poruku, a u ovom radu će biti korištena sljedeća poruka:

Ovaj tekst predstavlja proizvoljnu ulaznu poruku koja se koristi u svrhu prakticnih primjera hash funkcija u okviru diplomskog rada pod nazivom Primjena kriptografskih hash funkcija pri ocuvanju integriteta podataka.

Datoteku je potrebno spremiti odabirom naredbe *File → Save as…,* dodijeliti ime te odabrati mjesto pohrane datoteke. Kako bi se mogla prikazati sigurnosna svojstva *hash* funkcija i usporediti *hash* vrijednost tekstualne datoteke sa *hash* vrijednosti izmijenjene tekstualne datoteke potrebno je otvoriti dijalog odabirom naredbi u programskoj traci: *Indiv. Procedures → Hash → Hash Demonstration.*



Sl. 4.3. Izgled dijaloga *hash* funkcija

Korisnik može izabrati sljedeće *hash* funkcije: MD2, MD4, MD5, SHA, SHA-1, SHA-256, SHA-512 i RIPEMD-160. *Hash* vrijednosti mogu biti ispisane u heksadecimalnom, decimalnom i binarnom obliku. U radu će se koristiti *hash* funkcije: MD2, MD4, MD5, SHA-1, SHA-256, SHA-512, a *hash* vrijednosti će biti ispisane u heksadecimalnom obliku.

U prozoru dijaloga redom će se odabirati navedene *hash* funkcije, te će se zapisati dobivene *hash* vrijednosti:

- **MD2**: 92 A9 3B 6C A9 94 F9 FA 63 74 AE A2 3E 9A BF A5
- **MD4**: 31 DC 5B 54 DE FA 69 2F 0E 6D 29 B3 8A CD 6B 52
- **MD5**: C6 EF 16 C1 D5 F8 E1 C5 9C 29 A0 C4 43 05 B9 D8
- **SHA-1**: 7E 29 96 DC 5D 1F 84 E1 63 D8 AB C1 F2 D6 8E 4F 11 33 A8 6B
- **SHA-256**: 87 64 1E 39 8C 31 FE 79 BB CA CC B9 77 DF F3 EA 4B 5B 69 2E A8 D7 7B 05 09 3C F6 F1 3A A4 69 CF
- **SHA-512**: 72 83 0D 20 34 15 04 0E 86 A5 86 E1 7E CE 5B 1B 8B EF D3 12 B7 8D B6 1B AB 3E D5 D9 41 35 87 5E A6 FE F6 78 86 73 DF C4 E0 9A 3D E6 B3 50 B3 AA 94 7B E3 CB 40 40 A6 AB D3 89 D4 08 6A 09 DB 2D

Izmijenit će se ulazna poruka tako što će se obrisati zadnja riječ zajedno s točkom, pa će poruka glasiti:

Ovaj tekst predstavlja proizvoljnu ulaznu poruku koja se koristi u svrhu prakticnih primjera hash funkcija u okviru diplomskog rada pod nazivom Primjena kriptografskih hash funkcija pri ocuvanju integriteta

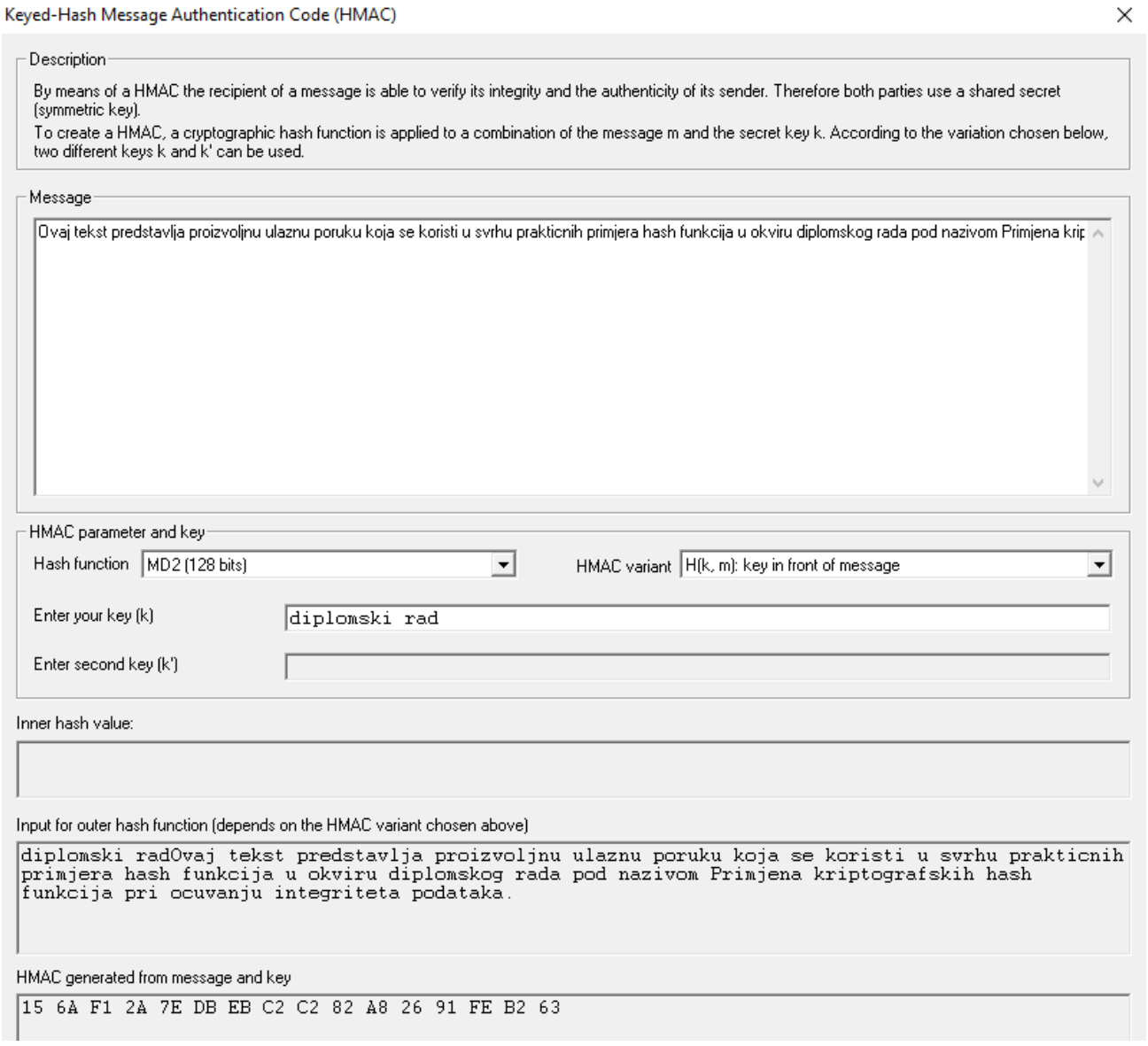
Nakon izmjene poruke, potrebno je ponovno zapisati dobivene *hash* vrijednosti:

- **MD2**: 92 1C 38 F2 DC B0 70 E6 A1 4C 96 73 D6 8C 35 1C
- **MD4**: 7A D2 2B 73 9B 45 36 AE 42 00 0B 38 0B 83 0F 5D
- **MD5**: D3 A2 F9 B6 C3 B1 6C 27 14 BC B3 9F 8B B2 AA C0
- **SHA-1**: 3D 9A EE EC BB AE 90 5C 56 7A 23 B3 04 D0 B3 B1 80 50 DA 2A
- **SHA-256**: C3 F9 66 1A 42 1A 45 41 1F CC 6C 0B 71 9C AF 78 6D A5 60 1D A1 22 B4 47 29 9C 99 E8 FD 82 08 2D
- **SHA-512**: 9D E6 00 D6 1E 22 D8 7A 03 99 99 D7 36 A6 36 E6 77 42 C9 77 2F 54 97 68 B5 BA F7 A4 D2 AF 8A 00 B1 67 15 E9 05 E4 4A 72 27 6F C4 59 36 F2 FA B5 CD 78 6D 96 58 82 80 EC 00 00 93 A4 3E E7 77 A6

4.1.2. Drugi primjer - Rad sa HMAC funkcijama

Potrebno je ponoviti korake iz prvog primjera, stvoriti novu datoteku te unijeti ulaznu poruku. U ovom primjeru će se koristiti ista ulazna poruka kao i u prvom primjeru.

Kako bi se mogao prikazati način rada HMAC algoritma koristeći *hash* funkcije i generiranje HMAC koda, potrebno je otvoriti dijalog odabirom naredbi u programskoj traci: *Indiv. Procedures* $\rightarrow$ *Hash* $\rightarrow$ *Generation of HMACs…*



Sl. 4.4. Dijalog HMAC koda

Pomoću ovog dijaloga korisnik može generirati HMAC za ulaznu poruku. HMAC vrijednost je prikazana u heksadecimalnom obliku. Potrebno je odabrati željenu *hash* funkciju i jednu od pet HMAC varijanti:

1. H (k, m) : ključ ispred poruke
2. H (m, k) : ključ na kraju poruke
3. H (k, m, k) : ključ ispred i na kraju poruke
4. H (k, m, k') : različiti ključevi
5. H (k , H (k, m)) : dvostruko *hashiranje*

gdje je

H = *hash* funkcija

k = ključ

m = poruka

Zatim je potrebno odrediti ključ pomoću kojega će se generirati HMAC. U ovom radu će se koristiti ključ: diplomski rad.

Potrebno je odabrati različite kombinacije *hash* funkcija i HMAC varijanti te zapisati dobivene HMAC vrijednosti:

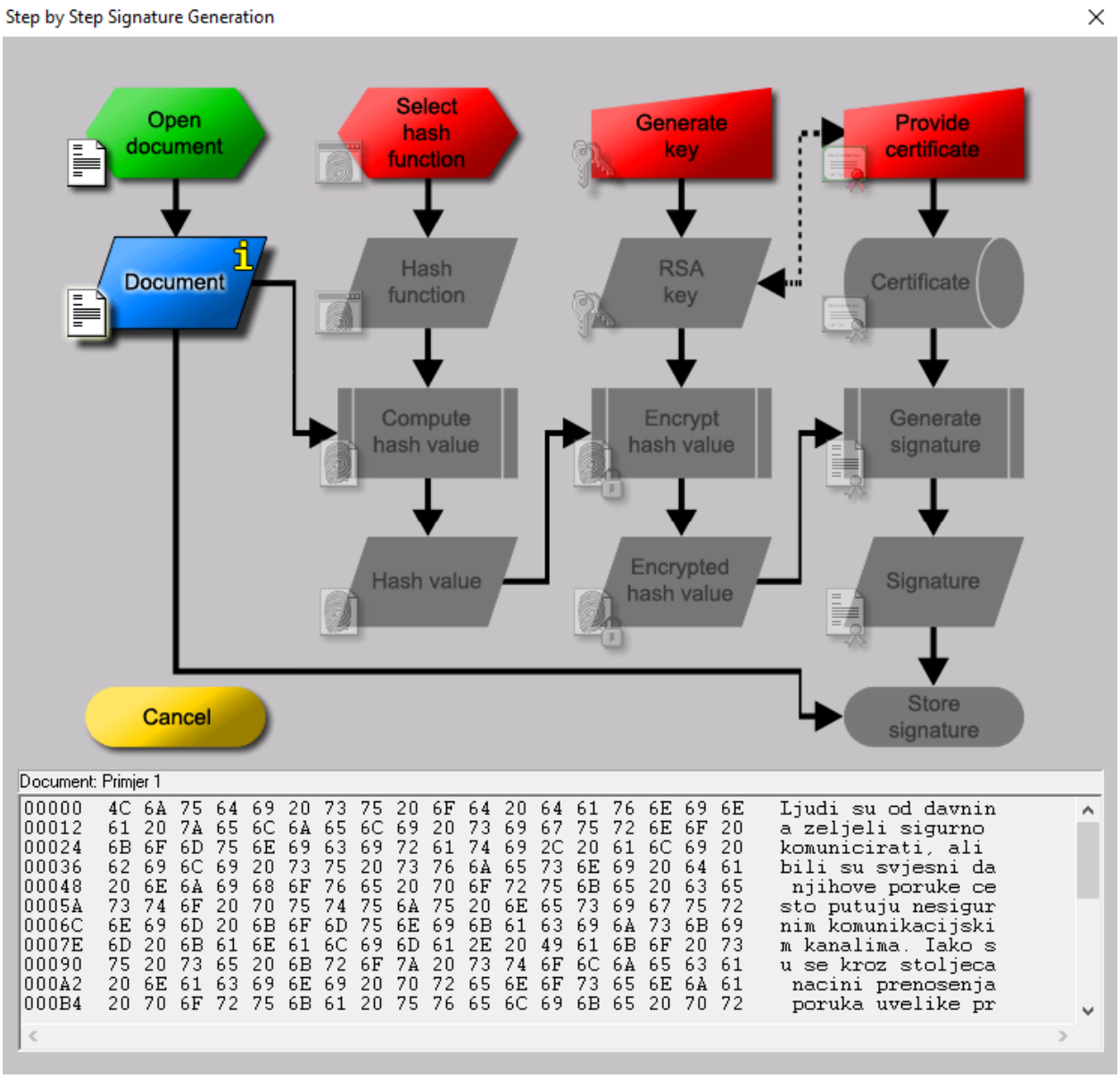
- **MD2, H(k, m)**: 15 6A F1 2A 7E DB EB C2 C2 82 A8 26 91 FE B2 63
- **MD2, H(k, m, k)**: A7 BB 25 20 2A 80 D7 6E 72 F7 97 9A 6B 17 D6 B7
- **MD4, H(k, m)**: 3D 50 C3 68 E6 8C 86 5C D9 7B 22 F1 CE 39 7C CE
- **MD4, H(m, k)**: 43 F9 57 0E 67 50 EE 38 15 9A 6F 74 39 72 41 42
- **MD5, H(m, k)**: B1 3C 31 2F E8 0A D7 56 E3 A4 2C 93 B1 30 48 3F
- **MD5, H(k, m, k)**: AA FE 4A 11 1C AA 4A 1A 04 F7 0E 0B 69 19 CF 2A
- **SHA-1, H(k, m, k)**: 4E B2 B2 44 66 5D 75 05 78 70 00 D4 A1 01 13 25 B3 61 D1 51
- **SHA-1, H(k, m)**: CA 46 CC 3E CC C4 8E 6B 09 FF EF D7 07 9A F3 14 7C 32 0D 27
- **SHA-256, H(k, m)**: 7 CA 83 2F F0 55 54 2C 45 88 04 EE D1 66 D6 73 54 13 0E EE ED 4E DA AF 46 DE 63 E3 F8 95 BD 8E

- **SHA-256, H(k, m, k)**: 63 3E FA 61 7D E4 44 DB DD B4 F3 7E 4F D4 06 B7 93 48 D7 7A DF 2A D3 8D 93 D2 18 EA 81 89 6E 37

4.1.3. Treći primjer - Digitalni potpis

Prema [7] digitalni potpis se kreira kroz više koraka, stoga je potrebno pratiti i razumjeti sve korake kako bi se mogao generirati digitalni potpis.

Potrebno je otvoriti grafičko sučelje programa *Cryptool 1.4.41* prikazano na slici 4.1., upisati proizvoljan tekst ili otvoriti tekstualni dokument te odabrati naredbu *Digital Signatures/PKI → Signature Demonstration(Signature Generation)…,* nakon čega će se otvoriti prozor *Step by Step Signature Generation* prikazan na slici 4.5.

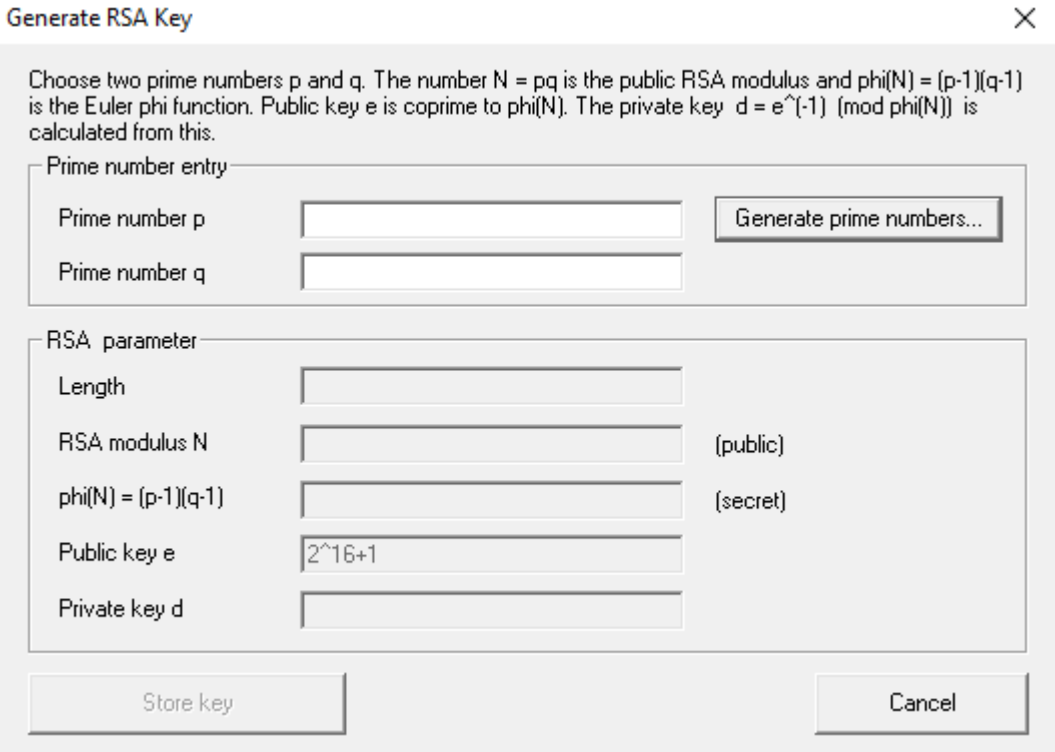


Sl. 4.5. *Step by Step Signature Generation*

Odabirom naredbe *Open document* korisnik može učitati proizvoljnu tekstualnu datoteku, a u ovom primjeru se koristi isti primjer tekstualne datoteke kao u prethodna dva primjera. Nakon učitavanja tekstualne datoteke potrebno je odabrati *hash* algoritam odabirom naredbe *Select hash function*, a u radu će se koristiti *MD5* algoritam. Nakon odabira željenog *hash* algoritma, potrebno je odabrati naredbu *Compute hash value* kako bi se izračunala *hash* vrijednost. Odabirom naredbe *Hash value* ispisat će se dobivena *hash* vrijednost, koja u slučaju odabira *MD5* algoritma glasi:

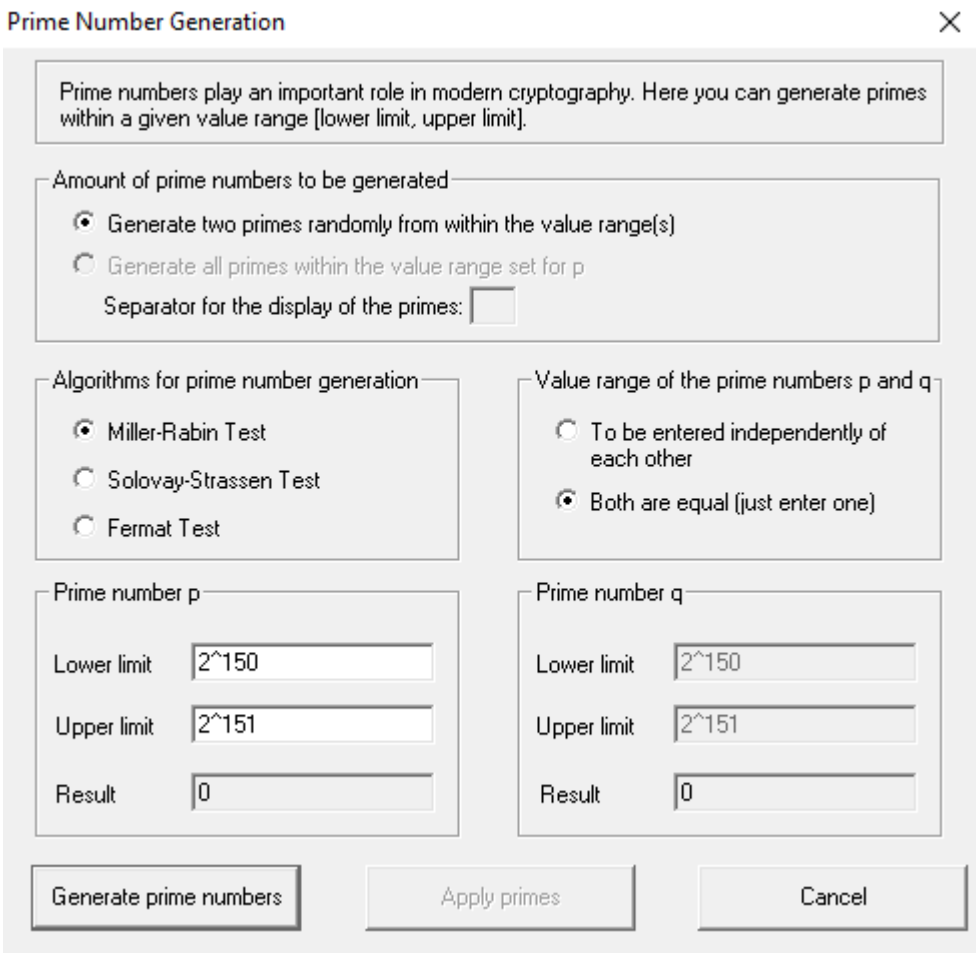
06 A2 48 25 F8 AF 02 C7 2C 1E 6E C9 A4 16 6C ED

Zatim je potrebno generirati privatni i javni ključ odabirom naredbe *Generate key,* nakon čega se otvara prozor prikazan na slici 4.6. Kako bi se ključevi mogli generirati, potrebna su dva velika prosta broja *p* i *q* i matematička funkcija.



Sl. 4.6. Generiranje privatnog i javnog ključa

Odabirom naredbe *Generate prime numbers…* otvara se prozor *Prime Number Generation* prikazan na slici 4.7., u kojem je moguće odabrati količinu prostih brojeva koji će se generirati, algoritam za generiranje prostih brojeva i raspon vrijednosti prostih brojeva. Na slici 4.7. prikazan su parametri generiranja prostih brojeva koji su korišteni u ovom radu.



Sl. 4.7. Generiranje prostih brojeva

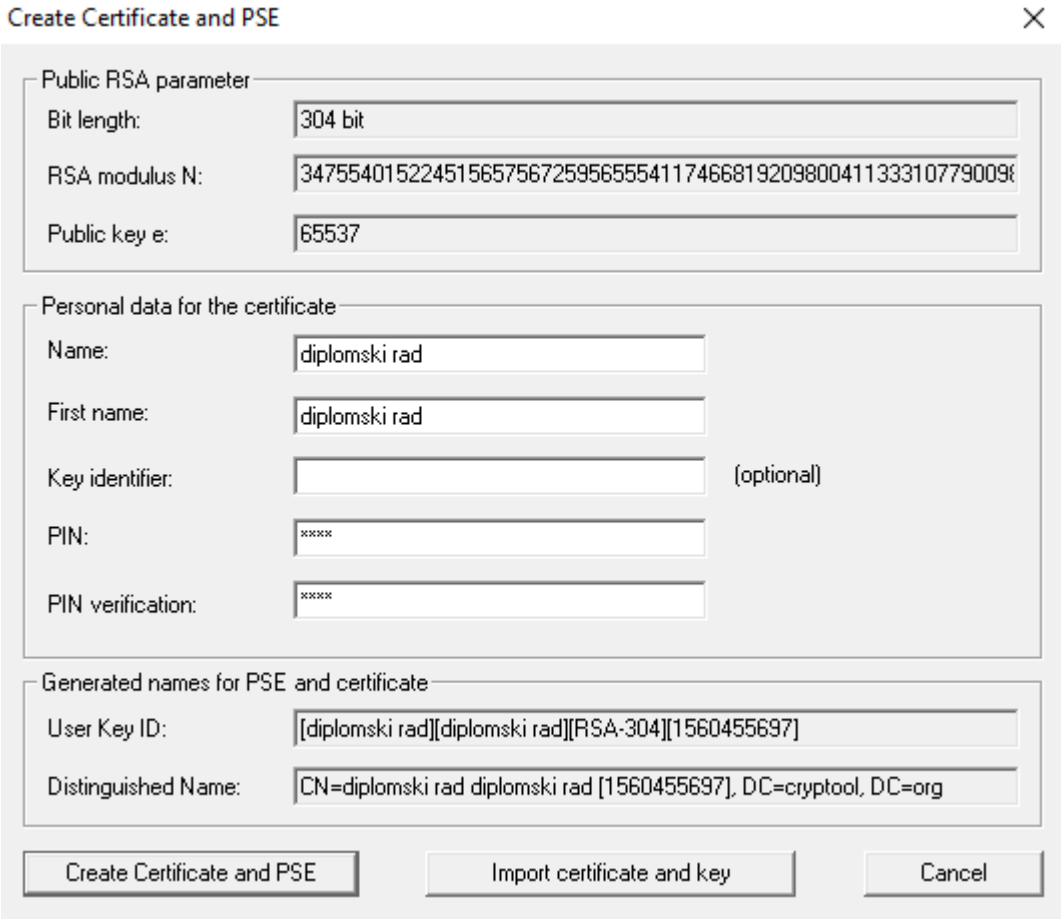
Potrebno je odabrati željene parametre, potom naredbu *Generate prime numbers,* a potom *Apply primes* kako bi se uspješno generirali prosti brojevi. Nakon generiranja prostih brojeva, program je izračunao privatni i javni ključ na sljedeći način:

Broj $N = pq$ predstavlja RSA module, a $phi(N) = (p-1)(q-1)$ predstavlja Eulerovu phi funkciju. Javni ključ je jednak *phi(N),* a privatni ključ se računa iz funkcije $d = e^{(-1)}*(mod\ phi(N))$.

Nakon računanja ključeva potrebno je odabrati naredbu *Store key*, potom je potrebno šifrirati *hash* vrijednost sa privatnim ključem odabirom naredbe *Encrypt hash value* nakon čega će se prikazati šifrirana *hash* vrijednost koja u ovom primjeru glasi:

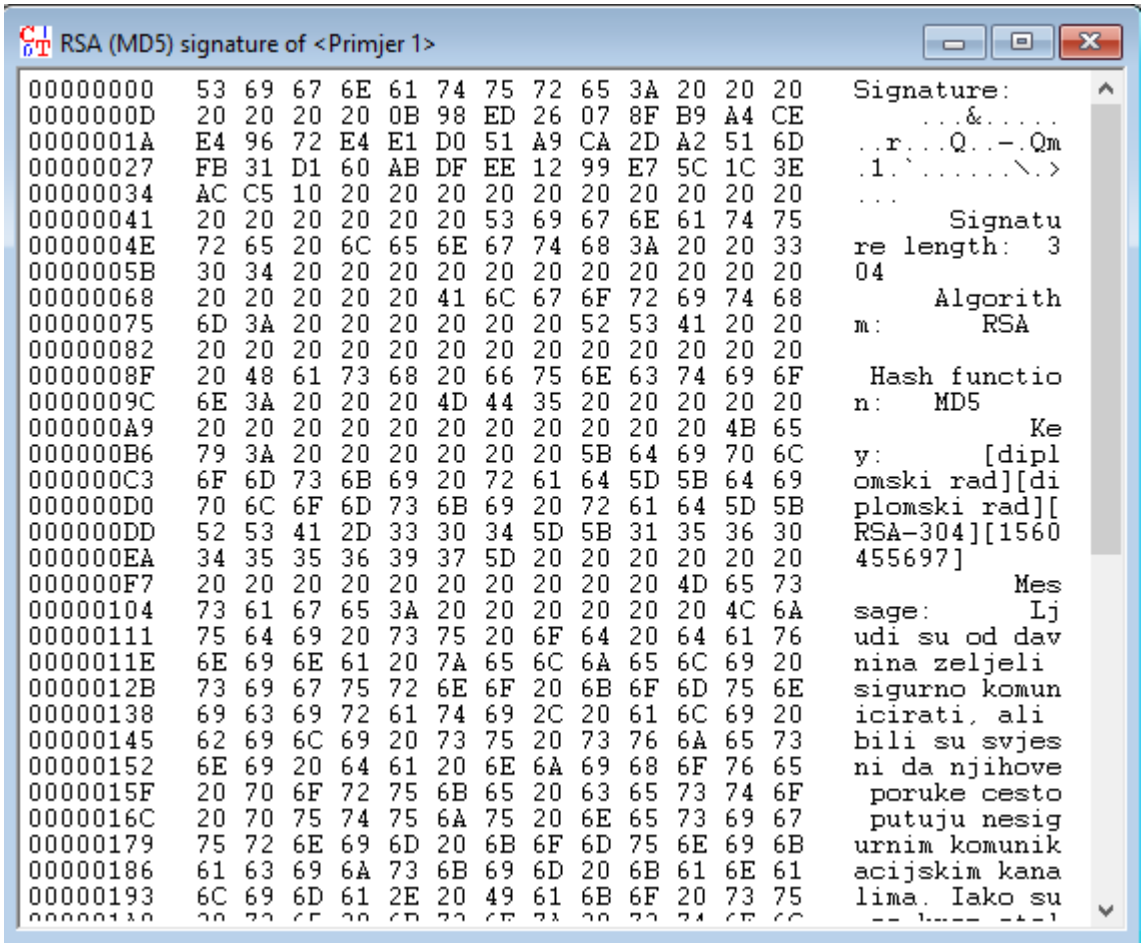
0B 98 ED 26 07 8F B9 A4 CE E4 96 72 E4 E1 D0 51 A9 CA 2D A2 51 6D FB 31 D1 60 AB DF EE 12 99 E7 5C 1C 3E AC C5 10

Idući korak predstavlja kreiranje certifikata povezanog sa ključem odabirom naredbe *Provide certificate*, nakon čega će se otvara prozor prikazan na slici 4.8. u kojem je potrebno odabrati ime certifikata i pin.



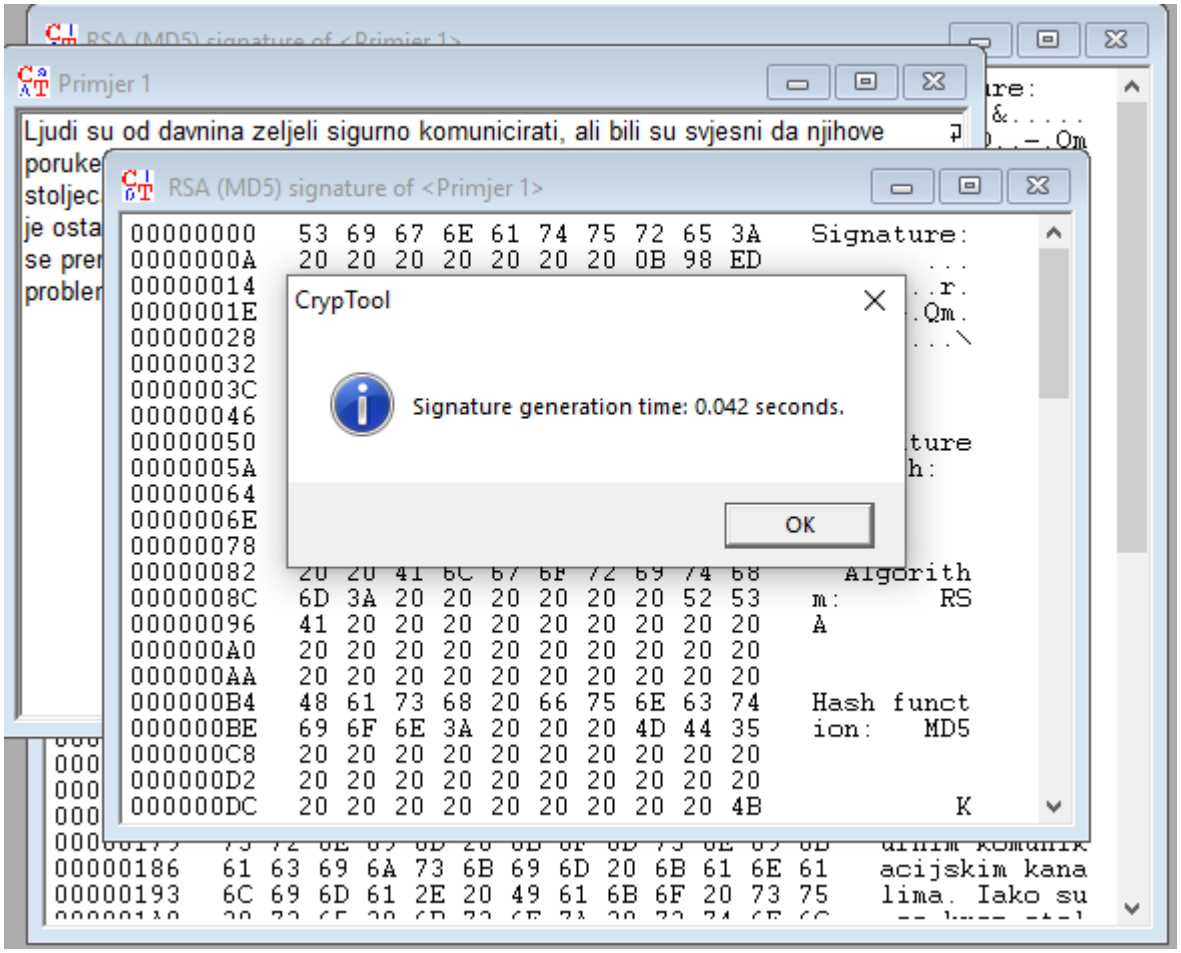
Sl. 4.8. Kreiranje certifikata

Nakon unesenog imena i pina, odabrati naredbu *Create certificate and PSE*, zatim naredbu *Create signature*, a potom naredbu *Store signature* nakon koje će se kreirati potpis. Primjer potpisa prikazan je na slici 4.9.



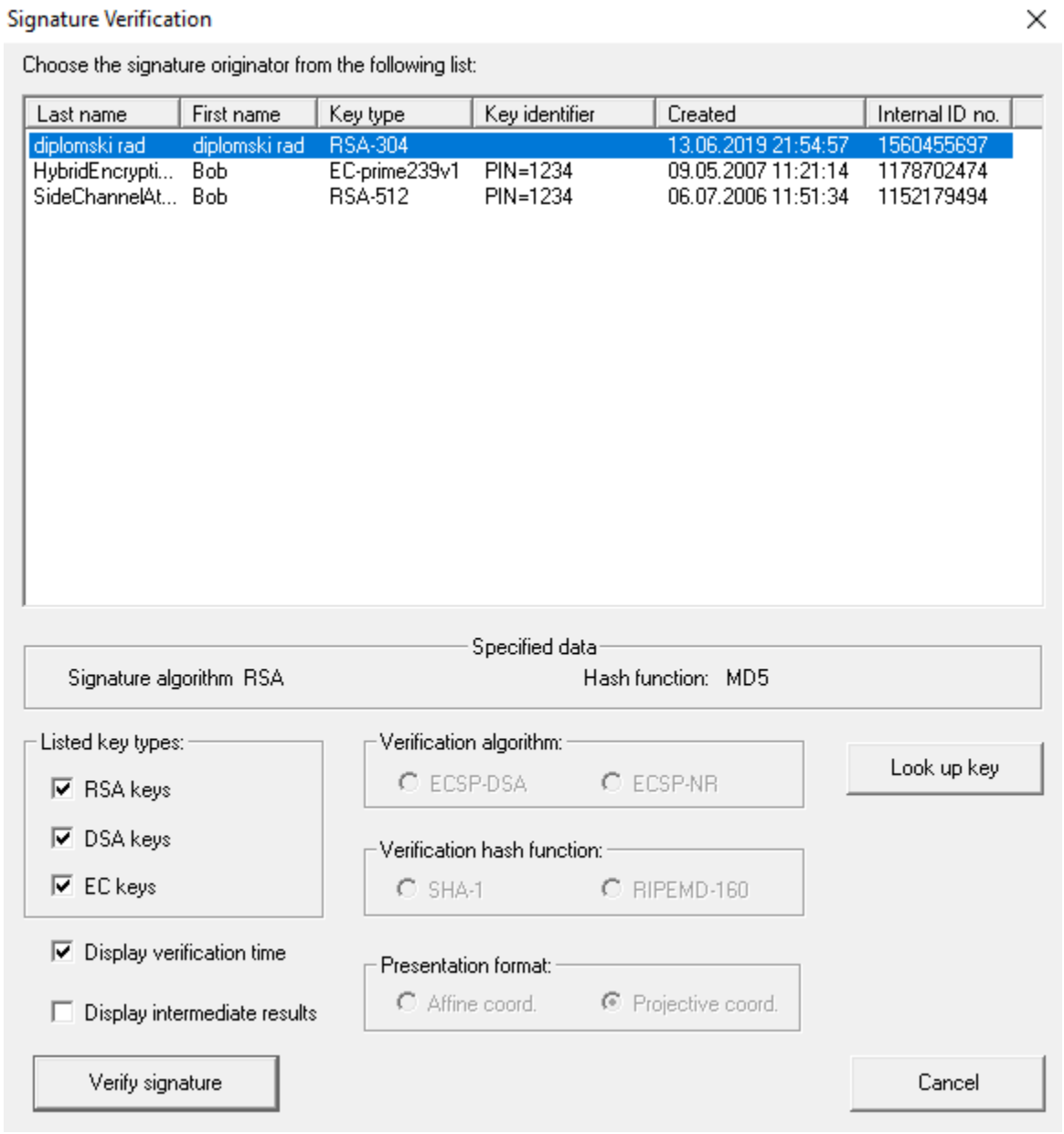
Sl. 4.9. Digitalni potpis dokumenta Primjer 1

Kako bi korisnik mogao potpisati željeni dokument, potrebno je odabrati željeni dokument, zatim naredbu *Digital Signatures/PKI → Sign Document…* te unijeti pin koji je odabran prilikom kreiranja potpisa i odabrati naredbu *Sign.*



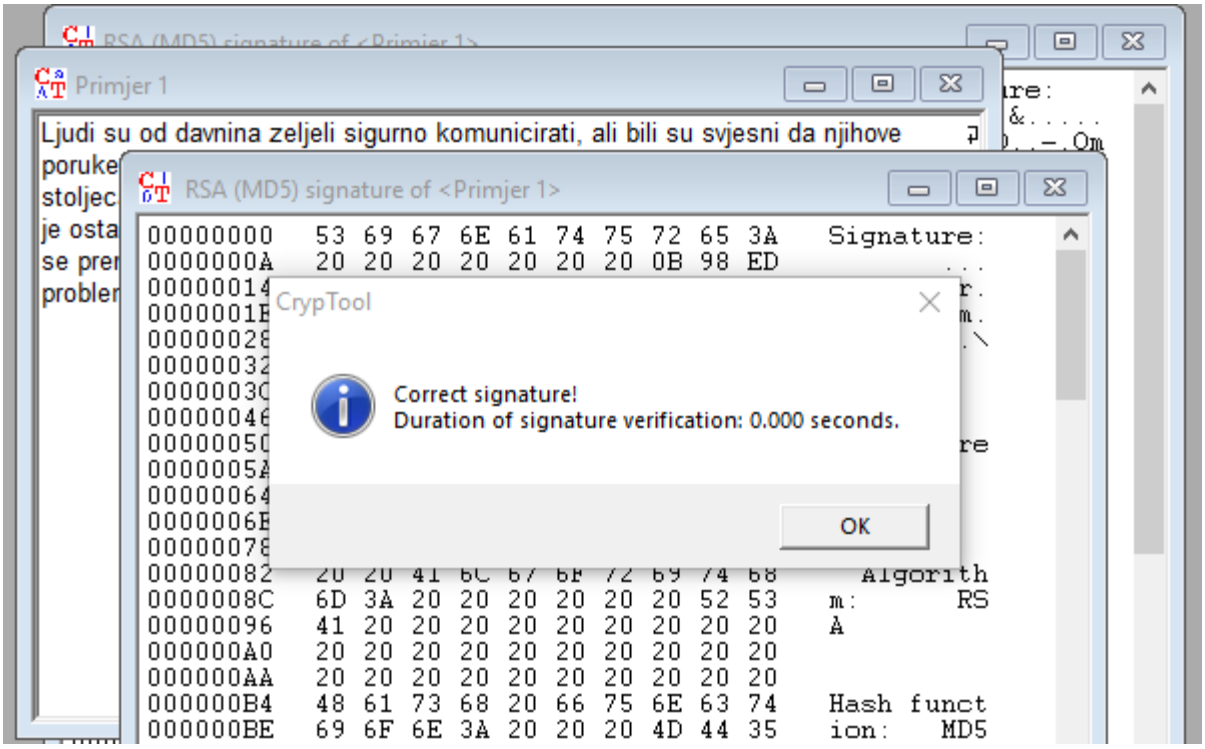
Sl. 4.10. Vrijeme potrebno za generiranje potpisa

Posljednji korak je provjera digitalnog potpisa, potrebno je odabrati naredbu *Digital Signatures/PKI → Verify Signature…,* a potom odabrati kreirani digitalni potpis te naredbu *Verify Signature*, što je prikazano na slici 4.11.



Sl. 4.11. Provjera potpisa

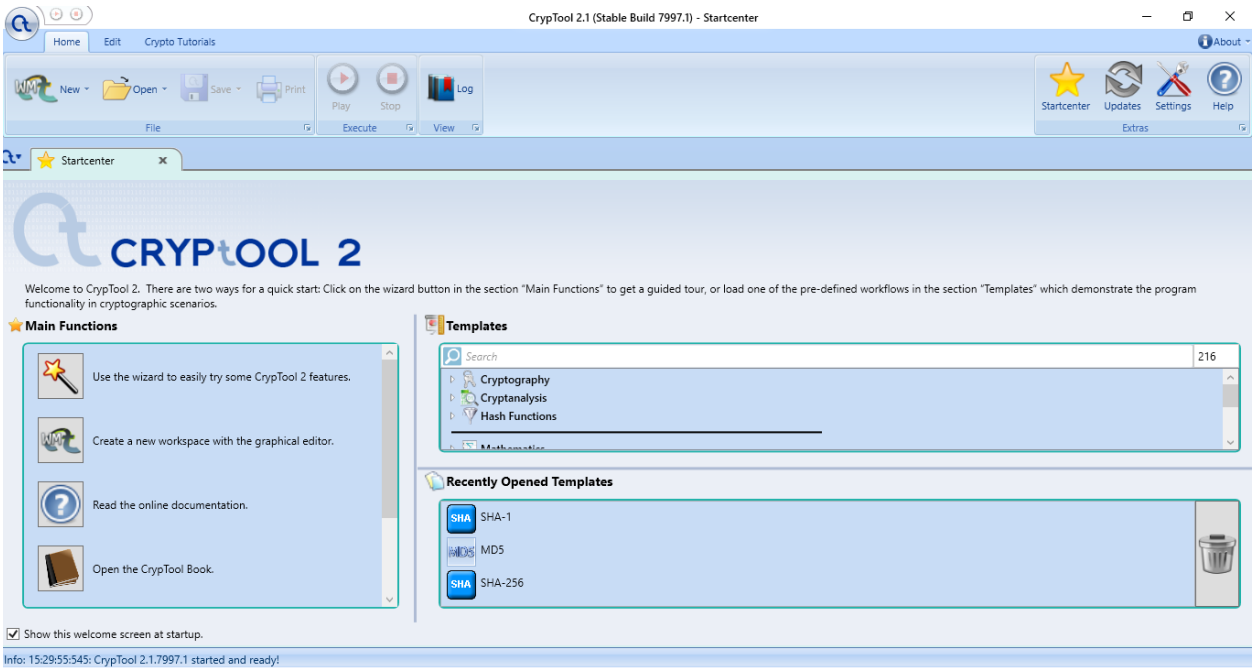
Ako je potpis ispravan, pojavit će se poruka koja potvrđuje ispravan potpis, prikazana na slici 4.12.



Sl. 4.12. Potvrda ispravnog potpisa

4.2. Primjeri primjene *Cryptool 2.1* aplikacije pri određivanju *hash* funkcija

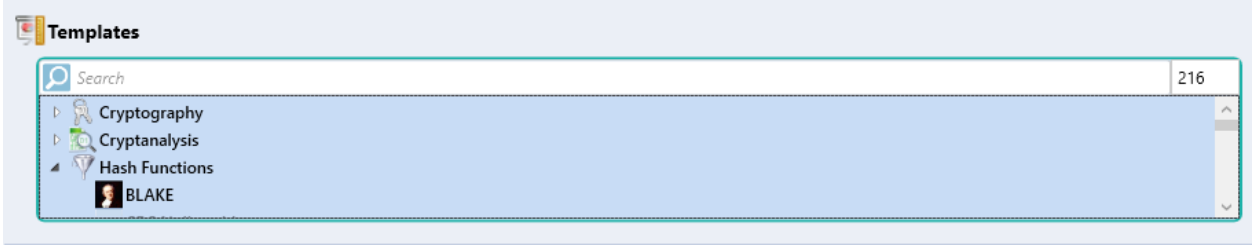
Potrebno je pokrenuti program *Cryptool 2.1* nakon čega će se otvoriti kartica *Startcenter* prikazana na slici 4.13.



Sl. 4.13. Grafičko sučelje *Cryptool 2.1 alata*

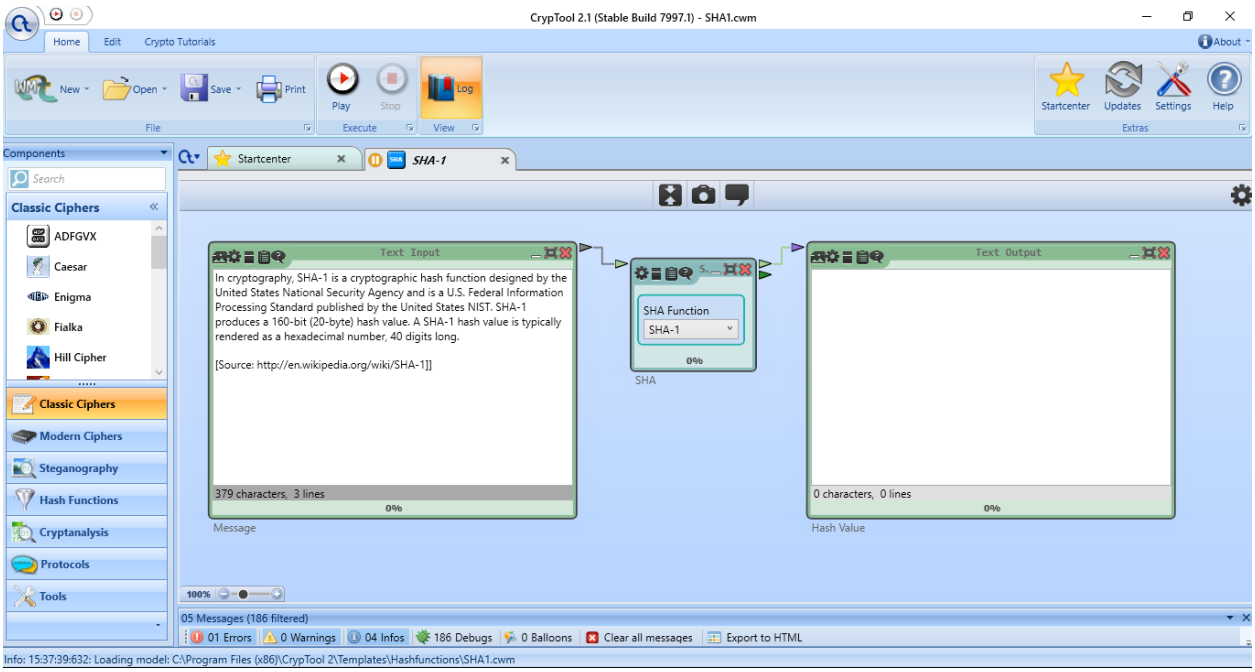
4.2.1. Četvrti primjer - Rad sa *hash* funkcijama

Željene *hash* funkcije je potrebno pronaći u odjeljku *Templates* → *Hash Functions* i potom odabrati željenu *hash* funkciju, slika 4.14.



Sl. 4.14. Odabir željene *hash* funkcije

Nakon odabira željene *hash* funkcije otvoriti će se nova kartica prikazana na slici 4.15. U primjeru navedenom u ovom radu korištena je SHA-1 funkcija.



Sl. 4.15. Blok shema SHA-1 *hash* funkcije

Blok shema SHA-1 funkcije sastoji se od tri bloka: *Text Input* (ulazna poruka), *SHA* (željena SHA *hash* funkcija) i *Text Output* (*hash* vrijednost). U blok *Text Input* potrebno je unijeti proizvoljnu ulaznu poruku, a u ovom primjeru će biti korištena sljedeća poruka:

Dobar dan!

Blok *SHA* omogućuje korisniku odabir željene SHA funkcije. Nakon što je unesena ulazna poruka te odabrana željena SHA funkcija, potrebno je pokrenuti simulaciju odabirom naredbe *Play* u programskoj traci alata, nakon čega će se u bloku *Text Input* pojaviti sljedeća *hash* vrijednost:

F6 6C 78 40 2C 5F 84 26 21 0D 84 18 9B 48 73 3A C7 21 DA FF

Nakon završetka simulacije potrebno je odabrati naredbu *Stop* koja se također nalazi u programskoj traci alata.

Navedeni postupak je potrebno ponoviti uz odabir ostalih SHA *hash* funkcija i zapisati dobivene *hash* vrijednosti:

- **SHA-256**: 5A 12 20 3F 09 B6 07 A5 53 F6 77 B9 C8 1C 33 63 29 ED BF 1D 5D A1 04 BA E3 52 87 D0 F2 4B B7 3A
- **SHA-384:** D1 90 B3 FC 6D 85 24 35 6A 9C 98 CC FB 6C 97 12 C2 86 24 B9 29 07 3E DB A8 69 74 ED 08 C4 66 9D 21 4B 90 A7 72 6A 03 0C 83 2B 77 8F E8 DF 88 A3
- **SHA-512**: 44 FD 00 0B 1A 51 86 14 30 6B D9 E3 3E 63 FE AE 7B 4C D7 BC E3 E2 E1 D3 12 12 8E 90 1F 4C 69 1A A9 DD 29 9E 40 9B 3A 1D 2B 9E 37 ED 33 8B 84 38 12 CB C3 43 FA DE E4 0B 5F 3B F8 09 55 3B F2 7A

Potrebno je obrisati prvu riječ u originalnom tekstu poruke te ponoviti postupak računanja *hash* vrijednosti.

Ulazna poruka:

dan!

Hash vrijednosti:

- **SHA-1**: 04 34 CA CD 76 2A AA AF C1 C8 E2 08 45 C2 14 2F 6C CC 1A 5A
- **SHA-256**: B9 62 81 7A 97 C6 13 B3 37 47 5E C0 03 D3 DE 7D 79 F3 63 99 1A 4D 75 6C 9F 1B 44 2D AE F6 B5 FA
- **SHA-384:** 85 B6 92 57 9D 9B 7C 6E 84 9C 5B 09 5D D3 D5 D9 38 DB 95 30 7A 36 72 FF 50 AE 34 FD 1F B5 D5 30 A2 AA 34 CE 75 5A 44 AB CD 19 57 EF A8 85 9F F8
- **SHA-512**: BC FB 80 A3 C8 56 CC 7C A5 4A D7 C3 29 AA 56 0F 36 DC 4F C8 72 90 1F 55 64 DB FC A1 0F A2 43 3B 43 84 3D 47 F7 AF 7B E8 F0 F5 40 6F AC AB 79 2D E6 1B EB C2 7D 99 4C 66 A8 C5 01 69 B0 F9 DA DE

Potrebno je promijeniti ulaznu poruku te ponoviti postupak računanja *hash* vrijednosti.

Ulazna poruka:

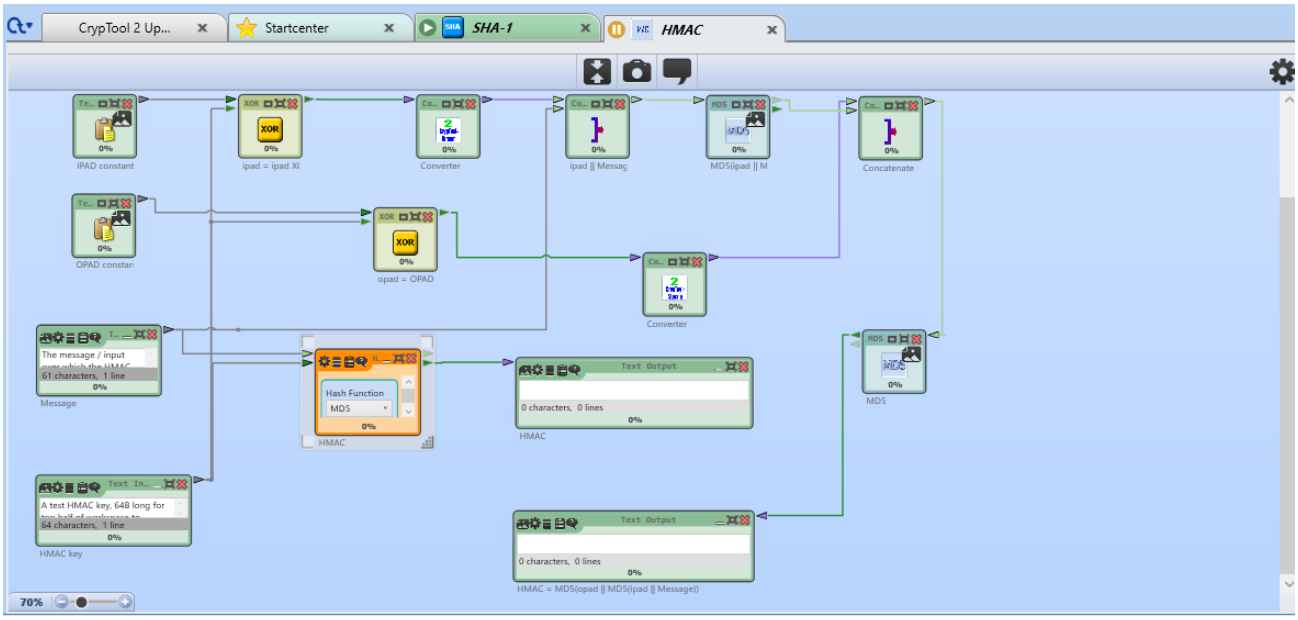
Ovaj tekst predstavlja proizvoljnu ulaznu poruku koja se koristi u svrhu prakticnih primjera hash funkcija u okviru diplomskog rada pod nazivom Primjena kriptografskih hash funkcija pri ocuvanju integriteta podataka.

Hash vrijednosti:

- **SHA-1**: 7E 29 96 DC 5D 1F 84 E1 63 D8 AB C1 F2 D6 8E 4F 11 33 A8 6B
- **SHA-256**: 87 64 1E 39 8C 31 FE 79 BB CA CC B9 77 DF F3 EA 4B 5B 69 2E A8 D7 7B 05 09 3C F6 F1 3A A4 69 CF
- **SHA-384:** 6F DA A4 91 1A 5A FC D2 94 66 71 A9 D5 13 C5 C3 09 E5 54 9F 4E DD 6F B2 03 F8 27 30 D4 E4 CC 7B A1 57 ED 7E 38 F6 09 AD A9 19 0C B8 5B C8 78 1F
- **SHA-512**: 72 83 0D 20 34 15 04 0E 86 A5 86 E1 7E CE 5B 1B 8B EF D3 12 B7 8D B6 1B AB 3E D5 D9 41 35 87 5E A6 FE F6 78 86 73 DF C4 E0 9A 3D E6 B3 50 B3 AA 94 7B E3 CB 40 40 A6 AB D3 89 D4 08 6A 09 DB 2D

4.2.2. Peti primjer - Rad sa HMAC kodovima

Potrebno je otvoriti karticu *Startcenter* prikazanu na slici 4.13, potom u odjeljku *Templates* odabrati naredbu *HMAC* nakon čega će se otvoriti nova kartica prikazana na slici 4.16.



Sl. 4.16. Blok shema HMAC algoritma

Donji dio blok sheme se sastoji od sljedećih blokova:

- *Message* predstavlja ulaznu poruku koja će se šifirati
- *HMAC key* predstavlja tajni ključ
- *HMAC* omogućuje odabir željene *hash* funkcije
- *HMAC output* predstavlja HMAC vrijednost

U blok *Message* je potrebno unijeti ulaznu poruku:

Ovaj tekst predstavlja proizvoljnu ulaznu poruku koja se koristi u svrhu prakticnih primjera hash funkcija u okviru diplomskog rada pod nazivom Primjena kriptografskih hash funkcija pri ocuvanju integriteta podataka.

U blok *HMAC key* potrebno je unijeti ključ: *diplomski rad*, te u bloku *HMAC* odabrati željenu *hash* funkciju. Pokrenuti simulaciju i zapisati rezultate.

HMAC vrijednosti:

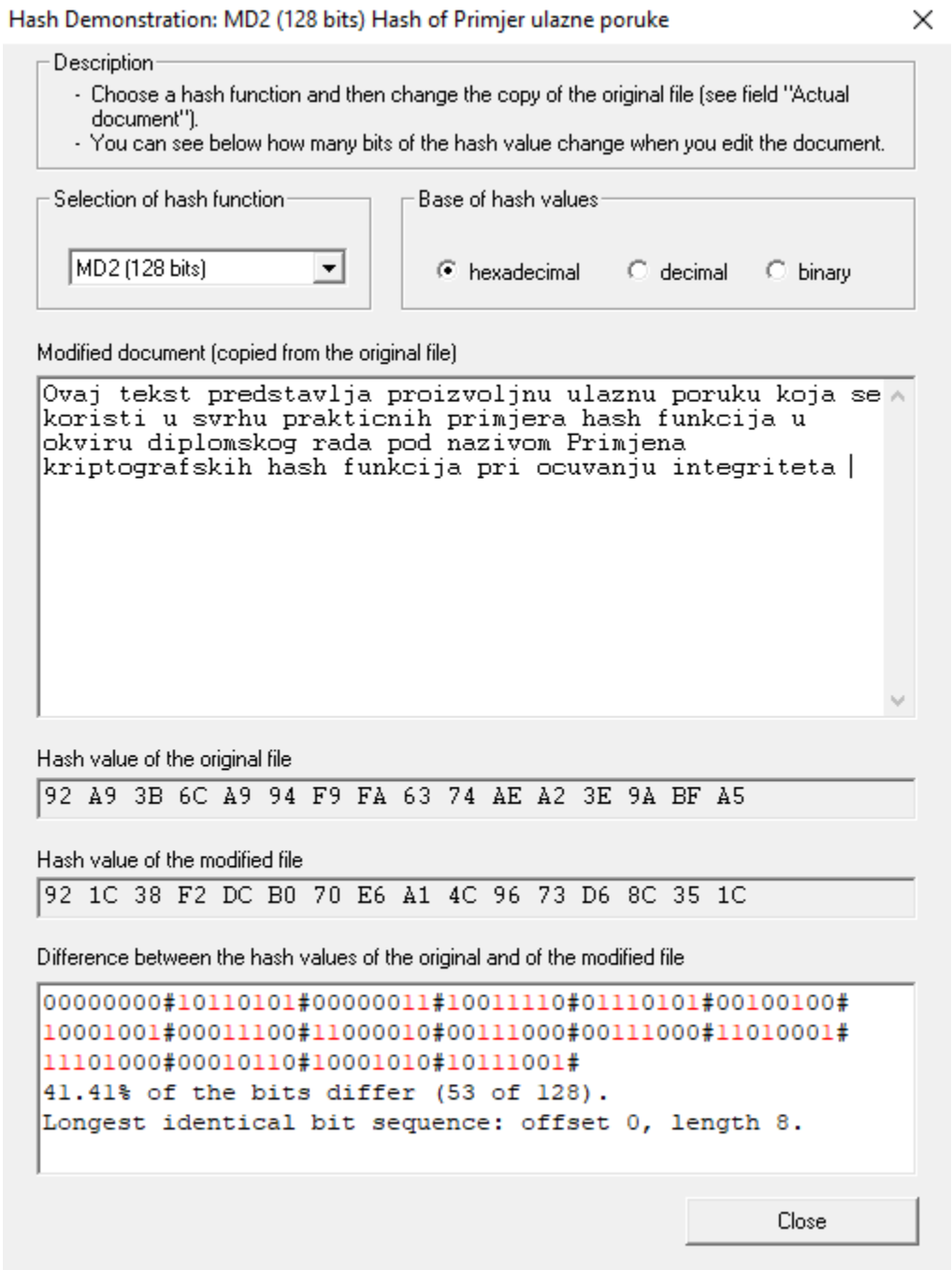
- **SHA-1**: 23 58 2C 7B EF EC 14 10 4E 63 E2 6B 08 6B 7A DC 4E 11 77 C7
- **SHA-256**: 57 C7 CD 64 C0 E9 63 27 1B 37 D9 FD 7E 24 86 F6 33 E2 07 22 90 43 DB A7 1F 44 25 82 A1 CF D2 41
- **SHA-384:** 52 31 B0 98 EA 13 E4 94 F2 37 F2 55 A4 F6 CA BF FB 40 AE 67 F6 AA A5 AC 92 97 A3 A6 69 CD F7 40 D5 9B 9F 36 EB 48 4E 45 33 98 95 2B F4 76 7F 38
- **SHA-512**: C0 DA 0E 25 53 86 D2 01 63 77 EE 40 51 D1 D1 A1 A3 CE 78 E2 4F 8E 79 31 91 C2 87 6D 3C 09 63 24 CD 05 0F E9 05 63 E9 AD A9 9D 5A 7A 17 16 1D 20 1E A0 75 D4 86 3D 7A 03 E9 FC 3A 3F 0E CB 39 49
- **MD5**: A5 6B 59 66 90 70 B6 46 FD 34 90 4B 29 11 DF 2A

5. ANALIZA REZULTATA

U prvom primjeru su se primjenjivale različite *hash* funkcije u programu *Cryptool 1.4.41* kako bi se šifrirao otvoreni tekst. Za istu ulaznu poruku različite *hash* funkcije daju *hash* vrijednosti koje se razlikuju u iznosu i broju bita pa tako MD2, MD4 i MD5 *hash* funkcije daju *hash* vrijednosti duljine 128 bita, SHA-1 *hash* funkcija daje *hash* vrijednost duljine 160 bita, SHA-256 *hash* funkcija daje *hash* vrijednost duljine 256 bita i SHA-512 *hash* funkcija daje *hash* vrijednost duljine 512 bita. Prilikom promjene ulazne poruke također je došlo i do promjene iznosa *hash* vrijednosti, a ta promjena se prikazuje u dijalogu programa što je prikazano na slici 5.1. Nakon promjene ulazne poruke, bitovi *hash* vrijednosti su se mijenjali ovisno o korištenoj *hash* funkciji:

- pri korištenju MD2 funkcije: 53 od 128 bitova se promijenilo, odnosno 41.41%
- pri korištenju MD4 funkcije: 59 od 128 bitova se promijenilo, odnosno 46.09%
- pri korištenju MD5 funkcije: 62 od 128 bitova se promijenilo, odnosno 48.44%
- pri korištenju SHA-1 funkcije: 77 od 160 bitova se promijenilo, odnosno 48.13%
- pri korištenju SHA-256 funkcije: 114 od 256 bitova se promijenilo, odnosno 44.53%
- pri korištenju SHA-512 funkcije: 264 od 512 bitova se promijenilo, odnosno 51.56%

Dobra *hash* funkcija će prilikom minimalne promjene ulazne poruke značajno promijeniti *hash* vrijednost.



Sl. 5.1. Promjena *hash* vrijednosti

U četvrtom primjeru u programu *Cryptool 2.1* također su se koristile različite *hash* funkcije kako bi se šifrirao otvoreni tekst. U primjeru su korištene različite SHA funkcije koje su primjenjivane na ulazne poruke različitih duljina. Promatrajući dobivene *hash* vrijednosti može se utvrditi da

promjenom ulazne poruke dolazi i do promjene *hash* vrijednosti. Bez obzira radi li se o kratkoj ili dugoj ulaznoj poruci i je li u pitanju minimalna promjena ulazne poruke ili se ulazna poruka značajnije promijenila, bitovi *hash* vrijednosti su se podjednako značajno mijenjali.

U drugom primjeru se prikazao način šifriranja otvorenog teksta korištenjem HMAC koda u programu *Cryptool 1.4.41*. Korisnik ima mogućnost odabrati jednu od 5 HMAC varijanti koje predstavljaju različite kombinacije ključa i ulazne poruke koje će se šifrirati. Odabirom različitih HMAC varijanti uz isti ključ i istu ulaznu poruku kao rezultat se dobiju različite *hash* vrijednosti, koje se ovisno o odabranoj *hash* funkciji razlikuju po iznosu i duljini bita. Odabirom različitih HMAC varijanti uz isti ključ, ulaznu poruku i *hash* funkciju, kao rezultat dobiju se *hash* vrijednosti koje se razlikuju po iznosu, ali imaju isti broj bita.

Peti primjer je također prikazao primjere šifriranja otvorenog teksta korištenjem HMAC koda. U programu *Cryptool 2.1* prikazana je blok shema HMAC algoritma, vidljiva na slici 4.16. Gornja polovica ove blok sheme sastoji se od postojećih komponenti za ručno računanje MD5-HMAC koda. Ove komponente su označene kako bi pokazale koji dio konačnog HMAC izlaza generiraju. Da bi ovakav raspored generirao točan rezultat, odabrani HMAC ključ mora imati duljinu točno 64 bajta. U donjoj polovici blok sheme nalazi se samostalna HMAC komponenta koja se može koristiti za provjeru rezultata HMAC koji koristi MD5 *hash* funkcije s ključem duljine 64 bajta i za izračunavanje HMAC kodova koji koriste različite *hash* algoritme i ključeve različite duljine. Korisnik odabire željenu *hash* funkciju kojom će šifrirati ključ i ulaznu poruku, a program potom računa HMAC vrijednost. Promatrajući dobivene HMAC vrijednosti može se zaključiti da prilikom minimalne promjene ulazne poruke dolazi do značajne promjene HMAC vrijednosti. Korištenjem HMAC kodova osiguran je integritet poruke i autentičnost korisnika.

U trećem primjeru je prikazan postupak kreiranja digitalnog potpisa u programu *Cryptool 1.4.41*. Korisnik prvo odabire *hash* funkciju kojom će se šifrirati ulazna poruka te dobiti *hash* vrijednost. Korisnik potom odabire 2 velika prosta broja tako što definira algoritam za generiranje prostih brojeva i raspon prostih brojeva. Nakon generiranja prostih brojeva, program računa privatni i javni ključ. Svrha privatnog ključ je šifriranje prethodno izračunate *hash* vrijednosti. Korisnik potom odabire certifikat sa pripadajućim imenom i lozinkom te kreira digitalni potpis. Procedura digitalnog potpisa je sljedeća:

Pošiljatelj koristi ulaznu poruku i privatni ključ kako bi generirao digitalni potpis poruke. Digitalni potpis ovisi o dokumentu koji se potpisuje, stoga su potpisi jednog sudionika sasvim različiti, osim ako su potpisani dokumenti potpuno jednaki. Čak i najmanja promjena u tekstu dokumenta generira drugačiju *hash* vrijednost koja će biti potpisana digitalnim potpisom. Dokument se zajedno sa digitalnim potpisom šalje primatelju, koji nakon primitka dokumenta može uz pomoć javnog ključa, dokumenta i digitalnog potpisa ustanoviti je li primljeni digitalni potpis točan. Primjenom *hash* funkcija se smanjuje nepotrebno povećanje podatkovnog prometa jer bi u suprotnom digitalni potpis bio dugačak koliko i sam dokument koji se šalje.

Korištenjem digitalnog potpisa osigurana je autentičnost korisnika, jer se može provjeriti dolazi li poruka od navedenog pošiljatelja, i integritet poruke tako što se može provjeriti je li došlo do promjene poruke.

6. ZAKLJUČAK

Ovaj rad bavi se problematikom očuvanja integriteta podataka kroz upotrebu kriptografskih *hash* funkcija. U današnje vrijeme kada svakodnevno raste potreba za internetskim uslugama, nužno se sve veća pozornost daje računalnoj sigurnosti i zaštiti integriteta podataka. Postoje razne opasnosti koje se mogu pojaviti prilikom prijenosa podataka, stoga je podatke potrebno zaštititi. Kriptografske *hash* funkcije predstavljaju moćan alat koji je u širokoj upotrebi u modernom računarstvu. *Hash* funkcije imaju široku primjenu u raznim sigurnosnim aplikacijama i internetskim protokolima. Pomoću *hash* funkcije vrši se šifriranje poruke, odnosno *hash* funkcija uzima poruku promjenjive duljine kao ulaz i stvara *hash* vrijednost fiksne duljine kao izlaz. U radu je objašnjena temeljna klasifikacija *hash* funkcija, odnosno podjela na *hash* funkcije sa ključem i *hash* funkcije bez ključa, i prikazana je modularna konstrukcija *hash* funkcija. Veća pozornost je usmjerena na SHA *hash* funkciju te MAC kodove koji su detaljnije analizirani i objašnjeni, a korištenjem programa *Cryptool* također je prikazana njihova primjena na stvarnim proizvoljnim primjerima. Radi usporedbe pojedinih *hash* funkcija i njihovih verzija, u pojedinim primjerima su korištene jednake ulazne poruke kako bi se što bolje mogla uočiti razlika između generiranih *hash* vrijednosti pojedinih *hash* funkcija. Ulazne poruke su također minimalno izmijenjene kako bi se uočile promjene iznosa *hash* vrijednosti i zabilježio postotak navedenih promjena *hash* vrijednosti. Izmjena ulaznih poruka je izvršena kako bi se dokazalo da pri minimalnim promjenama ulazne poruke dolazi do značajne promjene *hash* vrijednosti.

Razvoj računala i porast računalne moći omogućit će uspješnije napade na sadašnje poznate *hash* funkcije, no porastom napada na *hash* funkcije povećavaju se i napori usmjereni poboljšanju dizajna *hash* funkcija. Budućnost *hash* funkcija temelji se na sigurnim konstrukcijama *hash* funkcija, stoga je potrebno neprestano istraživati i razvijati *hash* funkcije te usavršavati njihovu konstrukciju kako bi se mogli zadovoljiti novi zahtjeve koji su postavljeni pred njih.

LITERATURA

[1] W., Stallings, „Cryptography and Network Security Principles and Practice”, Sixth Edition, Pearson, New Jersey, 2014.

[2] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone, „Handbook of Applied Cryptography”, CRC Press, Boca Raton, 1996.

[3] D. R. Stinson, „Cryptography Theory and Practice”, Third Edition, CRC Press, Boca Raton, 2006.

[4] B. Mennink, „Provable Security of Cryptographic Hash Functions”, doktorska disertacija, Faculty of Engineering Science, Arenbergkasteel, 2013.

[5] Cryptool 1, Dostupno na: https://www.cryptool.org/en/cryptool1

[6] Cryptool 2, Dostupno na: https://www.cryptool.org/en/cryptool2

[7] Cryptography Fundamentals, Part 3 – Hashing, Dostupno na: https://resources.infosecinstitute.com/cryptography-fundamentals-part-3-hashing/#gref

SAŽETAK

Diplomski rad fokusiran je na računalnu sigurnost i očuvanje integriteta informacijskog sustava. U svrhu očuvanja integriteta podataka u radu je prikazana upotreba jednostavnih *hash* funkcija. Primjena navedenih *hash* funkcija realizirana je korištenjem programa *Cryptool* pomoću kojeg je prikazana primjena pojedinih *hash* funkcija na proizvoljnim primjerima ulaznih poruka. Također je prikazana konstrukcija digitalnog potpisa. Svi koraci odabira i postavljanja parametara pojedinih *hash* funkcija detaljno su objašnjeni i opisani uz pripadajuće slike. Primjena *hash* funkcija vrši se na ulaznim porukama proizvoljne duljine i uspoređuje se razlika izračunatih *hash* vrijednosti prilikom promjene ulaznih poruka. Dobivene *hash* vrijednosti su uspoređene i objašnjene uz pomoć pripadajućih parametara zajedno sa grafičkim prikazima rezultata.

Ključne riječi: integritet, računalna sigurnost, kriptografske *hash* funkcije, SHA, MAC

ABSTRACT

Master's thesis focuses on computer security and preserving the integrity of the information system. Use of simple hash functions is presented in thesis, in order to preserve the integrity of the data. The implementation of specified hash functions is realized using Cryptool program which enables to display the application of individual hash functions on the arbitrary examples of the input messages and also shows the construction of the digital signature. All steps of selecting and setting parameters of individual hash functions are explained in detail and described with the related images. The application of the hash functions is used on the input messages of arbitrary lengths and the difference of the calculated hash values as the input messages changes is compared. The resulting hash values were compared and explained using the related parameters along with the result images.

Key words: integrity, computer security, cryptographic hash functions, SHA, MAC

ŽIVOTOPIS

Mislav Marić rođen je 24.kolovoza 1995. godine u Starim Mikanovcima. Osnovnu školu završava 2010. godine u OŠ Josipa Kozarca u Vinkovcima. Po završetku osnovne škole upisuje se u Gimnaziju Matije Antuna Reljkovića u Vinkovcima, smjer Opća gimnazija te je 2014. završava. Po završetku srednje škole upisuje se na preddiplomski sveučilišni studij Elektrotehnike na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. 2017. godine završava preddiplomski sveučilišni studij te na istom fakultetu upisuje diplomski sveučilišni studij Elektrotehnika, smjer komunikacije i informatika.