HMAC KOD ZA PROVJERU AUTENTIČNOSTI PORUKE

Diklić, Goran

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Diplomski sveučilišni studij računarstva

Diplomski rad

HMAC KOD ZA PROVJERU AUTENTIČNOSTI PORUKE

Goran Diklić

0069082349

Rijeka, studeni 2022.

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Diplomski sveučilišni studij računarstva

Diplomski rad

HMAC KOD ZA PROVJERU AUTENTIČNOSTI PORUKE

Mentor: izv. prof. dr. sc. Jonatan Lerga

Goran Diklić

0069082349

Rijeka, studeni 2022.

Umjesto ove stranice umetnuti zadatak

za završni ili diplomski rad

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad uz upotrebu navedene literature i konzultacija mentora izv. prof. dr. sc. Jonatana Lerge.

Rijeka, studeni 2022. _____

Goran Diklić

Zahvala

Velika hvala mojoj obitelji koja je uvijek vjerovala u mene i moj uspjeh, čak kada i sam nisam vjerovao te im zahvaljujem na neizmjernoj ljubavi, podršci i strpljenju.

Hvala mentoru izv. prof. dr. sc. Jonatanu Lergi kojeg izrazito uvažavam i kao čovjeka i kao stručnjaka. Zahvaljujem Vam na ukazanom vremenu i Vašem strpljenju tijekom studiranja i izrade ovoga rada.

SADRŽAJ

1. UVOD

Kriptologija je znanstvena disciplina zadužena za proučavanje i definiranje tehnika za zaštitu informacija (šifriranje) i otkrivanje šifriranih informacija (dešifriranje).

Kriptografija je grana kriptologije, uz kriptoanalizu, koja razvija tehnike za dešifriranje poruka bez poznavanja ključa, govori o procesima slanja poruka (informacija) u obliku koji je razumljiv samo onima koji znaju taj oblik pročitati, to jest samo onima kojima su te poruke namijenjene. Kriptografija je prije modernoga doba bila sinonim za šifriranje pretvarajući informacije iz čitljivoga oblika u nečitljivi oblik. Intencija kriptografije je zaštita informacija prilikom prijenosa od pošiljatelja do primatelja koristeći pritom procese šifriranja i dešifriranja.

Najjednostavnija metoda koristi simetrični ili sustav tajnoga ključa. Riječ je o vrsti enkripcije koja koristi samo jedan tajni ključ za proces šifriranja i dešifriranja informacija. Nužno je da pošiljatelj i primatelj znaju tajni ključ koji se koristi za šifriranje i dešifriranje razmijenjenih poruka. Jedan od problema koji se pritom javlja je razmjena ključa preko nesigurnog komunikacijskog kanala. Naime, ukoliko treća strana – osoba – otkrije tajni ključ tijekom razmjene, ona ima sve potrebno kako bi dešifrirala i pročitala poruku. Kriptolozi su osmislili asimetrični ili sustav javnoga ključa sa ciljem sprječavanja ovoga problema.

Asimetrični sustav koristi dva različita ključa: jedan javni i jedan privatni. Javni ključ je dostupan svima, a privatni ključ se drži u tajnosti. Navedenim sustavom se sprječava zloupotrebu ključeva. Poruka koja se šifrira javnim ključem može se dešifrirati isključivo privatnim ključem, a poruku šifriranu privatnim ključem moguće je isključivo dešifrirati javnim ključem. Primjer upotrebe asimetričnog sustava je sljedeći: prilikom slanja pošiljatelj traži javni ključ svoga primatelja, šifrira poruku i zatim je pošalje. U trenutku kada poruka stigne, samo je primatelj može dekodirati svojim privatnim ključem, čime se postiže učinak da presretanje poruke nema koristi bez posjedovanja odgovarajućega privatnog ključa.

Moderna se kriptografija u velikoj mjeri temelji na matematičkoj teoriji i praksi računalnih znanosti. Kriptografski su algoritmi razvijeni na temelju pretpostavki o težini računanja zbog čega je bilo kome teško ugroziti sigurnost takvih algoritama u stvarnosti. Teorijska istraživanja i napredak te brža računalna tehnologija zahtijeva da se algoritmi kontinuirano procjenjuju i, pod uvjetom da je potrebno, prilagođavaju.

Šifriranje poruke je postupak kojim se korištenjem ključa radi transformacija podataka na način da se čitljiva poruka pretvori u nečitljiv oblik svim osobama koje ne posjeduju ključ s

namjerom sprječavanja neovlaštenog čitanja poruke. Dešifriranje poruke je postupak kojim se šifrirana poruka koristeći ključ pretvori nazad u izvorni (čitljiv oblik).

Provjera integriteta i vjerodostojnosti informacija osnovna je potreba u računalnim sustavima i mrežama. Jedna je od metoda za osiguranje integriteta i autentičnosti poruke korištenje MAC-a. Riječ je o kodu za provjeru autentičnosti poruke (MAC – Message authentication code) koji predstavlja *oznaku* priloženu poruci s namjerom osiguranja integriteta i autentičnosti poruke, a on se dobiva primjenom MAC algoritma u kombinaciji s tajnim ključem.

Konkretnije: dvije strane koje komuniciraju preko nesigurnoga kanala zahtijevaju metodu kojom se informacije koje je poslala jedna strana mogu potvrditi kao autentične (ili nepromijenjene) od druge strane. U većini se slučajeva taj mehanizam zasniva na tajnom ključu koji dijele obje strane i ima oblik koda za provjeru autentičnosti poruke. Kod za provjeru autentičnosti se u terminologiji često zamijenjuje sa drugim terminima kao što su „vrijednost provjere integriteta“ ili „kriptografski kontrolni zbroj“.

Došlo je do porasta interesa za ideju konstruiranja MAC-ova iz kriptografskih *hash* funkcija. To se osobito vidi u internetskoj zajednici gdje je razvoj sigurnosnih protokola doveo do potrebe za jednostavnim, učinkovitim i široko dostupnim MAC mehanizmima.

Kodovi za provjeru autentičnosti poruke dijele sličnosti s kriptografskim *hash* funkcijama, međutim oni se bave različitim sigurnosnim zahtjevima. Osnovna je ideja MAC-a provjera autentičnosti izvora poruke i njezinoga integriteta. Za razliku od *hash-a* koji nastane korištenjem kriptografskih *hash* funkcija, MAC može generirati jedino primatelj koji ima pristup tajnom ključu.

Hash funkcije nisu izvorno dizajnirane za korištenje prilikom provjere autentičnosti poruke. Zato se mora obratiti posebna pažnja na njihovu upotrebu u tu svrhu. Konkretno, nedostaje im čvrsta i realistična sigurnosna analiza. Stoga postoji potreba za konstrukcijama koje održavaju učinkovitost *hash* funkcija, ali su podržane rigoroznijom analizom njihove sigurnosti.

Kod za provjeru autentičnosti poruke s ključem je proširenje MAC funkcije koja uključuje kriptografsku *hash* funkciju i tajni ključ u dobivanju koda za provjeru autentičnosti poruke. Za izračunavanje HMAC koda za provjeru autentičnosti poruke obično se koriste MD5 i SHA-1 kriptografske funkcije. Vrsta kriptografske *hash* funkcije koja se koristi u razvijanju HMAC-a se dodaje kako bi se označio sam algoritam (npr. HMAC-MD5, HMAC-SHA1...).

2. *HASH* FUNKCIJE

2.1. Pregled hash funkcija

Hash funkcija je matematička funkcija koja pretvara ulaznu vrijednost u drugu komprimiranu vrijednost. Osnovna podjela *hash* funkcija može se napraviti na dva razreda. Prvi razred bi predstavljao *hash* funkcije bez ključa koje na ulazu koriste samo poruku, a drugi razred bi predstavljale *hash* funkcije s ključem koje na ulazu koriste poruku i tajni ključ. *Hash* funkcije za ulaz koriste podatke proizvoljne duljine, a sam izlaz je uvijek fiksne duljine. Vrijednosti koje se dobiju korištenjem *hash* funkcija nazivaju se sažetkom poruke ili *hash* vrijednošću.

Osnovna su dva svojstva koja treba zadovoljiti dobra *hash* funkcija ta da funkcija treba biti vrlo brza za računanje te treba spriječiti dupliciranje vrijednosti koje se dobiju na izlazu. Sprječavanje dupliciranih vrijednosti još je poznato i pod terminom otpornosti na kolizije. Svojstvo otpornosti na kolizije predstavlja poteškoću pronalaska dvije različite ulazne vrijednosti u *hash* funkciji kojima bi se dobila ista izlazna hash vrijednost.

2.2. Osnovna svojstva kriptografskih *hash* funkcija

Kriptografske *hash* funkcije mapiraju nizove različitih duljina na kratke izlaze fiksne veličine. Ove su funkcije, npr. MD5 ili SHA-1, prvenstveno dizajnirane kako bi bile otporne na kolizije. To znači da ako takvu *hash* funkciju predstavimo s F, onda bi trebalo biti neizvedivo da se pronađu dva niza x i x′ takva da je F(x) = F(x′). Zamijećuje se da ovaj kriptografski pojam ne uključuje nikakav tajni ključ. Doista, svojstvo otpornosti na koliziju obično je povezano s funkcijama bez ključa. Primarna je inspiracija za takve funkcije kombiniranje s digitalnim potpisima na način koji te potpise čini učinkovitijima. Za tu je primjenu potrebno da funkcija bude javno izračunljiva i ne uključuje tajni ključ [1].

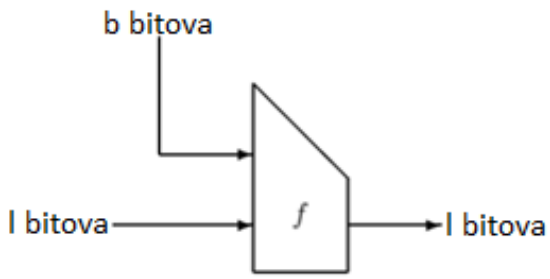
Uz osnovno svojstvo otpornosti na koliziju, kriptografske su *hash* funkcije obično dizajnirane tako da imaju neka svojstva slučajnosti, kao što su svojstva miješanja, neovisnost ulaza/izlaza, nepredvidljivost izlaza kada su dijelovi ulaza nepoznati itd. Navedena svojstva osim

što pomažu u otežavanju pronalaženja kolizija, također pomažu u nasumičnom odabiru ulaza za algoritam potpisivanja i šifriranja podataka (npr. RSA).

Kombinacija ovih svojstava koja se pripisuju kriptografskim *hash* funkcijama čini ih tako privlačnim za mnoge namjene izvan izvornoga dizajna kao funkcije otporne na kolizije. Ove su funkcije predložene kao osnova za pseudoslučajno generiranje, blok šifre, slučajnu transformaciju i kodove za provjeru autentičnosti poruka.

2.3. Iterirane konstrukcije

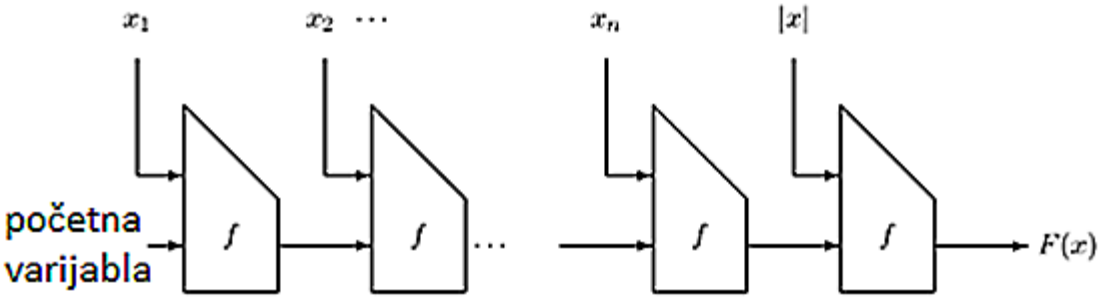
Merkle [2] (a kasnije Damgärd [3]) predložio je posebnu metodologiju za konstrukciju *hash* funkcija otpornih na kolizije. Ova metodologija čini osnovu za dizajn najčešćih kriptografskih *hash* funkcija poput MD5 i SHA-1. Zasniva se na osnovnoj značajki koja se zove funkcija kompresije koja obrađuje kratke ulaze fiksne duljine, a zatim se ponavlja na određeni način kako bi se „hashirali“ proizvoljno dugi ulazi. Takva funkcija kompresije, koju označavamo s f, prihvaća dva ulaza: lančanu varijablu duljine l i blok podataka duljine b. Za MD5 i SHA-1 imamo $b = 512$, dok je za MD5 $l = 128$, a za SHA-1 $l = 160$. Prikaz funkcije kompresije dan je na slici 2.1.



Slika 2.1. Funkcija kompresije (b = 512, l = 128 kod MD5)

Simbol f koristit će se za označavanje funkcije kompresije, a F za označavanje pridruženog iteriranoga *hash-a* za koji se pretpostavlja da uključuje standardni način dopunjavanja ulaza veličine do višekratnika b bitova.

Rad iterirane *hash* funkcije se odvija na sljedeći način. Termin početne varijable ili inicijalne vrijednosti se u nastavku objašnjenja koristi pod kraticom IV. Prvo, vrijednost *b*-bita početne varijable je fiksna ovisno o *hash* funkciji. Zatim se unos raspršuje ponavljanjem funkcije kompresije. To jest, ako je $x = x_1, x_2, ..., x_n$ ulaz, gdje su x_i blokovi svaki duljine *b*, a *n* proizvoljan broj blokova tada je $x_{n+1} = |x|$ duljina poruke. Vrijednost iterirane funkcije F koja se primjenjuje na *x* je h_{n+1} gdje je $h_0 = IV$ i $h_i = f(h_{i-1}, x_i)$ za $i = 1, 2, ..., n + 1$. Primjer iterirane konstrukcije je predočen na slici 2.2.



Slika 2.2. Iterirana konstrukcija hash funkcije s obzirom na kompresijsku funkciju f. Ulaz |x| u posljednjoj iteraciji ilustrira dodavanje duljine poruke kao u MD5 i SHA-1.

2.4. *Hash* funkcije s ključem

Nužno je prisjetiti se da je osnovni zadatak izgraditi sigurne funkcije provjere autentičnosti poruke iz kriptografskih *hash* funkcija (posebno iz iteriranih *hash* funkcija). Prva je očigledna prepreka činjenica da su tajni ključevi bitan element funkcije provjere autentičnosti poruke, a većina kriptografskih *hash* funkcija, a posebno funkcije poput MD5 ili SHA, uopće ne koriste ključeve. Iz toga se razloga prvo mora definirati način korištenja kriptografskih *hash* funkcija u sprezi s ključem.

Najčešći je pristup ključu *hash* funkcije [4] unos ključa kao dijela podataka koje je raspršila funkcija, npr. raspršivanje podataka *x* pomoću ključa *k* izvodi se primjenom *hash* funkcije F na konkatenaciju *k* i *x*. Pristup je da ključ bude dio početne varijable funkcije. Naime, umjesto korištenja fiksne i poznate početne varijable, kako je definirano izvornom funkcijom, ona je zamijenjena slučajnom i tajnom vrijednošću poznatom samo stranama.

Koristeći pristup „početne varijable s ključem“ možemo definirati *hash* funkcije s ključem kao obitelj funkcija. Neka je $f_k(x)$ definirana s $f_k(x) = f(k, x)$ funkcija kompresije s ključem, gdje je $|k| = l$ i $|x| = b$. Zatim je moguće pridružiti bilo kojoj iteriranoj *hash* konstrukciji (npr. MD5, SHA-1) obitelj funkcija (s ključem) $\{F_k\}_k$. Naime, za $x = x_1, ..., x_n$ treba definirati $F_k(x)$ kao k_{n+1} gdje je $k_i = f_{ki-1}(x_i)$ za $i = 1, ..., n + 1$, $k_0 = k$ i $x_{n+1} = |x|$. Da se razaznati da je prostor ključeva isti za funkcije kompresije s ključem i za iterirane *hash* funkcije s ključem – to je skup svih nizova duljine l. Izvorna iterirana *hash* funkcija dobiva se kao određeni član obitelji s ključevima.

Kao što je prije navedeno, pojam otpornosti na kolizije tradicionalno je pridružen javnim funkcijama (bez ključa). Ovdje je proširen pojam na (tajne) *hash* funkcije s ključem.

Kažemo da je obitelj *hash* funkcija s ključem $F_k(\varepsilon, t, q, L)$ slabo otporna na koliziju ako je bilo koji protivnik kojemu nije dat ključ k, ograničen da provede ukupno vrijeme t te pritom pogleda vrijednosti funkcije F_k izračunate na q poruka $m_1, m_2, ..., m_q$ po svom izboru, svaka duljine najviše L, ne može pronaći poruke m i m' za koje je $F_k(m) = F_k(m')$ s vjerojatnošću boljom od ε.

Iz gornjega se zahtjeva primijeti da je slabiji od tradicionalnoga zahtjeva otpornosti na koliziju iz *hash* funkcija (bez ključeva). U drugom je slučaju dovoljno pronaći kolizije za poznatu i fiksnu početnu varijablu. Također, u slučaju *hash* funkcija s tajnim ključem, treća strana treba svoje „primjere“ (poruke $m_1, m_2, ..., m_q$ u gornjoj definiciji) dobiti od samoga korisnika koji poznaje ključ k. U slučaju bez ključa napadač može raditi na pronalaženju kolizija koristeći algoritme grube sile. Algoritmi grube sile predstavljaju tehniku koja se koristi za rješavanje određenoga problema. Pod pojmom određenoga problema možemo smatrati otkrivanje npr. pina bankovne kartice jer se sastoji od četiri znamenke (0 – 9). Algoritam grube sile radi na tom principu da za svaku znamenku pina isproba sve moguće kombinacije (10^4 kombinacija), dakle tehnika se zasniva na nabrajanju svih mogućih kandidata kao rješenja. Nakon toga se vrši provjera zadovoljava li određeni kandidat svojstva da bi bio rješenje zadanog problema.

Štoviše, čak i napadi pretraživanja kolizija grube sile mogu postati izvedivi za funkcije poput MD5 zbog lakoće paralelizacije ovih napada [5]. Nasuprot tome, napadi na *hash* funkcije s tajnim ključem ne mogu se paralelizirati jer zahtijevaju interakciju s korisnikom.

Napadi koji pronalaze kolizije na *hash* funkciji s nasumičnim i poznatim početnim varijablama (takvi napadi su poznati za MD5 [6]) mogu se prilagoditi (putem napada proširenja)

da pronađu kolizije, čak ako je početna varijabla tajna. Takav bi napad kompromitirao slabu otpornost na koliziju *hash* funkcije. Zamijetno je da su u konstrukcijama napadi proširenja spriječeni dvostrukom primjenom *hash* funkcije.

2.5. Upotreba *hash* funkcija kao crne kutije

Kasnije predstavljene konstrukcije i analize ne ovise o posebnostima temeljne *hash* funkcije. Iskorištava se samo opća struktura funkcija kao što su MD5 i SHA-1 jer su razvijene na temelju osnovne funkcije kompresije koja radi na porukama fiksne duljine, a potom se ponavlja više puta kako bi se obrađivali ulazni podaci promjenjive duljine. Stoga se temeljna *hash* funkcija (ili odgovarajuća funkcija kompresije) može promatrati kao paket koji se može lako zamijeniti u slučaju da se pronađu ozbiljne slabosti u *hash* funkciji ili kada se pojave nove (efikasnije) dizajnirane *hash* funkcije.

Osim sigurnosne prednosti postoji praktična prednost MAC shema koje koriste temeljne *hash* funkcije kao „crnu kutiju“ (tj. primjenom *hash* funkcije ili funkcije kompresije, „kao što jest“, bez ikakvih izmjena). Naime, takve sheme dopuštaju neposrednu upotrebu postojećega i široko dostupnoga koda u knjižnicama koji implementira ove funkcije. Oni također dopuštaju korištenje hardverskih implementacija temeljne *hash* sheme. Konfiguracija ugniježđenog koda za provjeru autentičnosti koristi funkciju kompresije kao crnu kutiju; HMAC konfiguracija, što je još praktičnije, koristi samo pozive same iterirane *hash* funkcije.

3. KOD ZA PROVJERU AUTENTIČNOSTI PORUKE

Osiguravanje privatnosti je pojam koji se najčešće povezuje s kriptografijom. Pritom treba dati naglasak na provjeru autentičnosti poruka koja je važna jer nekada nije bitno da poruka koja se šalje ostane privatna zbog beznačajnog sadržaja, ali je značajno biti siguran tko je autor svake poruke.

Svojstvo u informacijskoj sigurnosti koje potvrđuje da poruka nije promijenjena tijekom prijenosa naziva se provjerom autentičnosti poruke ili autentifikacijom izvora podataka budući da provjerava autentičnost svake poruke. Tim svojstvom osigurava se cjelovitost podataka te se pruža mogućnost da strana koja prima poruku može provjeriti izvor poruke.

Provjera autentičnosti poruke obično se postiže korištenjem kodova za provjeru autentičnosti poruka (MAC), autentificiranom enkripcijom (AE) ili digitalnim potpisima. Kod za provjeru autentičnosti (MAC) poznat kao oznaka (*tag*) je u kriptografiji kratki dio informacija koji se koristi za provjeru autentičnosti poruke. Provjera autentičnosti poruke nudi mogućnost pošiljatelju da pošalje poruku primatelju tako da primatelj može detektirati promijene u poruci ukoliko se ista izmijeni na putu od pošiljatelja do primatelja. Autentifikacijom poruke se štiti integritet podataka poruke, čime se osigurava da svaka primljena poruka stigne u istom stanju u kojem se pošalje.

Kod za provjeru autentičnosti temelji se na korištenju kriptografskoga *hash-a* ili algoritma simetričnoga šifriranja. Ključ za provjeru autentičnosti dijele samo dvije strane, a sama autentifikacija neće uspjeti u postojanju treće strane koja posjeduje ključ jer algoritam više ne uspijeva otkriti krivotvorine jer nije moguće potvrditi jedinstveni izvor poruke. Osim toga, ključ se mora generirati nasumično kako bi se izbjegao njegov oporavak putem pretraživanja grubom silom (*brute-force*) i napada povezanih ključeva osmišljenih da ga identificiraju iz poruka koje prolaze kroz medij.

MAC funkcija uzima tajni ključ k (koji se dijeli između strana) i poruku m za vraćanje oznake $\mathrm{MAC}_k(m)$. Treća strana vidi niz (m_1, a_1), (m_2, a_2), ..., (m_q, a_q) parova poruka i njihovih odgovarajućih oznaka (tj. $a_i = \mathrm{MAC}_k(m_i)$) koje se prenose između strana.

Pojam koda integriteta poruke (MIC) često je zamijenjen pojmom MAC, posebno u komunikacijama [7], kako bi se razlikovao od upotrebe kao adrese za kontrolu pristupa medijima

(MAC adresa). Između ostaloga, neki autori [8] koriste termin MIC za pozivanje na sažetak poruke koji ima namjeru samo jedinstveno identificiranje jedne poruke.

Prema Request for Comments-u 4949 preporučuje se izbjegavanje korištenja izraza koda integriteta poruke (MIC) i umjesto toga preporučuje korištenje kontrolnoga zbroja, koda za otkrivanje pogrešaka, *hash-a*, raspršenog ključa, koda za provjeru autentičnosti poruke te zaštićenoga kontrolnog zbroja [9].

3.1. Terminologija

Sustav kodova za provjeru autentičnosti poruke sastoji od tri algoritma:

- algoritma za generiranje ključa – jednako nasumično odabire ključ iz prostora ključeva,
- algoritma potpisivanja – vraća oznaku s obzirom na ključ i poruku i
- algoritma za provjeru – provjerava autentičnost poruke s obzirom na ključ i oznaku, odnosno povratak je prihvaćen kada poruka i oznaka nisu neovlašteni ili krivotvoreni, a u suprotnom je povratak odbijen.

Sigurni kod za provjeru autentičnosti poruke mora odoljeti pokušajima protivnika da krivotvori oznake, za proizvoljne, odabrane ili za sve poruke, uključujući pod uvjetima poznate ili odabrane poruke. Bilo bi poželjno da je računski neizvedivo izračunati valjanu oznaku dane poruke bez znanja o ključu, unatoč tome ako i u najgorem slučaju pretpostavimo da protivnik zna oznaku bilo koje poruke osim one o kojoj je riječ.

Sustav koda za provjeru autentičnosti poruke (MAC) označuje algoritam koji se sastoji od tri algoritma (G, S, V) koji zadovoljavaju sljedeće:

- G (algoritam za generiranje ključa)
- S (algoritam za generiranje *oznake*)
- V (algoritam za verifikaciju)

3.2. Pretpostavka

Važnost korištenja kodova za provjeru autentičnosti poruka može se razmotriti na primjeru banke. Djelatnicima, a i samom sustavu banke, je ključno da sa sigurnošću zna tko mu šalje poruku/zahtjev. Ukoliko neželjeni korisnik može pristupiti bankovnom sustavu i pritom obaviti transakcije koje prividno izgledaju kao da ih je obavio autentificiran korisnik, sama sigurnost sustava je problematična.

U navedenom i sličnim primjerima očigledno je postojanje rizika gdje protivnik stvara poruke koje poprilično izgledaju kao da ih šalje konkretni pošiljatelj. Protivnik šalje poruku, imitirajući identitet pošiljatelja, primatelju te pritom isti biva prevaren.

Stvarni pošiljatelj ne mora nužno biti fizička osoba kao što je u navednom primjeru korišteno radi lakše ilustracije, već se može raditi o multinacionalnim firmama ili drugim entitetima. Postoje razni načini da primatelj shvati da mu poruka dolazi od stvarnog pošiljatelja, a jedan od njih je korištenje identifikatora u poruci koji označava pošiljatelja.

Kako bi se izvršila provjera autentičnosti poruke m gdje se koristi ključ k, pravi pošiljatelj će primjeniti neki od algoritama za provjeru autentičnosti poruke na k i m, prilikom čega nastaje autentificirana poruka m´ te potom pošiljatelj šalje poruku m´ primatelju. Prilikom slanja postoji mogućnost da primatelj ne primi poruku jer protivnik nadzire komunikacijski kanal kojim se poruka šalje. Primatelj na poruku koju primi, označit će se sa $\overline{m}$, primjenjuje algoritam za provjeru autentičnosti poruke prilikom čega su moguća dva scenarija:

1. primitak izvorne poruke m
2. potvrda kojom se poruka m ne smatra autentičnom

3.3. Pouzdanost

Primarni zadatak koji se nastoji riješiti korištenjem kodova za provjeru autentičnosti je sposobnost uočavanja promjene poruke na putu od pošiljatelja do primatelja. Nastoji se onemogućiti treću stranu u slanju poruke koja bi primatelju izgledala kao da je šalje stvarni pošiljatelj.

Treća strana razbija MAC shemu ukoliko može pronaći poruku *m*, koja nije uključena među m_1, ..., m_q, zajedno s odgovarajućom oznakom za autentifikaciju $a = MAC_k(m)$. Vjerojatnost je uspjeha protivnika ona vjerojatnost da se probije MAC shema. Pritom se da zamijetiti da protivnik koji pronađe ključ svakako razbija shemu, ali shema se također može razbiti tako da se na neki način kombinira nekoliko poruka i odgovarajućih kontrolnih zbrojeva u novu poruku i njezin valjani kontrolni zbroj.

Niz (m_1, a_1), (m_2, a_2), ..., (m_q, a_q) može nastati na nekoliko načina. Najjednostavniji je poznati napad porukama u kojem legitimne strane biraju poruke, a treća strana ih, prisluškujući komunikacijski kanal, preuzima. U slučaju da treća strana može odabrati slijed poruka $m_1, ..., m_q$, tada se to naziva *napadom odabrane poruke*. Treba primijetiti da je MAC shema zaštićena od odabranih poruka jača od one koja je sigurna samo protiv „poznatih poruka“. Ovdje se treba osvrnuti na napade na odabrane poruke, osim ako je drugačije navedeno. Odabrane poruke smatrat će se „upitima“ koje je odabrao protivnik i na koje je strana, ili strane, koje posjeduju ključ MAC-a odgovorila. Valja uočiti da protivnik može odabrati poruku m_i kao funkciju prethodno viđenih poruka i odgovarajućih oznaka za provjeru autentičnosti.

Prateći [10], sigurnost se kvantificira u smislu vjerojatnosti uspjeha koja se može postići kao funkcija broja q valjanih MAC primjera koje je vidio protivnik i raspoloživoga vremena t. Treba imati na umu da je vjerojatnost uspjeha koja se može postići za dane t, q ovisi o parametrima MAC sheme, posebno o duljini ključa. Tada vidimo da je MAC (ε, t, q, L) siguran MAC ako je bilo koji protivnik kojemu nije dat ključ k, ograničen da potroši ukupno vrijeme (broj operacija) t na napad i zatraži vrijednost funkcije MAC_k do q poruka $m_1, m_2, ..., m_q$ po svom izboru, svaki duljine najviše L, ne može prekinuti shemu osim s vjerojatnošću boljom od ε.

Kao konvencija u vremensko se ograničenje t uključuje vrijeme koje je potrebno za izračunavanje funkcije MAC_k u svakom od traženih upita. Ovo realnije prikazuje činjenicu da vrijeme potrebno za izračunavanje MAC-ova koji tvore odgovore na ove upite (osobito kada je broj q njih velik) može činiti značajan dio vremenske složenosti napada. U MAC oznaku se također uključuje veličina koda protivnikovoga algoritma. Može se zamisliti treća strana koja je unaprijed izračunala puno informacija i stavila ih u svoj kod. Primijetimo da je u ranijem odlomku opisano da ne ograničavamo napadača na bilo kakve posebne napade ili kriptoanalitičke tehnike. Sve što protivnik može učiniti pod zadanim granicama resursa (vrijeme i upiti) obuhvaćeno je ovom definicijom.

Uočava se da se predstavljajući sheme provjere autentičnosti poruka u kontekstu provjere autentičnosti poruka za komunikaciju, iste tehnike mogu koristiti za provjeru autentičnosti

informacija koje su pohranjene u nesigurnom mediju i one mogu biti podložne zlonamjernim modifikacijama. Naposlijetku, daje se naglasak isključivo na MAC konstrukcije koje koriste kriptografsku *hash* funkciju kao svoju osnovu, no postoji opsežna literatura koja pokriva MAC sheme temeljene na različitim tehnikama.

Temeljni problem predstavlja izgradnja funkcije provjere autentičnosti poruke na način da se težina krivotvorenja provjerene poruke može povezati s kriptografskom snagom temeljne *hash* funkcije.

Neke sigurnosne snage moraju se pretpostaviti od *hash* funkcije, odnosno hash funkcija treba svojim svojstvima poboljšati sigurnost. S druge strane, pretpostavke ne bi smjele biti prejake s obzirom na to da nema dovoljno povjerenja u aktualne kandidate poput MD5 i SHA-1.

Konkretno, bilo bi moguće smisliti „dokazivo sigurne“ MAC-ove ako se pretpostavi da se *hash* funkcije ponašaju kao potpune slučajne funkcije, ali to je manje korisno. Osnovni je zadatak dizajnirati MAC-ove koji uključuju korištenje kriptografskih hash funkcija na jednostavan način, ali se njihova sigurnost može argumentirati na temelju razumnih sigurnosnih pretpostavki o temeljnoj *hash* funkciji.

Navedeni se ciljevi mogu postići i moguće je predstaviti relativno jednostavnu analizu sheme koja pokazuje da napadač može, uz isti napor (vrijeme i količinu prikupljenih informacija), razbiti temeljnu *hash* funkciju u jednom od sljedećih načina:

1. napadač pronalazi kolizije u *hash* funkciji čak i kada je početna varijabla nasumična i tajna, a *hash* vrijednost nije eksplicitno poznata,
2. napadač je u stanju krivotvoriti funkciju kompresije s tajnim ključem koja se promatra kao MAC funkcija primijenjena na poruke fiksne duljine i djelomično nepoznate.

Posljedično, postojanje takvih napada bilo bi u suprotnosti s nekim od osnovnih pretpostavki o kriptografskoj snazi ovih *hash* funkcija.

Uspjeh u prvome od gore navedenih napada znači uspjeh u pronalaženju kolizija čija je prevencija glavni cilj dizajna kriptografskih *hash* funkcija. Međutim, uspjeh u prvom napadu iznad je još teži od pronalaženja kolizija u *hash* funkciji jer je kolizije, kada je početna varijabla tajna i vrijednost *hash* nije eksplicitno poznata, daleko teže pronaći nego kolizije u običnoj (fiksna početna varijabla) *hash* funkciji. Konkretno, napadi kada je početna varijabla tajna zahtijevaju interakciju s legitimnim korisnikom funkcije i onemogućuju paralelizam

tradicionalnih rođendanskih napada. Rođendanski napadi su vrsta napada u kriptografiji koji pronalaze kolizije u svakoj *hash* funkciji. Stoga, čak i ako *hash* funkcija nije bez kolizije u tradicionalnom smislu, sheme mogu biti sigurne.

Uspjeh drugoga napada iznad bi implicirao da su svojstva slučajnosti *hash* funkcija vrlo loša i da su svi bitovi *hash* izlaza istovremeno predvidljivi (čak i s tajnom početnom varijablom i djelomično nepoznatim ulazom).

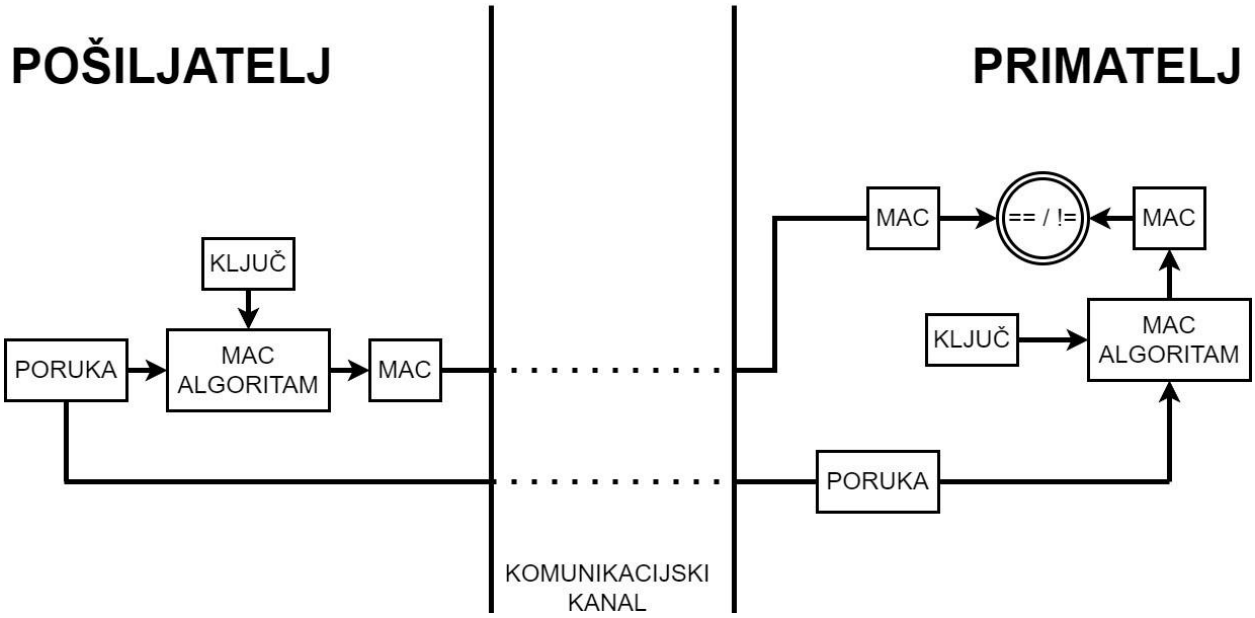
Unatoč tome što su MAC funkcije slične kriptografskim *hash* funkcijama, one imaju različite sigurnosne zahtjeve. Kako bi se mogla smatrati sigurnom, MAC funkcija mora biti otporna na egzistencijalno krivotvorenje pod napadima odabrane poruke. To označava da čak i u slučaju da napadač posjeduje tajni ključ i generira MAC-ove za poruke koje odabere, on ne može pogoditi MAC za druge poruke, odnosno za one koje nisu korištene za ispitivanje pretpostavke bez izvođenja neizvedivih količina računanja.

MAC se funkcije razlikuju od digitalnih potpisa zbog toga jer se MAC vrijednosti generiraju i provjeravaju koristeći se istim tajnim ključem. Time se sugerira da pošiljatelj i primatelj poruke moraju imati dogovor oko istog ključa prije samoga pokretanja komunikacije, kao što je i slučaj kod simetrične enkripcije.

Kao posljedica toga, MAC-ovi ne osiguravaju svojstvo neporicanja koje nude digitalni potpisi specijalno u slučaju dijeljenoga tajnog ključa na razini cijele mreže – svaki korisnik koji može provjeriti MAC, također je u mogućnosti generirati MAC-ove za druge poruke. Unatoč tome, digitalni se potpis generira korištenjem privatnoga ključa koji se nalazi u paru ključeva (javni i privatni), što je slučaj kod kriptografije s javnim ključem.

Budući da je spomenuti privatni ključ raspoloživ samo njegovom vlasniku, digitalni potpis dokazuje da je dokument potpisao nitko drugi nego taj vlasnik. Time se dokazuje da digitalni potpisi pružaju svojstvo neporicanja. Doduše, sustavi koji sigurno vežu informacije o korištenju ključa na MAC ključ mogu osigurati neporicanje – isti ključ posjeduju dvije osobe, ali jedna osoba ima kopiju ključa koja se može upotrijebiti za generiranje MAC-a, dok druga ima kopiju ključa u hardverskom sigurnosnom modulu koja dopušta samo provjeru MAC-a, no to se obično koristi u financijskoj industriji.

3.4. Primjer korištenja MAC-a



Slika 3.1. Prikaz funkcioniranja MAC-a

Na slici 3.1. vidljivo je kako pošiljatelj poruke aktivira MAC algoritam kako bi proizveo MAC podatkovnu oznaku. Poruka i MAC oznaka se nakon toga šalju primatelju. Primatelj zatim zauzvrat pokreće dio poruke u prijenosu kroz isti MAC algoritam koristeći isti ključ i isporučivši drugu MAC podatkovnu oznaku. Primatelj zatim radi usporedbu između prve, primljene MAC oznake u prijenosu, i drugom generiranom MAC oznakom. Ukoliko se utvrdi da su oznake identične, primatelj može sa sigurnošću zaključiti da poruka nije promijenjena ili neovlaštena tijekom prijenosa, čime je zagarantirana cjelovitost podataka. U slučaju da se oznake razlikuju dolazi se do potvrde da poruka nije autentična.

Usprkos tome, kako bi se omogućilo da primatelj može detektirati napade ponavljanja, potrebno je da sama poruka sadržava podatke kojima se osigurava da se ista poruka može poslati samo jednom (uzmimo za primjer vremenski žig, redni broj ili korištenje jednokratnoga MAC-a) jer bi inače napadač bio u mogućnosti, čak i bez razumijevanja sadržaja poruke, snimiti ju i reproducirati kasnije pritom proizvodeći isti rezultat kao izvorni pošiljatelj.

4. NESTED MESSAGE AUTHENTICATION CODE

U nastavku poglavlja slijedi predstavljanje osnovne konstrukcije NMAC-a i njegova analiza, a u narednom poglavlju se opisuje varijanta, HMAC, koja je dodatno usmjerena na praktične primjene. U korištenoj analizi sa f_k i F_k označene su dane funkcije kompresije i njezine iterirane funkcije.

4.1. Opis NMAC-a

Ugniježđeni kod za provjeru autentičnosti poruke, predstavljen je u radu [11] gdje je dokazano da se radi o sigurnom kodu za provjeru autentičnosti poruke, može se definirati na sljedeći način: uzmimo da su k_1 i k_2 dva neovisna i slučajna ključa koje koristi hash funkcija F putem iterirane konstrukcije funkcije kompresije f te se može zapisati u narednom obliku:

$$NMAC_k(x) = F_{k1}\big(F_{k2}(x)\big).$$

Ukoliko se tijekom implementacije NMAC-a koristi iterirana hash funkcija F kao unutarnja i vanjska funkcija, tada bi se ključ k_2 koristio kao početna vrijednost za unutarnju iteriranu hash funkciju s ključem, a pritom bi se, u vanjskoj iteriranoj hash funkciji s ključem koja se treba izvršiti samo jednom, ključ k_1 koristio kao početna vrijednost. Kada je riječ o ključevima, treba napomenuti da su duljine ključeva k_1 i k_2 iste te su duljine ključeva jednake duljini početne vrijednost hash funkcije F. Proučavajući implementaciju vidljivo je, s obzirom da se ključevi k_1 i k_2 koriste kao početna vrijednost za unutarnju i vanjsku funkciju, da ugniježđeni kod za provjeru autentičnosti poziva funkciju kompresije f hash funkcije F što je ranijem poglavlju o hash funkcijama objašnjeno kod upotrebe hash funkcija kao crne kutije.

Kako je ranije spomenuto, potrebno je da se samo jednom izvrši vanjska funkcija to jest funkcija kompresije vanjske hash funkcije pa se iz tog razloga hash funkcija može zamijeniti sa f - funkcijom kompresije s ključom, a gore napisana definicija NMAC-a može se zapisati kao:

$$NMAC_k(x) = f_{k1}\big(F_{k2}(x)\big).$$

U drugoj jednadžbi koristi se metoda dopune [12], koja se koristi za hash funkciju F, na način da zadnji blok x_n obuhvaća binarno kodiranu duljinu poruke. Izlaz koji nastaje primjenom funkcije F_{k2} također koristi istu metodu dopune kao i unutarnja funkcija.

4.2. Procjena pouzdanosti

Tijekom analize sigurnosti važno je da je sama analiza suvisla uzimajući u obzir treću stranu koja nastoji nadjačati NMAC s nekom vjerojatnošću, a pritom se može demonstrirati algoritam koji uz pomoć korištenja istih resursa narušava hash funkciju sa vjerojatnošću koja iznosi barem polovicu gore navedene. Takvim načinom se prezentira robusnost analize i istovremeno da je smanjenje sigurnosti temeljne hash funkcije minimalno.

Kod analize sigurnosti se znatno razmatraju prilagodljivi napadi, tj. napadi na odabrane poruke, budući da su isti najjači napadi od kojih se potrebno zaštiti. Nastavno na to, moguće je imati konkretnije definicije kojima se posebno procjenjuje broj odabranih i poznatih poruka koje su neophodne za napad, no to se izbjegava u analizama zbog jednostavnosti, a sama razlika je značajna u praksi.

Zahvaljujući korištenju dva različita sigurnosna ključa, k_1 i k_2, zaštićenosti funkcije doprinosi zasebno svaki pojednini ključ koji je duljine l, a ne kombinacija duljine oba ključa $2l$.

U radu [13] dana je preporuka da se koristi samo polovica bitova hash izlaza kao oznake za provjeru autentičnosti, a ta se preporuka temelji na konstataciji da se putem rođendanskih napada izvršavaju napadi krivotvorenja koji se daju izvesti sa složenošću $2^{l/2}$, a istodobno se korištenjem polovice bitova zahtijeva više odabranih poruka da bi napadi uspjeli. Time se zahtijeva da funkcija kompresije koja ima skraćeni izlaz tj. izlaz duljine $l/2$ predstavlja sigurnu oznaku za provjeru autentičnosti poruke. Kada se razmatra o tom segmentu, dolazi se do pitanja prihvatljivosti, a na isto se može odgovoriti u ovisnosi o svojstvima funkcije kompresije koja je u uporabi. Jasno je da korištenjem manje bitova treća strana ima manje bitova koje mora otkriti kod oznake za provjeru autentičnosti, no važno je također napomenuti da treća strana usput manje proučava izlaz funkcije kompresije promatrajući oznake za autentifikaciju poruka koje stvore pošiljatelj i primatelj.

Proučavajući rad [11] dolazi se do bilješke 4.9. prema kojoj se sigurnosna analiza NMAC-a može podijeliti u kategorije te tom podjelom može uvidjeti da je primjena vanjske funkcije NMAC-a korisna za sprječavanje napada proširenja.

Prva kategorija bi predstavljala slučaj kada je ključ k_1 tajan, a ključ k_2 nije te ga pritom zna treća strana. U tom slučaju je moguće pronaći kolizije na funkciji F_{k2} jer je kolizije lakše pronaći nego kada je ključ k_2 nasumičan i kad nije poznat trećoj strani.

Napad odabranom porukom je svojstvo MAC funkcije koje je moguće izvesti kada treća strana pronađe x i x´ tako da vrijedi $F_{k2}(x) = F_{k2}(x')$. Zamislimo treću stranu koja šalje poruku x kao ulaz u NMAC. NMAC zatim kreira MAC vrijednost te se ta vrijednost koristi za poruku x´ kao krivotvorena vrijednost. Nastavno na to, kada treća strana detektira koliziju unutarnje funkcije, ona može proširiti kolizije na porukama x i x´ u obliku $F_{k2}(x||s) = F_{k2}(x'||s)$, a pritom s označava proizvoljan tekst konkateniran sa porukama. Navedni napad se detaljnije opisuje kasnije u radu pri čemu se kao kratak zaključak može uzeti da se za postizanje sigurnog NMAC-a treba posjedovati funkcija F_{k2} koja je otporna na kolizije.

Druga kategorija predstavlja slučaj kada je ključ k_2 tajan, a ključ k_1 je poznat trećoj strani. U toj situaciji treća strana treba, kako bi mogla krivotvoriti MAC oznaku, izvršiti jednu od sljedećih mogućnosti. Prva mogućnost predstavlja opciju da treća strana mora pronaći inverz funkcije f_{k1} za poznati izlaz NMAC-a. Druga mogućnost je da treća strana pronađe kolizije za NMAC pošto sama treća strana ne može dobiti pravu vrijednost za $F_{k2}(x)$ nego samo vrijednost nakon primjene funkcije f_{k1}. Navedena vrijednost, $F_{k2}(x)$, može se tumačiti kao tajni ulaz koji ovisi o poruci koju koristi funkcija f_{k1}, pa bi iz tog razloga funkcija fk1 trebala biti jednosmjerna pošto bi trećoj strani, koristeći x i izlaz dobiven iz NMAC-a, trebalo biti teško otkriti $F_{k2}(x)$. U radu [14] pokazano je da slaba otpornost na kolizije funkcije F_{k2} sugerira na otpornost na kolizije funkcije f pretpostavivši da je f pseudoslučajna funkcija. Može se pretpostaviti da se vanjska funkcija kompresije f razlikuje od funkcije kompresije u unutarnjoj iteriranoj hash funkciji F te se prateći analizu danu u [11] doima prirodno da funkcija f_{k1} mora biti jednosmjerna pa i otporna na kolizije za nekoga tko poznaje ključ k_1.

Kada se razmotri teoretsko stajalište dano u prethodih nekoliko odlomaka dolazi se do zaključka da vanjska funkcija bez tajnosti nije u svim slučajevima dovoljno sigurna da onemogući napade proširenja.

Posljednja, treća kategorija predstavlja slučaj kada treća strana poznaje oba ključa, k_1 i k_2, te je evidentno da funkcije F_{k2} i f_{k1} moraju biti otporne na kolizije. Ovaj slučaj prikazuje $\frac{n}{2}$ bitnu

razinu sigurnosti spram napada proširenja gdje n predstavlja veličinu izlaza u bitovima. Slučaj je poznat pod pojmom dvostrukog raspršivanja koji se koristi za postizanje veće razine sigurnosti kod napada proširenja, a isti je opisan u [15].

5. HASH MESSAGE AUTHENTICATION CODE

Definicija i analiza konstrukcije HMAC-a prvi put je objavljena 1996. u radu Mihira Bellarea, Ran Canettija i Huga Krawczyka, a oni su također 1997. godine i napisali RFC 2104. Rad iz 1996. godine također je definirao ugniježđenu varijantu nazvanu NMAC. FIPS PUB 198 generalizira i standardizira upotrebu HMAC-ova. HMAC se koristi unutar IPsec, SSH, TLS protokola i za JSON web tokene.

HMAC, ponekad tumačen kao kod za provjeru autentičnosti poruke s ključem ili kao kod za provjeru autentičnosti poruke zasnovan na *hash-u*, je specifična vrsta koda za provjeru autentičnosti poruke (MAC) koji uključuje kriptografsku *hash* funkciju i tajni kriptografski ključ. HMAC se, kao i ranije spomenuti MAC, može istodobno koristiti za provjeru integriteta podataka i autentičnosti poruke.

HMAC nudi mogućnost autentifikacije korištenjem zajedničke tajne umjesto korištenja digitalnih potpisa s asimetričnom kriptografijom. On nadomješta potrebu za složenom infrastrukturom javnoga ključa delegirajući razmjenu ključeva na strane u komunikaciji, koje su odgovorne za uspostavljanje i korištenje pouzdanoga kanala za dogovor o ključu prije komunikacije.

Zbog široke dostupnosti besplatnoga koda biblioteka za postojeće *hash* funkcije (osobito MD5), praktična je prednost izgradnja MAC mehanizama koji koriste ove funkcije kao crnu kutiju, tako da se MAC može implementirati jednostavnim pozivanjem postojeće funkcije. Predstavljena NMAC konstrukcija zahtijeva izravan pristup kodu za funkciju kompresije (a ne za cjelokupnu *hash* funkciju). Takva je promjena trivijalna za funkcije s dobro strukturiranim kodom poput MD5 [16]. Međutim, u nekim bi se slučajevima ipak željelo izbjeći čak i one minimalne promjene i koristiti kod (ili hardversku implementaciju) kakav jest. Ovdje se predstavlja preinaka NMAC-a koja postiže ovaj cilj. Kao dodatnu prednost ova konstrukcija uključuje jedan dugi ključ k duljine l bita za razliku od dva različita ključa kao u NMAC-u. Ta značajka ima neke prednosti na razini upravljanja ključeva. Uz dodatnu se pretpostavku o temeljnoj funkciji kompresije može pokazati primjenjivost NMAC analize na HMAC.

5.1. HMAC funkcija

Neka je *F hash* funkcija. Radi jednostavnosti opisa može se pretpostaviti da je *hash* funkcija MD5 ili SHA-1; no konstrukcija i analiza mogu se primijeniti i na druge funkcije. *Hash* funkcija prima ulaze bilo koje duljine i proizvodi *l*-bitni izlaz (l = 128 za MD5 i l = 160 za SHA-1). Neka x označava podatke na koje treba primjeniti MAC funkciju i neka k bude tajni ključ provjere autentičnosti poruke koji dijele dvije strane. Ne smije biti veći od 64 bajta, veličine bloka raspršivanja, a ako je kraći, dodaju se nule kako bi njegova duljina bila točno 64 bajta. Nadalje definiramo dva fiksna i različita niza od 64 bajta *ipad* i *opad* kako slijedi (¨i¨ i ¨o¨ su mnemotehnike za unutranji i vanjski):

ipad = bajt 0x36 ponovljen 64 puta

opad = bajt 0x5C ponovljen 64 puta

Funkcija HMAC uzima ključ $\bar{k}$ i x te proizvodi:

$$\text{HMAC}_k(x) = F(\bar{k} \oplus \text{opad}, F(\bar{k} \oplus \text{ipad}, x)).$$

Proces se odvija u nekoliko koraka:

1. dodaju se nule na kraj $\bar{k}$ da bi se dobio 64 bajtni niz,
2. vrši se XOR operacija (bitovni operator isključivo ili) na nizu od 64 bajta izračunatom u koraku (1) s ipad-om,
3. dodaje se x nizu od 64 bajta koji je rezultat koraka (2),
4. primjenjuje se F na niz generiran u koraku (3),
5. vrši se XOR operacija (bitovni operator isključivo ili) na 64 bajtnom nizu izračunatom u koraku (1) s opad,
6. dodaje se rezultat F iz koraka (4) nizu od 64 bajta koji je rezultat koraka (5),
7. primijeni se F na niz generiran u koraku (6) i ispisuje rezultat.

Preporučena duljina ključa je najmanje l bita. Duži ključ ne doprinosi značajno sigurnosti funkcije, iako bi mogao biti preporučljiv ako se slučajnost ključa smatra slabom.

HMAC po izboru dopušta skraćivanje konačnoga izlaza na recimo 80 bita.

Kao rezultat dobivamo jednostavnu i učinkovitu konstrukciju. Ukupni trošak za autentifikaciju niza x je blizu troška raspršivanja toga niza, pogotov kako x postaje velik.

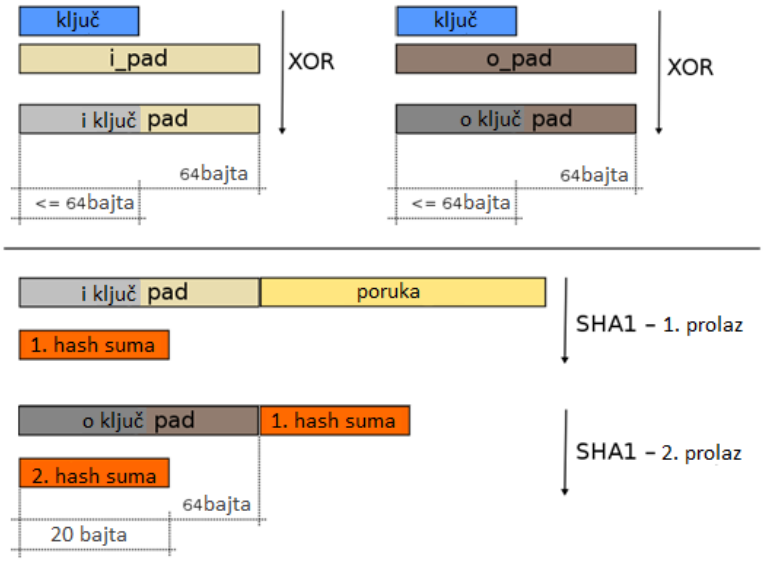
Napomena HMAC koristi *hash* funkciju *F* kao crnu kutiju. Za implementaciju HMAC-a nisu potrebne modifikacije koda za *F*. Ovo olakšava korištenje knjižničnoga koda za *F*, a također olakšava zamjenu određene *hash* funkcije, kao što je MD5, s drugom, kao što je SHA-1, ako se pojavi potreba za tim.

HMAC je nedavno izabran kao obvezan za implementaciju autentifikcijske transformacije za internetske sigurnosne protokole koje je dizajnirala IPSEC radna grupa IETF-a (zamjenjuje kao obaveznu transformaciju onu opisanu u [17]). U tu svrhu HMAC je opisan u internetskom nacrtu [18] i u nadolazećem RFC-u. I drugi internetski protokli prihvaćaju HMAC (npr. s-http [19], SSL [20]).

5.2. Postupak generiranja HMAC koda za provjeru autentičnosti poruke

Bilo koja kriptografska *hash* funkcija, uzimajući za primjer SHA-2 ili SHA-3, može se koristiti u izračunu HMAC-a: rezultirajući MAC algoritam naziva se HMAC-X, gdje je X korištena *hash* funkcija (npr. HMAC-SHA256, HMAC-SHA3-512...). Kriptografska snaga HMAC-a ovisi o kriptografskoj snazi temeljne *hash* funkcije.

HMAC koristi dva prolaza/koraka *hash* računanja. Tajni se ključ prvo koristi za dobijanje dva ključa – unutarnjega i vanjskoga. Prvi prolaz algoritma proizvodi interni *hash* izveden iz poruke i unutarnjega ključa. Drugi prolaz proizvodi konačni HMAC kod izveden iz unutarnjega *hash* rezultata i vanjskoga ključa. Stoga algoritam osigurava bolju otpornost na napade produženja duljine.



Slika 5.1. Primjer generiranja HMAC-SHA1 algoritma

Iterativna *hash* funkcija razbija poruku u blokove fiksne veličine i iterira ih pomoću funkcije kompresije. Za primjer uzmimo SHA-256 koja radi na 512-bitnim blokovima.

Hash funkcija -> H	b (bajtovi)	L (bajtovi)
MD5	64	16
SHA-1	64	20
SHA-224	64	28
SHA-256	64	32
SHA-512/224	128	28
SHA-384	128	48
SHA-512	128	64
SHA3-224	144	28
SHA3-256	136	32
SHA3-384	104	48
SHA3-512	72	64
izlaz = H (ulaz)		
L = duljina (izlaz)		
b = unutarnja duljina hash bloka		

Tablica 5.1. Prikaz hash funkcija te odgovarajućih duljina izlaza i unutarnjih duljina hash blokova

Veličina izlaza iz HMAC-a je ista kao i temeljne *hash* funkcije (npr. 256 i 512 bita u slučaju SHA-256 i SHA3-512 respektivno), iako se po želji može skratiti.

HMAC ne šifrira poruku te umjesto toga poruka (šifrirana ili ne) mora biti poslana uz HMAC *hash*. Stranke s tajnim ključem same će ponovno hashirati poruku, a ako je autentična, primljeni i izračunati *hashovi* će se podudarati.

U sljedećem će se dijelu ukratko objasniti neki od razloga korištenih u [11] za opravdanje HMAC konstrukcije.

Standardni pristup procjeni sigurnosti je traženje napada na MAC konstrukciju kandidata. Kada se mogu pronaći praktični napadi, njihov je učinak svakako konačan: konstrukcija se mora odbaciti. Problem je kada napadi nisu još pronađeni. U tom se slučaju postavlja pitanje treba li usvojiti konstrukciju, a na pitanje nije jasno dati odgovor jer bi se napadi mogli pronaći u budućnosti.

Pravilo kojim se vodila konstrukcija HMAC-a bilo je da *nepostajanje napada danas ne predstavlja sigurnost za budućnost*. Mora se naći bolji način da se opravda sigurnost konstrukcije prije nego što se ona usvoji.

Dobro opravdana konstrukcija MAC-a je ona prema kojoj se sigurnost MAC-a može što je moguće bliže povezati s (pretpostavljenim) sigurnosnim svojstvima temeljne *hash* funkcije.

Pretpostavke o sigurnosti *hash* funkcije ne bi trebale biti prejake, budući da ipak nije stečeno dovoljno povjerenja u trenutne kandidate (poput MD5 ili SHA). Zapravo, što su slabija pretpostavljena sigurnosna svojstva *hash* funkcije, to je rezultantna MAC konstrukcija jača.

Donose se pretpostavke koje odražavaju standardnije postojeće upotrebe *hash* funkcije. Svojstva koja se zahtijevaju uglavnom su izostanak kolizije i nešto ograničene „nepredvidivosti“. Ono što je prikazano je da ako funkcija *hash* funkcije ima ova svojstva, MAC je siguran; jedini način na koji bi MAC mogao zakazati je ako zakaže *hash* funkcija.

Ustvari, pretpostavke koje se stvaraju na mnogo su načina *slabije* od standardnih. Posebno se zahtijeva samo slab oblik otpornosti na kolizije. Stoga je moguće narušavanje svojstava *hash* funkcije *F* (npr. pronađu se kolizije), a HMAC temeljen na *F* ipak preživi.

5.3. Sigurnost HMAC-a

Kriptografska snaga HMAC-a ovisi o veličini tajnoga ključa koji se koristi. Najčešći napadi na HMAC-ove su napadi grube sile (*brute force*) koji se koriste za otkrivanje tajnoga ključa. HMAC-ovi su znatno manje pod utjecajem kolizija od njihovih temeljnih algoritama raspršivanja (hashiranja) [21, 22]. Detaljnije: Mihir Bellare je dokazao da je HMAC pseudoslučajna funkcija pod pretpostavkom da je funkcija kompresije pseudoslučajna [23]. Time se vraća jamstvo temeljeno na dokazu budući da ni jedan poznati napad ne ugrožava pseudoslučajnost funkcije kompresije, a također pomaže objasniti otpornost na napad koju je HMAC pokazao, čak i kada se implementira s *hash* funkcijama čija je (slaba) otpornost na kolizije ugrožena. HMAC-MD5 stoga ne pati od istih slabosti koje su pronađene kod MD5.

RFC 2104 zahtijeva da se „ključevi duži od b bajtova prvo hashiraju pomoću *hash* funkcije pod oznakom H“ čime se dolazi do zbunjujuće pseudokolizije: ako je ključ duži od veličine bloka hashiranja (npr. 64 bajta za SHA-1), tada se HMAC (*k*, *m*) izračuna kao HMAC (H(*k*), *m*). To se svojstvo ponekada navodi kao moguća slabost HMAC-a u scenarijima hashiranja lozinki: pokazano je da je moguće pronaći dugi niz ASCII i slučajnu vrijednost čija će *hash* vrijednost također biti ASCII niz, a obje će vrijednosti proizvesti isti HMAC izlaz [24].

Jongsung Kim, Alex Biryukov, Bart Preneel i Seokhie Hong su 2006. godine pokazali kako su razlikovali HMAC sa smanjenim verzijama MD5 i SHA-1 ili pune verzije HAVAL, MD4 i SHA-0 od nasumične funkcije ili HMAC sa slučajnom funkcijom. HMAC s punom verzijom MD4 može se krivotvoriti s tim znanjem. Ovi napadi nisu u suprotnosti sa sigurnosnim dokazom HMAC-a, ali pružaju uvid u HMAC na temelju postojećih kriptografskih *hash* funkcija [25].

Xiaoyun Wang [26] je 2009. godine predstavio prepoznatljiv napad na HMAC-MD5 bez korištenja povezanih ključeva. Može se razlikovati instanciranje HMAC-a s MD5 od instanciranja sa slučajnom funkcijom s 297 upita sa vjerojatnošću od 0.87.

Informativni RFC 6151 [27] je objavljen 2011. godine kako bi se sažela sigurnosna razmatranja u MD5 i HMAC-MD5. Za HMAC-MD5 u RFC-u je napravljen sažetak da se, usprkos tome što je sigurnost same *hash* funkcije MD5 ozbiljno ugrožena, trenutno čini da poznati „napadi na HMAC-MD5 ne predstavljaju praktičnu ranjivost kada se koriste kao kod za provjeru autentičnosti poruke“, međutim dodaje se da „za novi dizajn protokola ne bi trebao biti uključen paket šifriranja s HMAC-MD5“.

U svibnju 2011. godine objavljen je RFC 6234 s pojedinostima o apstraktnoj teoriji i izvornom kodu za HMAC-ove temeljene na SHA-u.

Sigurnost MAC-a znači sigurnost od krivotvorenja. MAC mehanizmu su narušena svojstva ako napadač, koji nema ključ $\bar{k}$, može pronaći neki tekstualni niz *x* zajedno s njegovom ispravnom MAC vrijednošću HMACk(*x*). Pretpostavlja se da je napadač u stanju prikupiti određeni broj primjera parova tekstova i njihovih valjanih MAC-ova promatrajući promet između pošiljatelja i primatelja.

Doista, protivniku je čak dopušten napad odabranom porukom kojim može utjecati na izbor poruke za koje pošiljatelj izračunava MAC-ove. Slijedeći [10, 28] kvantificira se sigurnost u smislu vjerojatnosti uspješnoga krivotvorenja pod takvim napadima.

Sigurnost HMAC-a temelji se na sigurnosti NMAC-a. Glavno zapažanje za povezivanje ove dvije funkcije i njihove sigurnosti je da definiranjem $k_1 = f(\bar{k} \oplus opad)$ i $k_2 = f(\bar{k} \oplus ipad)$, dobivamo da je $HMAC_k(x) = NMAC(k_1, k_2)(x)$. Drugim riječima, gornja transformacija na ključu čini HMAC posebnim slučajem NMAC-a, gdje su ključevi k_1 i k_2 'pseudoslučajno' izvedeni iz *k* pomoću funkcije kompresije *f*. Budući da analiza NMAC-a pretpostavlja da su k_1 i k_2 nasumični i neovisno odabrani ključevi, tada je za primjenu ove analize na HMAC potrebno pretpostaviti da napadač ne može razlikovati k_1 i k_2 izvedene pomoću *f* od istinski slučajnih ključeva.

Ovo predstavlja dodatnu pretpostavku o kvaliteti funkcije *f* (koja se unosi putem ulaza *k*) kao pseudoslučajne funkcije. Zahtijeva se relativno slab oblik pseudoslučajnosti budući da protivnik koji pokušava saznati o mogućim ovisnostima k_1 i k_2 ne može izravno vidjeti izlaz pseudoslučajne funkcije na bilo kojem ulazu. Ukratko, napadi koji rade na HMAC, a ne na NMAC su u principu mogući. Međutim, takav bi napad otkrio velike slabosti pseudoslučajnih svojstava temeljne *hash* funkcije.

Važno je napomenuti da se u praksi većina ključeva bira pseudoslučajno, a ne kao istinski slučajni nizovi; osobito je vjerojatno da će čak i ako netko koristi NMAC, implementacije izabrati da izvedu k_1 i k_2 koristeći pseudoslučajni generator. U slučaju HMAC-a takav pseudoslučajni generator je „ugrađen“ kroz definiciju funkcije korištenjem funkcije *f* i gore definiranih *pad-ova*. Ova je upotreba za pseudoslučajno generiranje funkcija poput MD5 ili SHA-1 vrlo česta u praktičnim implementacijama (dizajneri SHA-1 preporučili su korištenje ove funkcije za pseudoslučajno izvođenje različitih veličina u DSS standardu).

Gore su navedene određene vrijednosti za *opad* i *ipad* odabrane kako bi imale vrlo jednostavan prikaz (kako bi se pojednostavila specifikacija funkcije i minimizirala mogućnost

pogrešaka implementacije) te da bi se osigurala velika Hammingova udaljenost između *padova*. Upravo spomenuto je namijenjeno iskorištavanju svojstava miješanja koja se pripisuju funkciji kompresije koja leži u osnovi *hash* shema u upotrebi. Ova svojstva su važna kako bi se osigurala računalna neovisnost između dva izvedena ključa.

Konačno, napomena je da korištenje jednog ključa dugoga l bita za razliku od dva (nezavisna) ključa ne predstavlja slabljenje funkcije u odnosu na iscrpno pretraživanje ključa, jer čak i kada su odabrani neovisno, ključevi k_1 i k_2 mogu se pojedinačno pretraživati kroz napad podijeli pa vladaj.

5.4. Razmatranja o implementaciji HMAC-a

U ovome se dijelu rada ukazuje na neke probleme implementacije HMAC-a. Primijeti se da HMAC rezultira sporijom funkcijom od NMAC-a budući da prva zahtijeva dva dodatna izračunavanja funkcije kompresije (na blokovima ($\bar{k} \oplus$ opad) i ($\bar{k} \oplus$ ipad)). To može imati zanemariv učinak kod provjere autentičnosti dugih nizova podataka, ali može biti značajno za kratke podatke. Na sreću, implementacija može izbjeći ovo dodatno izračunavanje „spremanjem“ vrijednosti k_1 i k_2 odnosno, ove se vrijednosti izračunavaju samo jednom kada se ključ k generira ili dijeli prvi put, a zatim se pohranjuju kao stvarni ključevi funkcije NMAC. Za korištenje ovih zasebnih ključeva implementacija mora biti u mogućnosti inicijalizirati NMAC *hash* funkcije na te vrijednosti prije obrade podataka. Kao što je već ranije spomenuto, to je obično vrlo lako učiniti. Na taj način, HMAC služi onim implementacijama koje zahtijevaju korištenje iterirane *hash* funkcije bez modifikacija (tj. s fiksnom početnom varijablom), a u isto vrijeme ne kažnjava implementacije koje mogu ključati funkciju kroz početnu varijablu.

Može se uvidjeti da se može definirati funkcija HMAC tako da podržava ključeve promjenjive duljine. Međutim, manje od l bitova za ključ se ne preporučuje jer bi to oslabilo snagu ključne početne varijable (tj. k_1 i k_2). S druge strane, ključevi duži od l bita općenito neće pružiti dodatnu snagu budući da su izvedeni k_1 i k_2 ionako duljine l (ipak, dulji ključ k može pomoći, ovisno o svojstvima kompresije funkcija f i slučajnost ključa k, da ima jači pseudoslučajni učinak na generiranje k_1 i k_2).

Konačno, naglašava se da je, kao i u svakoj kriptografskoj implementaciji, sigurno upravljanje ključevima ključno za sigurnost funkcija poput onih koje su ovdje spomenute.

Posebno se savjetuje periodično osvježavanje ključeva. Čak i ako bi se pod trenutno poznatim napadima mogao koristiti isti ključ tijekom iznimno dugoga vremenskog razdoblja bez kriptoanalitičkoga kompromisa, implementacija bi trebala ograničiti vrijeme i količinu informacija obrađenih istim ključem.

5.5. Dizajn

Specifikacija dizajna HMAC-a inspirirana je postojanjem napada na trivijalnije mehanizme za kombiniranje ključa s *hash* funkcijom. Kao primjer bi se moglo pretpostaviti da bi se ista količina sigurnosti koju pruža HMAC mogla osigurati s MAC oznakom gdje hash funkcija kao ulaz koristi ključ konkateniran porukom. Međutim, ova metoda ima konkretan nedostatak: s većinom *hash* funkcija je lako dodati podatke u poruku bez poznavanja ključa i pritom dobiti još jedan važeći MAC („napad duljine proširenja“). Kao opcija, dodavanja ključa pomoću MAC-a = H (poruka || ključ), javlja se problem da napadač koji može pronaći koliziju u *hash* funkciji, ima koliziju u MAC-u (kao dvije poruke m_1 i m_2 koje daju isti *hash* će osigurati isti početni uvjet za *hash* funkciju prije nego što se dodani ključ hashira, zato će konačni *hash* biti isti). Korištenjem MAC = H (ključ || poruka || ključ) je bolje, međutim razni sigurnosni dokumenti sugeriraju ranjivost ovoga pristupa, čak i kada se koriste dva različita ključa.

Nisu pronađeni poznati napadi proširenja na trenutnu HMAC specifikaciju koja je definirana kao H (ključ || H (ključ || poruka)) jer vanjska primjena *hash* funkcije maskira međurezultat internoga *hasha*. Vrijednosti *ipad* i *opad* nisu kritične za sigurnost algoritma, ali su definirane na način da imaju veliku Hammingovu udaljenost jedna od druge, i tako će unutarnji i vanjski ključ imati manje zajedničkih bitova. Smanjenje sigurnosti HMAC-a zahtijeva da se razlikuju barem u jednom bitu.

Keccak [29] *hash* funkcija, koju je NIST odabrao kao pobjednika SHA-3 natjecanja, ne treba ovaj ugniježđeni pristup i može se koristiti za generiranje MAC-a jednostavnim dodavanjem ključa prije poruke, jer nije podložna napadima produženja duljine.

6. PRIJETNJE I USPOREDBA S DRUGIM PRIJEDLOZIMA

U ranijem je dijelu uspostavljen čvrst i opći odnos između sigurnosti funkcije NMAC i temeljne *hash* funkcije na način koji nije poznat za bilo koju drugu sličnu konstrukciju. Stoga se sa sigurnošću dolazi do zaključka da ako se koristi „dobra“ kriptografska *hash* funkcija, onda svi napadi na sheme neće biti praktični.

6.1. Rođendanski napadi

Rođendanski napadi su, kao što je već navedeno, tipovi napada u kriptografiji koji pronalaze kolizije u svakoj *hash* funkciji. Kao što je prikazano u [13, 30], rođendanski se napadi mogu primijeniti za napad i na MAC sheme s ključem temeljene na iteriranim funkcijama (uključujući CBC-MAC i druge sheme). Ovi se napadi odnose na većinu (ili sve) predloženih konstrukcija MAC-ova temeljenih na *hash-u*.

Konkretno, oni predstavljaju najpoznatije napade prilikom kojih se krivotvore konstrukcije HMAC i NMAC. Razmatranja ovih napada su važna jer snažno poboljšavaju naivne napade iscrpnoga pretraživanja. Međutim, njihova je praktična relevantnost u odnosu na ove funkcije zanemariva s obzirom na tipične duljine raspršivanja poput 128 ili 160, budući da ti napadi zahtijevaju poznavanje MAC vrijednosti (za dati ključ) na oko $2^{l/2}$ poruka (gdje je l duljina *hash* izlaza). Za vrijednosti $l \geq 128$ napad postaje potpuno neizvediv.

Za razliku od rođendanskoga napada na *hash* funkciju bez ključa, novi napadi zahtijevaju interakciju s vlasnikom ključa kako bi proizveli MAC vrijednost na velikom broju poruka, a zatim ne dopuštaju paralelizaciju. Na primjer, kada se koristi MD5, takav napad zahtijeva provjeru autentičnosti 2^{64} bloka (ili 2^{73} bita) podataka pomoću istoga ključa. Na komunikacijskoj vezi od 1 Gbit/sek bilo bi potrebno 250000 godina za obradu svih podataka potrebnih za takav napad. To je u oštroj suprotnosti s rođendanskim napadima na *hash* funkcije bez ključa koje omogućuju daleko učinkovitije i bliže realističnim napadima [5].

Uočava se da ovi napadi uzrokuju krivotvorenje MAC funkcije, ali ne i oporavak ključa. U [31] se, međutim, pokazuje da u nekim verzijama metode omotnice (slučaj kada se isti ključ koristi za dodavanje ispred i dodavanje iza, a ne izvodi se poravnavanje bloka dodanoga ključa), rođendanski napadi mogu biti dodatno poboljšani kako bi se osigurao potpuni oporavak ključa u

vremenu mnogo kraćem nego što to zahtijeva potpuna iscrpna pretraga. Budući da ti napadi zahtijevaju barem gore spomenutu složenost za krivotvorenje na temelju rođendanskih napada, ne mogu se smatrati praktičnim. Ipak, zanimljivo je primijetiti da se oni ne primjenjuju ni na jednu od konstrukcija, budući da ovdje nije primjenjivo pitanje poravnanja koje su iskorištavali ovi napadi.

Oblici rođendanskih napada koji se primjenjuju na konstrukcije mogu postati izvedivi samo ako se otkriju vrlo značajne slabosti u vjerojatnosti kolizije temeljne *hash* funkcije. Međutim, u takvom slučaju osnovna bi upotreba takve funkcije kao što je otpornost na koliziju (kako je izvorno zamišljena) bila jako ugrožena te bi funkciju trebalo odbaciti za kriptografsku upotrebu. Na kraju se napominje da se ovi rođendanski napadi (barem u njihovom jednostavnom obliku) mogu izbjeći randomiziranjem MAC konstrukcije prema poruci. Daljnje pojedinosti su opisane u [30].

6.2. Napadi kolizija na *hash* funkciju bez ključa

Razmotrimo konstrukciju „samo za dodavanje“: $MAC_k(x) = F(x, k)$. Pretpostavimo da su poznata dva niza x i x' za koje je $F(x) = F(x')$ (ova kolizija odgovara *hash* funkciji bez ključa). Tada se, bez obzira na ključ k u upotrebi, zna da je $MAC_k(x) = MAC_k(x')$ (zapravo, to vrijedi i za ekstenzije x i x'). Pronalaženje para kolizije x, x' za funkciju F je daleko lakše nego napadati NMAC kroz koliziju u F_k, gdje je k nepoznat, kao što je prikazano u gornjoj raspravi o rođendanskim napadima, dok je drugo potpuno neizvedivo: čak i za duljine raspršivanja od $l = 128$, pronalaženje kolizije s običnom *hash* funkcijom putem rođendanskih napada približava se izvedivosti [5]. Razlog je taj što se takav kolizijski napad na običnu *hash* funkciju može izvesti izvan mreže i neovisno o bilo kojem tajnom ključu (i stoga ne zahtijeva interakciju s legitimnim vlasnikom k), te je snažno paraleliziran. Nijedna od ovih prednosti za napadača ne postoji kada napada NMAC. Osim toga, mnogo je lakše pronaći kolizije analitičkim metodama (npr. [6, 32, 33]) protiv funkcije bez ključa nego razbiti sheme. Konačno se napominje da je varijanta NMAC-a, u kojoj je vanjska funkcija ključna, ali ne i unutarnja (tj. $F_k(F(_x))$), podložna je istom napadu kroz obične kolizije kao konstrukcija samo za dodavanje i znatno je slabija od NMAC.

6.3. Napad proširenja

Kada se uzme u obzir konstrukcija „samo na početku“: $MAC_k(x) = F(k, x)$ (tj. ključ k se dodaje podacima x, a *hash* funkcija – s fiksnom početnom varijablom – izračunava se na spojenim informacijama). Zbog iterativne strukture F lako je vidjeti da ako se zna vrijednost $MAC_k(x)$ gdje x sadrži integralni broj blokova, tada se može izračunati vrijednost MAC_k na bilo kojem proširenju y od x (tj. niz y koji sadrži x kao prefiks) samo korištenjem rezultata $MAC_k(x)$ kao međuvrijednosti varijable ulančavanja u izračunavanju $MAC_k(y)$. Za ovaj napad nije potrebno znanje ili izravan napad na ključ k. U NMAC-u ovaj je napad spriječen vanjskom primjenom F_{k1}, čime se izbjegava izlaganje rezultata iterirane funkcije F_{k2}.

Kao što je ranije navedeno, napadi proširenja mogu omogućiti transformaciju napada na *hash* funkciju koja koristi nasumičnu. ali poznatu početnu varijablu u napadu na tajne početne varijable. Poznati su takvi napadi protiv MD4 [6] i vjerojatno je da postoje protiv MD5 [32, 33]. Međutim, ovi su napadi neprimjenjivi na MAC konstrukcije gdje je, kao što je gore navedeno, vanjska primjena *hash* funkcije sprječava napade proširenja.

6.4. Napad podijeli pa vladaj

Razmotrit će se metoda poznata kao metoda „omotnice“ koja kombinira gore navedene konstrukcije dodavanja prije i samo na početku, naime $MAC_{k1,k2}(x) = F(k_1, x, k_2)$. Preneel i van Oorschot [13] primjećuju da u napadu usmjerenom na oporavak cijeloga ključa ne treba raditi eksponencijalno vrijeme na dodanoj duljini ključeva k_1 i k_2, ali se mogu oporaviti oba ključa u ukupnom vremenu koje je eksponencijalno na dužina jednoga ključa. To se postiže tako da se prvo pronađu kolizije u MAC funkciji, a zatim se iscrpno traži ključ (k_1) koji proizvodi te kolizije. Nakon što imamo pravi k_1, jednostavno je pronaći k_2 iscrpnim pretraživanjem. Iako je ovaj napad nepraktičan, on služi za ilustraciju osnovne činjenice da snaga funkcije dolazi iz pojedinačnih ključeva, a ne iz njihove kombinirane duljine.

Sličan napad vrijedi i protiv NMAC-a. To nije u suprotnosti s analizom koja pokazuje sigurnost NMAC-a na temelju snage pojedinačnih temeljnih funkcija, tj. funkcije kompresije s ključem kao MAC-a i iterirane funkcije s ključem kao slabo otporne na kolizije. Napad zavadi pa vladaj pokazuje da se izraz $\varepsilon_f + \varepsilon_F$ može zamijeniti mnogo jačim $\varepsilon_f * \varepsilon_F$. Također, služi kako bi

pokazao da korištenje jednoga *l*-bitnog drugoga ključa u HMAC-u ne slabi funkciju protiv iscrpnoga pretraživanja.

6.5. Usporedba s konstrukcijom [13]

U [13] je predložena konstrukcija koja je također varijanta metode omotnice. Koristi početnu varijablu s ključem i pridodan ključ, ali koristi i treći ključ koji se primjenjuje kako bi utjecao na unutarnje iteracije funkcije kompresije u upotrebi. Svi su ovi ključevi izvedeni iz jednoga temeljnog ključa. Ovo je heuristička mjera namijenjena suprotstavljanju mogućim slabostima funkcije kompresije u uporabi, a formalna analiza konstrukcije nije pružena. Napominje se da je ova konstrukcija „nametljivija“, u smislu da zahtijeva još neke promjene u postojećim *hash* funkcijama, te utječe na performanse na umjeren, ali zamjetan način.

6.6. Usporedba s RFC1828

MAC shema opisana u RFC1828 [17] predložena je kao standardni mehanizam za provjeru autentičnosti poruke u kontekstu IP (Internet Protocol) sigurnosti. Ova se funkcija, koja koristi MD5 kao temeljnu *hash* funkciju, temelji na metodi omotnice, ali se dodaje ključ na početku na granicu cijeloga bloka. Osim toga, koristi isti ključ za dodavanje prije i na početku. Najbolja analiza poznata za ovu vrstu funkcije dana je u [30] koja pokazuje da se pri korištenju različitih i neovisnih ključeva (za predočenje i dodavanje) sigurnost funkcije može temeljiti na pseudoslučajnim svojstvima temeljne funkcije kompresije. NMAC funkcija predstavljena u radu prikazuje analizu sigurnosti u dva važna aspekta: zahtijeva slabije pretpostavke o temeljnoj *hash* funkciji, a sigurnost je temeljne *hash* funkcije očuvana na znatno jači način nego u analizi u [30]. Druga je važna razlika u tome što se HMAC varijanta bolje nosi s upotrebnom jednoga ključa od konstrukcije RFC1828; u drugom, korištenje istoga ključa za dodavanje prije i dodavanje iza čini analizu u [30] manje primjenjivom i, posebno, čini shemu osjetljivom na gore spomenuti napad vraćanja ključa [31].

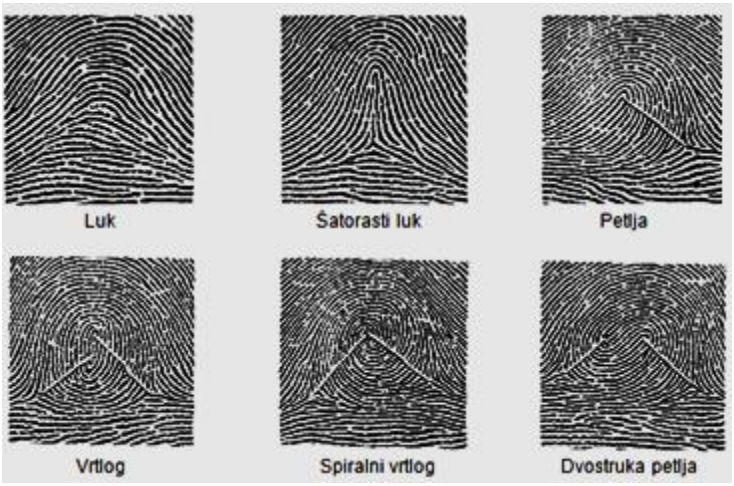
HMAC je sada zamijenio RFC1828 konstrukciju kao obveznu za implementaciju transformacije provjere autentičnosti za internetske sigurnosne protokole [22].

7. PRIMJER UPOTREBE HMAC-A

Racionalnu primjenu HMAC-a izvršile su nigerijske zatvorske službe (NPS) koje su realizirale dizajn i implementaciju biometrijskoga sustava. Sustavom je omogućeno da se pomoću biometrijskih podataka, u ovom primjeru otiskom prsta, vrši upis korisnika tj. zatvorenika gdje se otisak prsta snimi koristeći čitač otiska prsta te se zatim otisak kriptira primjenjujući HMAC. Zatim je moguće jedinstveni HMAC korisnika koji je generiran pohraniti ili u bazu podataka aplikacije ili na pametnu memorijsku karticu za buduću upotrebu i referenciranje.

7.1. Korištenje biometrijskih podataka u kriptografiji

Biometrija je znanost koja se bavi točnom identifikacijom ljudi na temelju anatomskih karakteristika (npr. otisak prsta, šarenica, dlan, glas i dr.) ili osobina ponašanja (npr. potpis).



Slika 7.1. Primjeri grebena na otisku prsta [34]

Sam pojam biometrije potječe iz grčkog jezika i sastoji se od dvije riječi: bios = „život“ i metron = „mjera“. Biometrija posjeduje nekoliko prednosti za razliku od tradicionalnih metoda provjere identiteta gdje za primjer možemo uzeti osobne iskaznice (tokene) ili lozinke, a neke od prednosti su:

- kod identifikacije biometrija eliminira obvezu pamćenja lozinke ili nošenjem tokena
- jednostavna za korištenje

- neprenosiva
- nepromjenjiva tijekom života korisnika
- teško je krivotvoriti
- osoba koju se treba identificirati ne mora biti fizički prisutna na mjestu identifikacije

Supstitucijom PIN-ova (ili korištenjem biometrije uz PIN-ove), biometrijske tehnike potencijalno mogu pomoći u zaustavljanju neovlaštenoga pristupa. Primjena biometrijske tehnologije iz dana u dan dobija na značaju u bankarskim sustavima. Naime, biometrija se tu koristi najčešće u vidu korištenja otiska prsta ili prepoznavanja šarenice oka s namjerom kontroliranja pristupa određenim sektorima bankarskog sustava te postizanju većeg stupnja sigurnosti transakcija kao i očuvanja baza. Jedan od problema korištenja starih metoda identifikacije kao što su PIN-ovi ili lozinke je mogućnost da ih korisnik jednostavno zaboravi. Drugi segment koji se razmatra kod identifikacije su osobni dokumenti. Osobna iskaznica, putovnica i drugi predstavljaju dostupne osobne dokumente, no čak i oni imaju negativnu osobinu. Naime, kod njih uvijek postoji mogućnost krađe, gubitka ili krivotvorenja. Razmatrajući sve navedeno, došlo je do toga da se uporabom biometrije svi navedeni problemi riješe te da se poboljša sigurnost, a najkorištenije biometrijske oznake su lice, šarenica oka i otisak prsta.

Biometrijski sustavi se mogu podijeliti u dvije kategorije: ručni sustav koji se temelji na staroj metodi biometrijskoga prikupljanja i provjere autentičnosti pomoću tinte i papira te automatizirani biometrijski sustav.

Ručni sustav nije otporan na manipulacije i on je vrlo je osjetljiv na napade. Stara metoda za biometrijsko prikupljanje i provjeru autentičnosti može se lako narušiti za kratko vremensko razdoblje stvarajući prostor za podatke i zapise kojima se neprestano manipulira, što uvelike utječe na učinkovito donošenje odluka. Ranjivost ručnog sustava daje prostor za manipulaciju kao što je detekcija pogrešnog ili zamjena identiteta što je riješeno implementacijom automatiziranog sustava identifikacije gdje je osigurano jedinstvo biometrijskih obilježja gdje je evidentno da ne postoje dvije osobe koje posjeduju iste biometrijske osobine.

Automatizirani biometrijski sustav je elektronička verzija unaprijeđenoga ručnoga biometrijskog sustava. Automatiziranim biometrijskim sustavom upravljaju administratori s namjerom identifikacije pojedinca ilil kriminalca prema tragovima ostavljenim na mjestu zločina; to je iz razloga zato što se biometrijske informacije oslanjaju na tjelesna obilježja kako bi se napravila osobna identifikacija.

Administratori koji upravljaju biometrijskim sustavom automatizirane identifikacije nemaju razloga manipulirati sustavom koji koriste krajnji korisnici u bilo koju negativnu svrhu, kao što je pravo pristupa ili nezakonito stjecanje neovlaštenih privilegija, iz razloga što svaki pojedinac posjeduje jedinstvena biometrijska svojstva te se biometrijska tehnologija koristi kao oblik upravljanja pristupom identitetu i kontrole pristupa.

Biometrijske su značajke osjetljivi podaci koji su univerzalni, jedinstveni, postojani i digitalno prikupljani, prihvatljivi u smislu brzine i izvedbe te ih je potrebno adekvatno zaštititi kako bi se eliminiralo iskrivljavanje podataka i manipulacija od strane varalica.

U analizi je poboljšana sigurnost biometrijskih obilježja prikupljenih od zatvorenika upotrebom kriptografskih *hash* funkcija za šifriranje snimljenih detalja otiska prsta putem čitača otiska prsta. Algoritmi za prepoznavanje otiska prsta koriste se za analizu detalja otiska prsta, strukture grebena i gustoće slike kako bi se identificirao pojedinac [35] budući da se svaka pojedinačni otisak prsta razlikuje po svojim obilježjima.

Nadalje, biometrijska autentifikacija otiskom prsta je najpoznatija i najraširenija metoda od svih biometrijskih tehnologija koja je temeljito provjerena kroz razne provedbe te je dokazala svoju pouzdanost i učinkovitost u kriminalističkim istragama [36]. Time se provjera autentičnosti otiskom prsta čini najprihvatljivijim oblikom biometrijske tehnologije.

7.2. Poteškoće korištenja biometrijskih podataka

Unatoč tome što biometrijska tehnologija otiska prsta nudi snažnu sigurnosnu zaštitu i zbog jedinstvenosti biometrijske tehnologije otiska prsta koja je izuzetno korisna u zaštiti osjetljivih informacija, biometrijska tehnologija je na niskoj razini primjene u nekim značajnim institucijama u Nigeriji, posebno primjerice u zatvorima gdje je ova tehnologija vrlo važna.

Pravilna implementacija biometrijskog sustava u zatvorima je od iznimne koristi jer se zbog loše implementacije ili samog nedostatka sustava identifikacije putem otiska prsta može dogoditi neželjen događaj zamjene identiteta ili identifikacija pogrešnog identiteta što je vidljivo u slučaju [37], a vjerojatno i u brojnim drugim.

Prilikom implementacije biometrijskog sustava u nigerijskim zatvorima proučavan je već postojeći sustav kod kojeg se uočio veliki problem. Naime, radi se o tome da je korišteni sustav

imao manu lokaliziranosti čime se nudila mogućnost zatvoreniku ukoliko pobjegne iz zatvora da slobodno prijeđe u neki drugi dio države s argumentacijom da je slobodni građanin pošto ga nije moguće jednostavno pratiti i nadzirati zbog prevelike lokaliziranosti sustava, a to se tražilo i riješilo implementacijom biometrijskog sustava koji se izborio sa tim zahtjevima.

7.3. Koncept

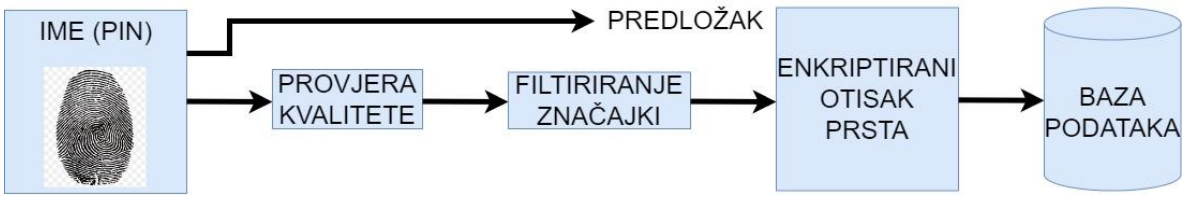
U nastavku rada prikazat će se koncept i same komponente modela te će se prikazati princip njegovoga funkcioniranja.

7.3.1. Sastavnice modela

Biometrijski model korišten za proučavanje sastoji se od sljedećih procesa:

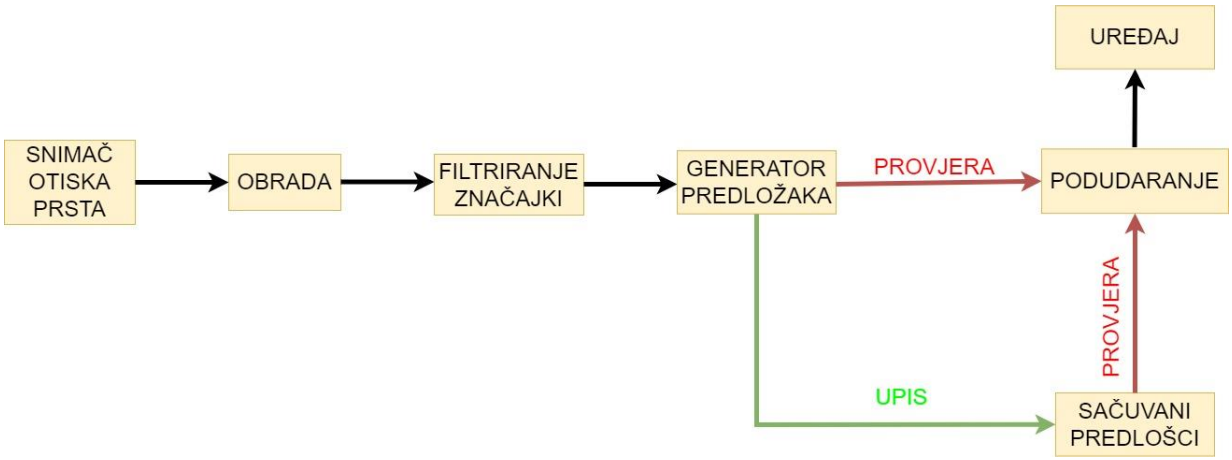
1. upis / snimanje otiska prsta,
2. prepoznavanje / provjera autentičnosti otiska prsta i
3. određivanje procesa.

- Upis – tijekom ovog procesa pojedinac stavi prst na senzor/skener gdje se izvrši snimanje otiska prsta. Snimljeni otisak prsta se zatim analizira algoritmom koji izdvaja detalje otiska prsta. Proces upisa prikazan je na slici 8.1.



Slika 7.2. Prikaz biometrijske prijave

- Autentifikacija – u ovom procesu se vrši usporedba biometrije jedan naprema jedan sa snimljenom biometrijom s namjerom smanjenja manipulacije i osiguranja da se u bazi nalazi prava osoba, a kako bi se to ostvarilo kao pomagalo se koriste pametne kartice ili identifikacijski brojevi. Automatizirani sustav identifikacije otiska prsta izvodi komparaciju ulazne slike otiska prsta i prethodno registrirane podatke kako bi utvrdio autentičnost snimljenih detalja otiska prsta.



Slika 7.3. Proces autentifikacije

- Određivanje procesa

Algoritam korišten u sustavu slijedi sljedeće korake:

1. u prvom koraku pojedinac stavlja prst na skener otiska prsta kako bi snimio detalje otiska prsta;
2. zatim su detalji otiska prsta izvađeni i kvaliteta je poboljšana uklanjanjem prljavštine;
3. u trećem se koraku filtrirani detalji otiska prsta hashiraju;
4. nakon toga je hashirani otisak prsta šifriran i generiran HMAC kod;
5. potom se šifrirani otisak prsta i HMAC kod pohranjuju u bazu podataka sustava.

Slika 7.4. Enkriptirani otisak prsta i HMAC kod [38]

6. slijedi autentifikacija otiska prsta korisnika: ako je korisnik pronađen, otvara se obrazac za registraciju osobe i registrira se osoba te joj se odobri pristup, inače se potrebno vratiti na korak 5,
7. naposlijetku slijedi završetak algoritma.

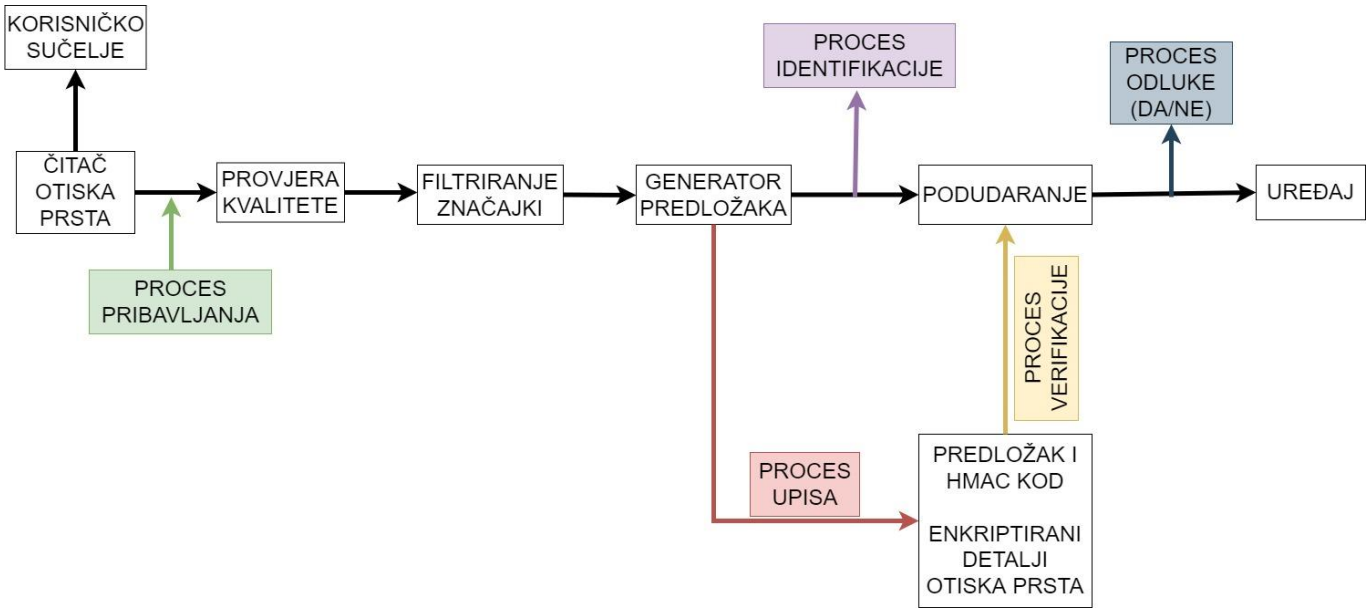
7.4. Modeli prijetnji na biometrijske podatke

Postoje različiti načini biometrijskih napada u današnjem globalnom društvu [39], a oni uključuju:

a) Slabosti vezane uz referanciranje i bazu podataka – neželjeni korisnik može snimati podatke tijekom prijenosa ili tijekom pohrane u bazu podataka. Kod realizacije gdje su biometrijski podaci sačuvani na uređajima koji koriste pojedinci, neželjeni korisnik koji ima pristup uređaju ima nesmetan pristup biometrijskim podacima, ali to se sprječava čuvanjem biometrijskih podataka na centralnom sustavu.
b) Napadi na sustav – prijetnje na biometrijske sustave su svakako izvedivi i moraju se uzeti u razmatranje kada se radi o značajnim vrijednostima i sofisticiranim napadačima. Za sprječavanje napada koriste se sigurnosne metode koje nisu specifične za biometrijske sustave. Valja napomenuti da se kod pohranjivanja podataka stvara značajna količina podataka koje je nužno zaštiti jer napadačima oni predstavljaju veliku vrijednost.

c) Presretanje izlaza senzora – neželjeni korisnik može pokušati izmijeniti ili presresti podatke koji se dobiju sa snimača otiska prsta. Prethodno snimljeni otisak može koristiti ili se može zamijeniti biometrijskim podacima druge osobe prilikom upisa u bazu.
d) Prijetnja od doušnika – svi sigurnosni sustavu su u opasnosti od napada koje može izvesti administrator ili operator sustava te se ovisno o razini pristupa mogu korigirati parametri sustava.
e) Prisila – neželjeni korisnik može prisliti legitimnoga korisnika (na primjer pod prijetnjom oružjem) da mu odobri pristup sustavu.

7.5. Predloženi biometrijski sustav za zatvorsku službu



Slika 7.5. Predloženi sustav za zatvor

Predloženi biometrijski sustav za nigerijske zatvore sastoji se od nekoliko segmenata. Prvi segment predstavlja čitač otiska prsta koji je zadužen za prikupljanje neobrađenih biometrijskih podataka osobe u formatu slike. Nastavno na čitač otiska prsta slijedi segment provjere kvalitete koji pretprocesira snimljene biometrijske značajke.

Idući segment, zadužen za filtriranje značajki, radi na na biometrijskim podacima i izdvaja istaknuti skup značajki za predstavljanje. Tijekom procesa upisa korisnika izdvojeni skup značajki se, označen identitetom korisnika, šifrira i pohranjuje u biometrijski sustav koji se

naziva predložak s pripadajućim HMAC kodom koji se generira. HMAC kod se koristi za dodatno poboljšanje sigurnosti izdvojene biometrijske značajke [40].

Kod segmenta podudaranja vrši se usporedba skupa značajki izdvojenih tijekom provjere autentičnosti s prijavljenim predlošcima i generiraju se rezultati podudaranja.

Na završetku, kod segmenta odlučivanja radi se obrada rezultata podudaranja te se isti koriste kako bi se utvrdio ili bolje rečeno potvrdio identitet osobe.

Dakle, nastavno na navedene i objašnjene segmente vidljivo je da biometrijski sustav ima nekoliko slojeva zaštite koji osiguravaju da je sustav pouzdan, učinkovit, jednostavan za korištenje i vrlo robustan.

7.6. Razvoj aplikacije i izvedba

Aplikacija je razvijena pomoću razvojnih alata i softvera kao što su Java Language Enterprise Edition i Java Database. Testiranja su izvedena korištenjem Microsoftovoga čitača otiska prsta.

Za korištenje aplikacije sa osobnoga računala potrebno je pokrenuti sustav upravljanja Secure Prison. Prvi zaslon koji se pojavljuje nakon pokretanja je stranica za administratorski pristup. Potrebno je izvršiti autentifikaciju administratora, a za autentifikaciju je potrebno korisničko ime i lozinka. Nakon uspješne autentifikacije vrši se preusmjeravanje do stranice korisničkoga sučelja.

Segment, Dijalog za prijavu administratora, onemogućava neovlaštene osobe da koriste aplikaciju. Administrator sustava ima mogućnost mijenjanja i administriranja pristupnoga koda bilo kojem korisniku.

7.6.1. Stranica korisničkoga sučelja

Nakon uspješnog pristupa korisničkom sučelju, korisniku je omogućeno kretanje profilom zatvorenika, dosjeom slučaja i evidencijom aktivnosti i posjetitelja.

Korisničko se sučelje sastoji od tri izbornika: datoteka, pogled i kategorija.

Izbornik datoteka sastoji se od podizbornika za zatvaranje i izlaz. Kada korisnik klikne na podizbornik za zatvaranje, paket se zatvara, a ako korisnik klikne na podizbornik na izlaz, izići će iz paketa.

Izbornik pogleda sadrži četiri podizbornika, tj. radi se o sljedećim podizbornicima: „Zatvorenici“, „Slučajevi“, „Posjete“ i „Aktivnosti“. Ispod se nalaze snimke zaslona koje prikazuju iste.

Izbornik opcija ima samo podizbornik „Korisnik“ i podizbornik „Računi“. Podizbornikom „Računi“ omogućeno je administratoru da putem izbornika „Opcija“ kreira novoga korisnika, ali i izbriše postojećega korisnika i posebno odredi korisničku razinu pristupa sustavu.

7.6.2. Snimanje otisaka prstiju zatvorenika

U podizborniku profila zatvorenika dobivaju se neke osnovne informacije i bilježe se detalji zatvorenikovoga otiska prsta putem Microsoft čitača otiska prsta.

Na slici 7.6. se nalaze šifrirani detalji otiska prsta i odgovarajući HMAC kod.

Slika 7.6. Prikaz generiranoga šifriranog otiska prsta i HMAC-a [38]

8. PROGRAMSKA REALIZACIJA POMOĆU DOSTUPNIH KRIPTOGRAFSKIH KNJIŽICA U PYTHONU I PRIMJERI

U nastavku rada opisane su kriptografske knjižice u Pythonu koje podržavaju primjenu i generiranje kodova za provjeru autentičnosti poruke te će se uz sam opis knjižica demonstrirati primjeri funkcioniranja istih knjižica.

8.1. Hashlib

Računalno hashiranje je postupak primjene determinističke matematičke funkcije na poruku bilo koje veličine i njezinoga pretvaranja u niz bitova fiksne veličine. Funkcija koja se koristi za izračunavanje *hash-a* je jednosmjerna funkcija, stoga ne možemo vratiti transformiranu poruku kako bismo dobili izvornu poruku. Jedini je način da se dobije izvorna poruka pokušaj brutalnoga pretraživanja poruka kako bi se generirao *hash* koji odgovara trenutnom *hash-u*. Funkcije hashiranja koje se obično koriste u kriptografiji nazivaju se sigurnim funkcijama hashiranja. Ponekad se umjesto pojma *hash-a* kao njegova zamjena koristi pojam sažetka poruke. Sigurno hashiranje ima mnoge primjene poput indeksiranja podataka u *hash* tablici, digitalnih potpisa, otkrivanja duplikata, identifikacija datoteka, upotrebe kao kontrolnih zbrojeva za provjeru oštećenja i dr.

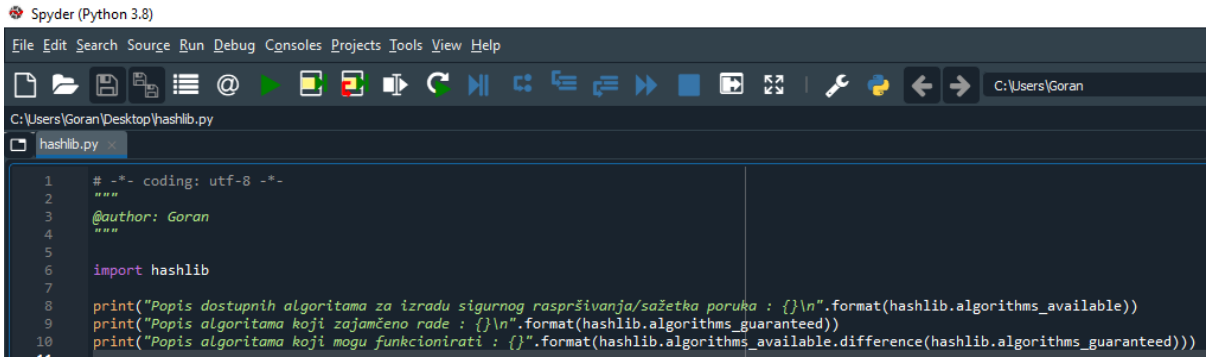
U Python-u postoji modul nazvan hashlib koji omogućuje implementaciju većine često korištenih *hash* funkcija. U nastavku će rada biti prezentirano kako se može koristiti hashlib API za generiranje *hash* poruka s jednostavnim i lako razumljivim primjerima.

Hashlib [41] pruža popis algoritama koji su dostupni na svim platformama, a dostupnost nekih algoritama ovisit će o OpenSSL biblioteci koju Python koristi. Hashlib pruža konstruktor za algoritme koji su također dostupni na svim platformama. Algoritmi koji ovise o OpenSSl verziji dostupni su putem metode pod nazivom *new()*.

Hashlib nudi dva važna atributa koji nam daju na znanje popis sigurnih algoritama hashiranja dostupnih na određenom sustavu:

- Algorithms_guaranteed – atribut vraća popis algoritama koji su implementirani u ovoj knjižici i za koje je zajamčeno da rade. Hashlib nudi konstruktore za svaki od ovih algoritama.
- Algorithms_available – atribut vraća popis svih mogućih algoritama dostupnih na platformi. Uključuje dostupne algoritme putem OpenSSL biblioteke instalirane na sustavu.

Na slici 8.1. je prikazan kod koji se koristi za provjeru popisa algoritama dostupnih na sustavu, a na slici 8.2. nalazi se dobiveni izlaz sa popisom dostupnih algoritama koji se koriste u ovom radu u primjerima koji slijede u nastavku.



Slika 8.1. Kod za provjeru dostupnih algoritama

```
In [1]: runfile('C:/Users/Goran/Desktop/hashlib.py', wdir='C:/Users/Goran/Desktop')
Popis dostupnih algoritama za izradu sigurnog raspršivanja/sažetka poruka : {'shake_128', 'md4', 'sha3_224',
'blake2b', 'sha384', 'sha3_512', 'sha3_256', 'ripemd160', 'sha1', 'sm3', 'sha512', 'whirlpool', 'shake_256', 'sha224',
'blake2s', 'sha3_384', 'sha256', 'sha512_256', 'mdc2', 'md5-sha1', 'sha512_224', 'md5'}

Popis algoritama koji zajamčeno rade : {'sha256', 'shake_128', 'sha512', 'sha3_224', 'blake2b', 'sha3_256',
'sha3_512', 'shake_256', 'sha384', 'sha1', 'sha224', 'blake2s', 'sha3_384', 'md5'}

Popis algoritama koji mogu funkcionirati : {'md4', 'whirlpool', 'sha512_256', 'ripemd160', 'mdc2', 'md5-sha1',
'sha512_224', 'sm3'}
```

Slika 8.2. Prikaz dostupnih algoritama

8.1.1. Primjer korištenja SHA-1 pomoću hashlib-a

U navedenom primjeru demonstrirat će se kako se može izračunati siguran *hash* zadanoga niza koristeći algoritam hashiranja SHA-1. SHA-1 algoritam generira *hash* od 20 bajta

za danu ulaznu poruku radeći na bloku od 64 bajta podataka odjednom. Opis korištenoga konstruktora slijedi u nastavku:

- sha1(message_bytes = None) – ovaj konstruktor stvara instancu *sha1* s početnom porukom danom kao bajtovi. Zatim se može koristiti za generiranje sažetka poruke. Također se može stvoriti samo instanca *sha1* bez ikakve početne poruke. Poruke možemo dodati pomoću poziva metode *update().*

Nakon što je opisan sam konstruktor sha1, treba spomenuti i opisati važne metode koje posjeduje instanca sha1, a one su:

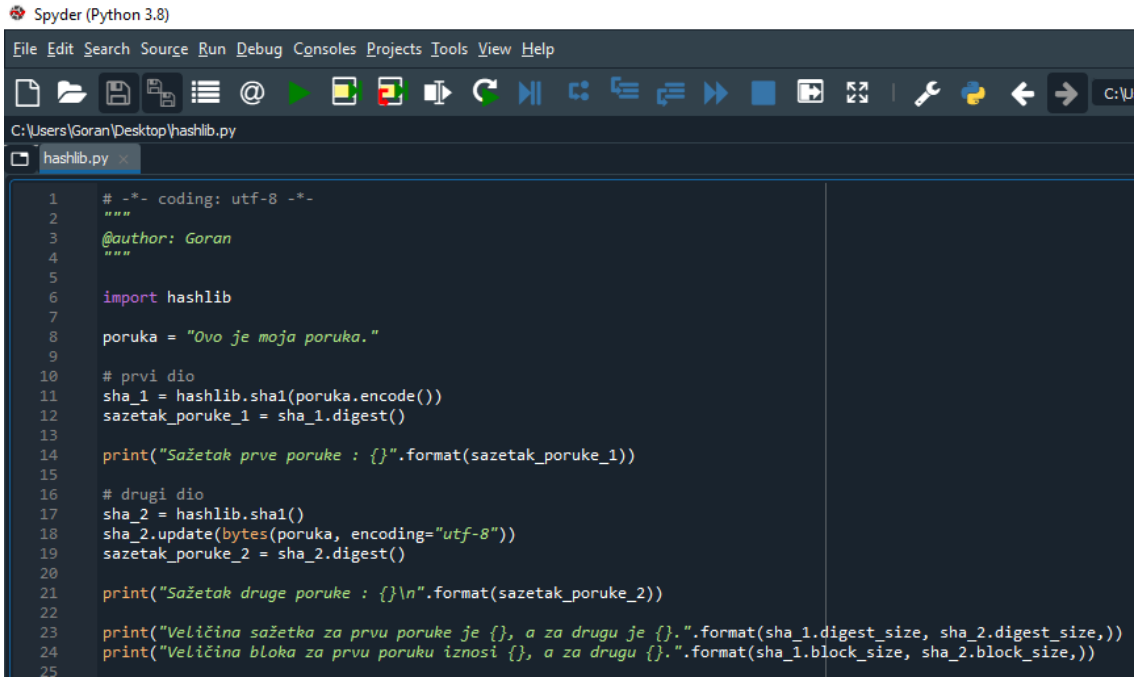
- digest() – metoda vraća sažetak poruke podataka u formatu bajtova, a veličina izlaza je 20 bajtova;
- hexdigest() – metoda vraća sažetak poruke podataka kao heksadecimalne znamenke. Heksadecimalni sažetak bit će dvostruko veći od sažetka bajtova jer jedan bajt može predstavljati dvije heksadecimalne znamenke. Veličina izlaza je 40 heksadecimalnih znamenki.;
- update(message_bytes) – ova metoda dodaje poruke dane kao ulaz u već postojeću poruku. Navedenu metodu možemo pozvati više puta i ona će nastaviti dodavati poruke.

Nastavno na opisani konstruktor i metode koje instanca koristi nužno je spomenuti i same atribute sha1 instance koji su korišteni u primjeru prikazanom u radu, a oni su:

- Block_size – atribut vraća cjelobrojnu vrijednost koja predstavlja veličinu bloka algoritma. Sigurni *hash* algoritmi dijele podatke u blokove gdje svaki blok sadrži broj bajtova podataka na kojima će algoritam raditi.
- Digest_size – atribut vraća cijeli broj koji predstavlja broj bajtova sigurnoga raspršivanja koje će algoritam generirati.

Kod prikazan na slici 8.3. započinje stvaranjem instance SHA1 s prethodno definiranim ulaznim nizom. Nakon toga se unose ulazni podaci u formatu bajtova pozivanjem metode encode() na ulaznom nizu – poruka. Zatim se generira sažetak poruke i ispisuje. Drugi dio koda ponovno stvara instancu SHA1, no bez ikakve početne poruke. Koristi se metoda *update()* za ažuriranje poruke algoritma. Zatim se ispisuje sažetak poruke i na kraju se ispisuje veličina sažetka i veličina bloka algoritma.

Pritom treba pripaziti i imati na umu da se naizmjenično koriste pojmovi sigurni *hash* i sažetak poruke, no oba predstavljaju istu stvar.



Slika 8.3. Primjer korištenja SHA-1 dostupnog u knjižici hashlib

```
In [6]: runfile('C:/Users/Goran/Desktop/hashlib.py', wdir='C:/Users/Goran/Desktop')
Sažetak prve poruke : b'\xe4\xf3\xc8k<>\xc3B\x8cy>/\x1e\x12\x80\xdb\xcfM\x16E'
Sažetak druge poruke : b'\xe4\xf3\xc8k<>\xc3B\x8cy>/\x1e\x12\x80\xdb\xcfM\x16E'

Veličina sažetka za prvu poruke je 20, a za drugu je 20.
Veličina bloka za prvu poruku iznosi 64, a za drugu 64.
```

Slika 8.4. Dobiveni rezultati za prethodni kod

8.2. Kod za autentifikaciju poruke temeljen na hashu pomoću Pythona

HMAC je algoritam koji generira *hash* poruke koristeći kriptografsku *hash* funkciju i tajni kriptografski ključ. Može se koristiti za provjeru integriteta i autentičnosti podataka. Omogućuje izračunavanje autentičnosti i integriteta poruke koristeći zajednički tajni ključ između dvije strane bez upotrebe složene infrastrukture javnih ključeva koja uključuje certifikate.

Python nam daje modul pod nazivom *hmac* [42] koji nudi implementaciju ovog algoritma za hashiranje, a on je jedan od algoritama dostupnih putem Python knjižice hashlib. U ovom će se radu prikazati na jednostavnim primjerima kako se može generirati kod za provjeru autentičnosti poruke pomoću Python modula *hmac*.

8.2.1. Prvi primjer

U prvom će se primjeru predočiti generiranje koda za provjeru autentičnosti poruke za danu poruku na temelju ulaznoga ključa i algoritma sigurnoga raspršivanja koristeći modul *hmac*. Konstruktor koji se nalazi u knjižici *hmac*, a koji je korišten u primjerima, opisan je u sljedećem odlomku:

- new(key, message = None, digestmode = '') – ovaj konstruktor stvara instancu HMAC-a s početnom porukom danom kao bajtovi. Nakon toga se može koristiti za generiranje koda za provjeru autentičnosti poruke. Može se stvoriti samo instanca HMAC-a bez ikakve početne poruke, ali trebat će nam ključ i *digestmod*. Poruke možemo dodati pomoću poziva metode *update()*. *Ključ* mora biti u formatu bajtova. Parametar *digestmod* prihvaća nazive algoritama za sigurno raspršivanje koji su dostupni putem modula hashlib.

Metode koje posjeduju HMAC instance, a koje su korištene u primjeru opisane su u nastavku:

- digest() – metoda vraća kod za provjeru autentičnosti poruke podataka u formatu bajtova. Veličina izlaza ovisi o algoritmu sigurnoga raspršivanja ulaza. Uzmimo za primjer: ako je ulazni algoritam raspršivanja SHA-1, tada će izlaz biti 20 bajtova.
- hexdigest() – metoda vraća kod za provjeru autentičnosti poruke podataka kao heksadecimalne znamenke. Heksadecimalni sažetak bit će dvostruko veći od sažetka bajtova jer jedan bajt može predstavljati dvije heksadecimalne znamenke. Veličina izlaza ovisi o algoritmu sigurnoga raspršivanja ulaza. Npr. ako je ulazni algoritam raspršivanja SHA-1, tada će izlaz biti 40 heksadecimalnih znamenki.
- update(message_bytes) – ova metoda dodaje poruke dane kao ulaz već postojećoj poruci. Ovu metodu možemo pozvati više puta i ona će nastaviti dodavati poruke.

Daljnim tijekom dolazimo do korištenih atributa HMAC instanci, a oni su:

- block_size – atribut vraća cjelobrojnu vrijednost koja predstavlja veličinu bloka algoritma. Sigurni *hash* algoritmi dijele podatke u blokove gdje svaki blok sadrži broj bajtova podataka na kojima će algoritam raditi.
- digest_size – atribut vraća cijeli broj koji predstavlja broj bajtova sigurnoga raspršivanja koje će algoritam generirati.

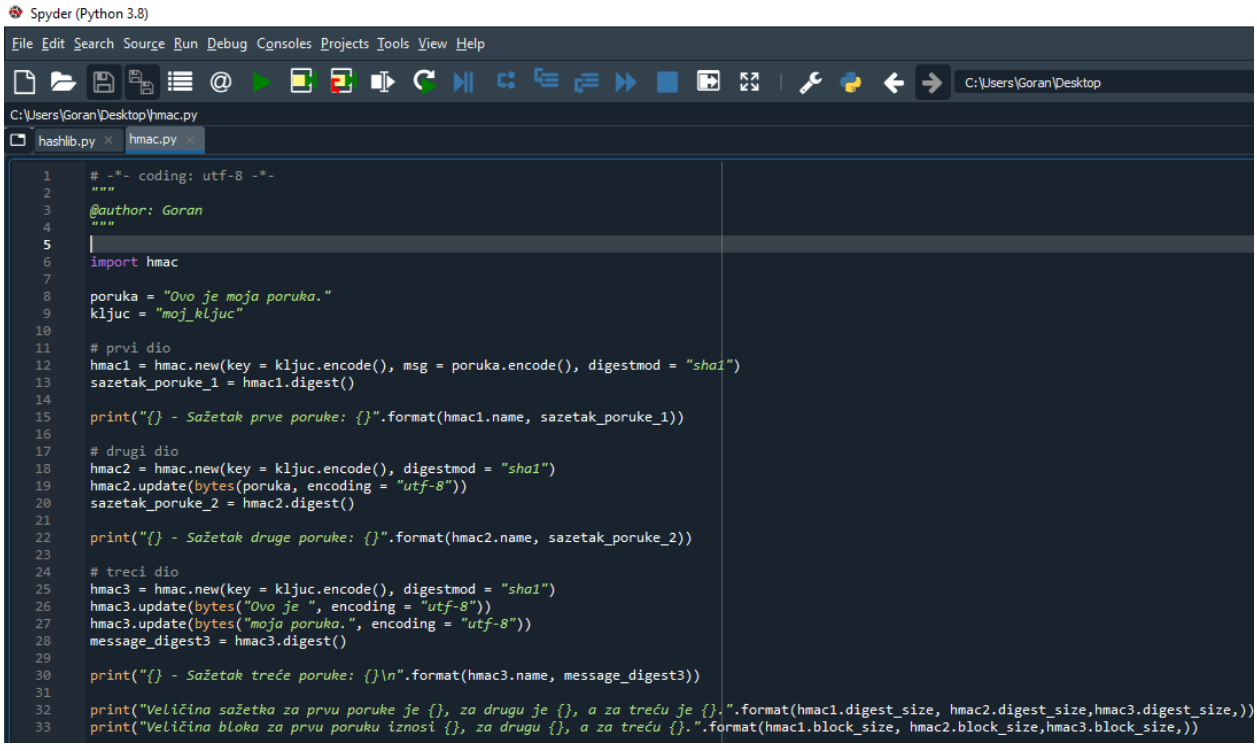
Kod korišten u prvom primjeru započinje inicijalizacijom poruke i ključa. Zatim se generira kod za provjeru autentičnosti poruke za danu poruku koristeći *ključ* i algoritam SHA-1 na tri različita načina.

Kod prvoga se načina generiranja stvara instanca HMAC-a koristeći metodu *new()* dajući joj *ključ* i poruku kao bajtove i naziv algoritma za raspršivanje kao SHA-1 te naposlijetku ispisujemo kod za provjeru autentičnosti poruke.

U drugome se načinu stvara HMAC instanca bez ikakve početne poruke te se zatim koristi metoda *update()* za dodavanje poruke pa se na kraju izračunava sažetak poruke i isti ispisuje.

Treći dio koda stvara instancu HMAC-a bez ikakve početne poruke. Zatim se dodaju poruke u dva navrata pomoću metode *update()*. Na kraju se izračunava sažetak i ispisuje.

Kodom korištenim u primjeru se na kraju ispisuje veličina sažetka i veličina bloka za svaku HMAC instancu.



Slika 8.5. Primjer stvaranja HMAC-a – prvi primjer

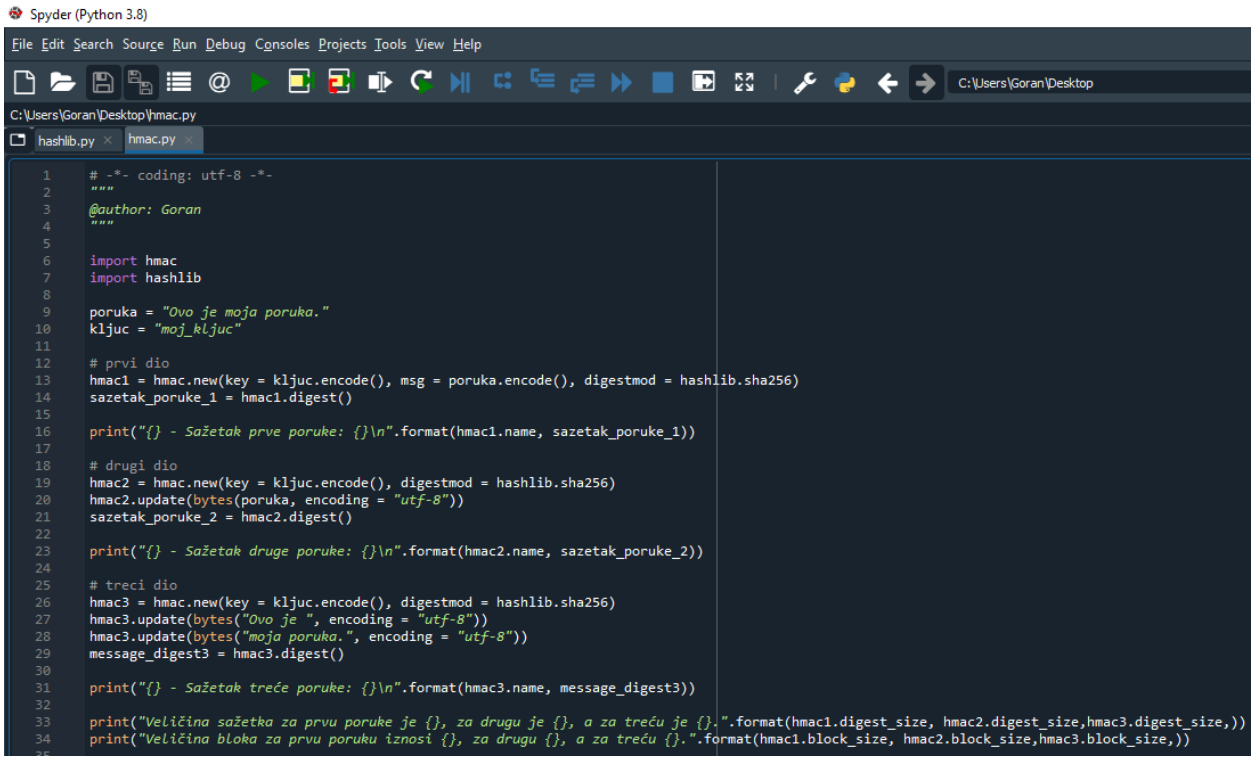
```
In [2]: runfile('C:/Users/Goran/Desktop/hmac.py', wdir='C:/Users/Goran/Desktop')
hmac-sha1 - Sažetak prve poruke: b'\x18XQP"}\x14\xd4\x83`M\x9edyZW\xd0\xfe\x8d-'
hmac-sha1 - Sažetak druge poruke: b'\x18XQP"}\x14\xd4\x83`M\x9edyZW\xd0\xfe\x8d-'
hmac-sha1 - Sažetak treće poruke: b'\x18XQP"}\x14\xd4\x83`M\x9edyZW\xd0\xfe\x8d-'

Veličina sažetka za prvu poruke je 20, za drugu je 20, a za treću je 20.
Veličina bloka za prvu poruku iznosi 64, za drugu 64, a za treću 64.
```

Slika 8.6. Dobiveni izlaz – prvi primjer

8.2.2. Drugi primjer

U drugome se primjeru pojašnjava kako se može ponovno izračunati kod za provjeru autentičnosti poruke koristeći HMAC algoritam, ali u ovome je primjeru korišten SHA256 algoritam za sigurno hashiranje.

Kod korišten u ovom primjeru je potpuno isti kao kod prethodnoga primjera, ali s manjim izmjenama. Koristi se referenca na SHA-256, algoritam koji je dostupan putem hashlib biblioteke.



Slika 8.7. Stvaranje HMAC-a koristeći SHA-256

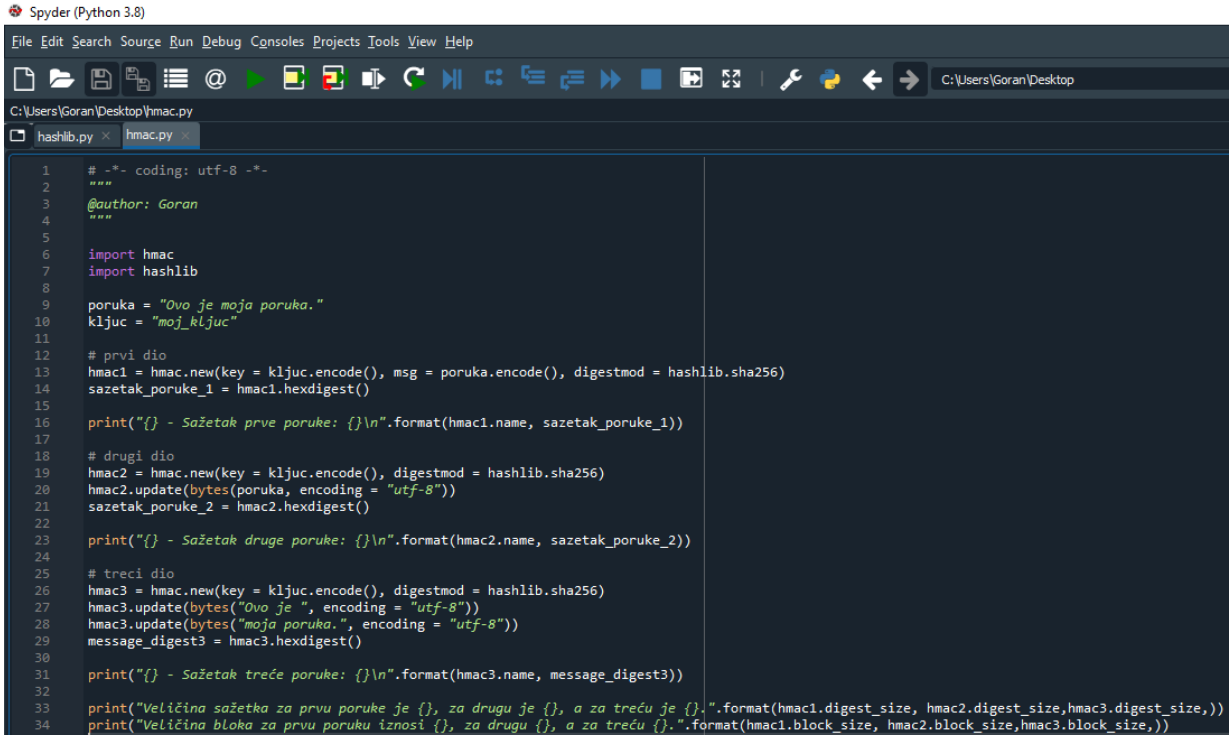
```
In [4]: runfile('C:/Users/Goran/Desktop/hmac.py', wdir='C:/Users/Goran/Desktop')
hmac-sha256 - Sažetak prve poruke: b"\x81p\xb9y\xb5\xbb\xdb\x84\xba1\x1b\xeb4\xd4j\t?1k\xd1'\xf6_
\xd4\x19L~k\\)\xe6\xc4"

hmac-sha256 - Sažetak druge poruke: b"\x81p\xb9y\xb5\xbb\xdb\x84\xba1\x1b\xeb4\xd4j\t?1k\xd1'\xf6_
\xd4\x19L~k\\)\xe6\xc4"

hmac-sha256 - Sažetak treće poruke: b"\x81p\xb9y\xb5\xbb\xdb\x84\xba1\x1b\xeb4\xd4j\t?1k\xd1'\xf6_
\xd4\x19L~k\\)\xe6\xc4"

Veličina sažetka za prvu poruke je 32, za drugu je 32, a za treću je 32.
Veličina bloka za prvu poruku iznosi 64, za drugu 64, a za treću 64.
```

Slika 8.8. Dobiveni izlaz – drugi primjer

8.2.3. Treći primjer

U trećem se primjeru proučava kako se može generirati heksadecimalni kod za provjeru autentičnosti poruke koristeći HMAC algoritam sa SHA256 kriptografskom *hash* funkcijom kao pozadinom.

Kod korišten u ovom primjeru je potpuno isti kao kod prethodnoga primjera s jedinom promjenom da pozivamo metodu hexdigest() za izračunavanje heksadecimalnoga autentifikacijskog koda.



Slika 8.9. Stvaranje HMAC-a koristeći metodu hexdigest za heksadecimalni zapis

```
In [5]: runfile('C:/Users/Goran/Desktop/hmac.py', wdir='C:/Users/Goran/Desktop')
hmac-sha256 - Saţetak prve poruke: 8170b979b5bbdb84ba311beb34d46a093f316bd127f65fd4194c7e6b5c29e6c4

hmac-sha256 - Saţetak druge poruke: 8170b979b5bbdb84ba311beb34d46a093f316bd127f65fd4194c7e6b5c29e6c4

hmac-sha256 - Saţetak treće poruke: 8170b979b5bbdb84ba311beb34d46a093f316bd127f65fd4194c7e6b5c29e6c4

Veličina saţetka za prvu poruke je 32, za drugu je 32, a za treću je 32.
Veličina bloka za prvu poruku iznosi 64, za drugu 64, a za treću 64.
```

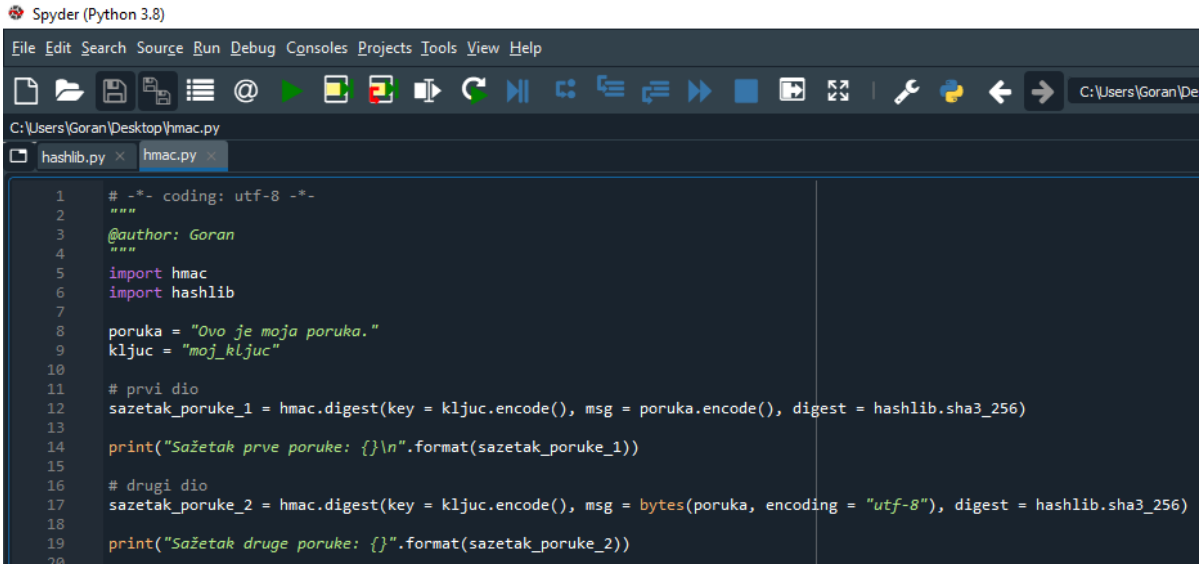
Slika 8.10. Dobiveni izlaz – treći primjer

8.2.4. Četvrti primjer

U četvrtom se primjeru prikazuje kako se može generirati kod za provjeru autentičnosti poruke pomoću metode *digest()* modula *hmac* bez stvaranja instance HMAC-a. Opis metode je dan u nastavku:

- digest(key, msg, digest) – metoda prihvaća ključ, poruku za kodiranje i algoritam sažetka kao ulaz i generira kod za provjeru autentičnosti poruke za danu ulaznu poruku.

Kod korišten u primjeru ilustrira kako se može koristiti metoda *digest()* za generiranje koda za provjeru autentičnosti poruke, za ulaznu poruku koristeći ključ za unos izravno bez stvaranja HMAC instance kao u prethodnom primjeru. Ova metoda radi brže u usporedbi sa stvaranjem autentifikacijskoga koda za male poruke putem HMAC instance jer koristi optimiziranu C implementaciju za stvaranje sažetka.



Slika 8.11. Generiranje koda za provjeru autentičnosti poruke putem digest() metode

```
In [8]: runfile('C:/Users/Goran/Desktop/hmac.py', wdir='C:/Users/Goran/Desktop')
Sažetak prve poruke: b'\xe8\x1c\x8c9\x80u\xadU\x18\x1b\x8f\xe2\x93 s\xe0\x02\xa2u#s\x94\xad\x7f_
\x11\xb0\xd6L9\x90Z'

Sažetak druge poruke: b'\xe8\x1c\x8c9\x80u\xadU\x18\x1b\x8f\xe2\x93 s\xe0\x02\xa2u#s\x94\xad\x7f_
\x11\xb0\xd6L9\x90Z'
```

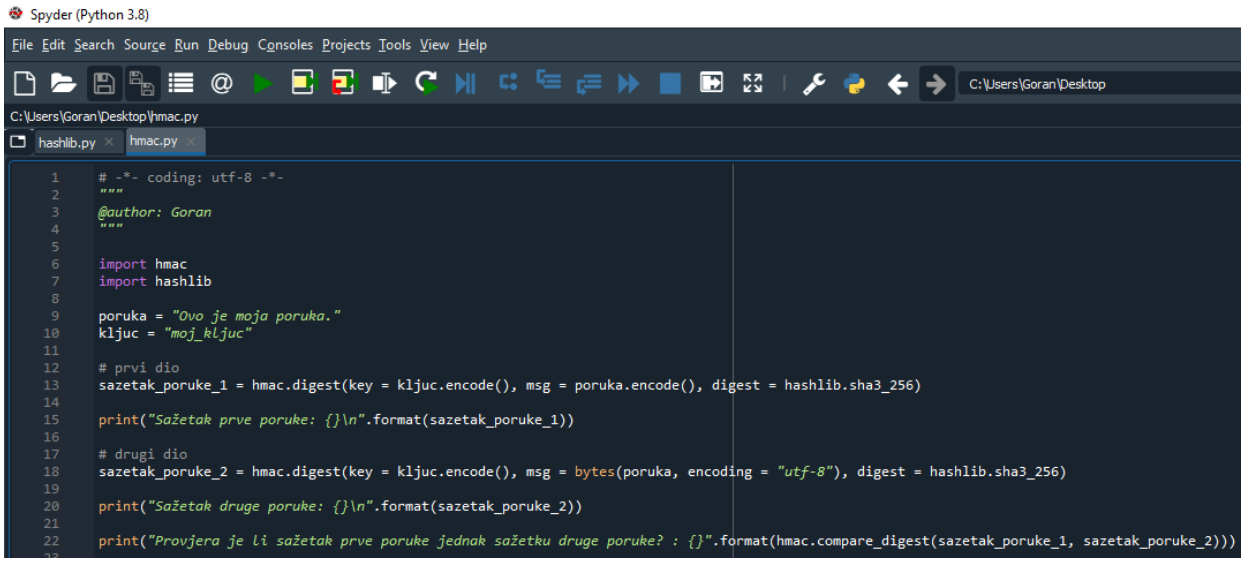
Slika 8.12. Dobiveni izlaz – četvrti primjer

8.2.5. Peti primjer

U petom se primjeru prezentira kako se može koristiti metoda modula *hmac*, *compare()*, za usporedbu kodova za provjeru autentičnosti poruka. Sam je opis metode dan u sljedećem odlomku:

- compare(a,b) – ova metoda uspoređuje dva koda za provjeru autentičnosti i vraća True ako su jednaki, a u suprotnom False. Ova funkcija pomaže u sprječavanju napada vremenske analize koji napadačima mogu pomoći da saznaju veličinu koda za provjeru autentičnosti poruke.

Kod korišten u primjeru stvara dva koda za provjeru autentičnosti poruke i uspoređuje ih pomoću metode *compare()*.



Slika 8.13. Usporedba dva koda za provjeru autentičnosti poruke pomoću compare() metode

```
In [9]: runfile('C:/Users/Goran/Desktop/hmac.py', wdir='C:/Users/Goran/Desktop')
Sažetak prve poruke: b'\xe8\x1c\x8c9\x80u\xadU\x18\x1b\x8f\xe2\x93 s\xe0\x02\xa2u#s\x94\xad\x7f_
\x11\xb0\xd6L9\x90Z'

Sažetak druge poruke: b'\xe8\x1c\x8c9\x80u\xadU\x18\x1b\x8f\xe2\x93 s\xe0\x02\xa2u#s\x94\xad\x7f_
\x11\xb0\xd6L9\x90Z'

Provjera je li sažetak prve poruke jednak sažetku druge poruke? : True
```

Slika 8.14. Dobiveni izlaz – peti primjer

9. ZAKLJUČAK

Dvije strane koje komuniciraju preko nesigurnoga kanala trebaju tehniku kojom se otkriva svaki pokušaj izmjene informacija koje jedna strana šalje drugoj strani ili lažiranja njihovog porijekla. Najčešće se takav mehanizam temelji na zajedničkom ključu između dviju strana, a ovdje se to naziva MAC ili kod za provjeru autentičnosti poruke. Osim pojma MAC-a često se spominju pojmovi vrijednosti provjere integriteta ili kriptografskoga kontrolnog zbroja.

Pošiljatelj dodaje podacima autentifikacijsku oznaku izračunatu kao funkciju podataka i zajedničkoga ključa. Pri prijemu primatelj ponovno izračunava oznaku provjere autentičnosti na primljenoj poruci pomoću zajedničkoga ključa i prihvaća podatke kao valjane, samo ako ta vrijednost odgovara oznaci priloženoj primljenoj poruci.

Primjena kriptografskih *hash* funkcija kao što su MD5 ili SHA-1 za provjeru autentičnosti poruke je postala kriterij u brojnim internetskim aplikacijama i protokolima. Konstrukcijske sheme provjere autentičnosti poruka, NMAC i HMAC, koje se temelje na kriptografskoj *hash* funkciji su dokazano sigurne dok temeljna *hash* funkcija ima razuman nivo kriptografske snage. Štoviše, pokazuje se da sheme zadržavaju gotovo svu sigurnost temeljne *hash* funkcije te su, osim toga, učinkovite i praktične. Izvedba samih shema je temeljena na *hash* funkciji. Naime, koriste se *hash* funkcije kao crna kutija tako da se široko dostupne knjižnice koda ili hardver mogu koristiti za njihovu implementaciju na jednostavan način, a pritom je lako podržana zamjenjivost temeljne hash funkcije.

Analiza uzima u obzir svaki generički napad na MAC sheme i pokazuje da je takav napad uspješan samo u slučaju ako je temeljna *hash* funkcija slaba. Ukratko, ono što analiza tvrdi jest da ako se ikad pronađu značajne slabosti u MAC shemama koje se analiziraju, onda ne samo da temeljna *hash* funkcija mora biti izostavljena iz konkretnih upotreba već se mora izostaviti i iz širokoga raspona druge standardne i popularne upotrebe. Dapače, konstrukcije zahtijevaju od *hash* funkcije znatno slabija svojstva od standardne funkcije bez kolizija. Konkretno, trenutne uspješne metode za pronalaženje kolizija u MD5 su neprimjenjive za razbijanje shema kada je MD5 *hash* funkcija u upotrebi.

Kod primjera upotrebe HMAC-a prikazanoga u radu osnovni je naglasak na procesu automatizacije stare metode biometrijske identifikacije koju su koristile zatvroske službe u Nigeriji. Detalji otiska prsta su šifrirani i hashirani prije pohranjivanja u bazu podataka sustava za buduće referenciranje i korištenje u donošenju osjetljivih odluka od strane agencija za

provođenje zakona, čime se otežava kažnjenicima da zaobiđu sigurnosni sustav. Ovaj je automatizirani sustav poboljšan, pouzdan, brz i vrlo jednostavan za korištenje.

Koristeći programski jezik Python te njegove dostupne knjižice koje podržavaju kriptografske funkcije te funkcije za provjeru autentičnosti poruke, hmac i hashlib, demonstrirani su primjeri kojima se prikazuju različiti načini stvaranja koda za provjeru autentičnosti. Pritom su definirani atributi, konstruktori i metode koje su korištene u primjerima kako bi se mogao razumjeti prikazani programski kod.

10. SAŽETAK

Tema je ovoga rada HMAC – kod za provjeru autentičnosti poruke koji koristi kriptografsku *hash* funkciju i tajni kriptografski ključ. U samome je radu prvobitno prikazan princip funkcioniranja i komponente kodova za provjeru autentičnosti poruke. Zatim se nastavno na tu analizu izvršilo proučavanje dva tipa koda za provjeru autentičnosti poruke (NMAC i HMAC). Kod proučavanja se poseban naglasak dao na sigurnosnu analizu te primjenu HMAC-a u biometrijskom sustavu. Na samom su kraju demonstrirane kriptografske knjižnice u Pythonu koje podržavaju implementaciju kodova za provjeru autentičnosti poruke.

Ključne riječi: kod za provjeru autentičnosti poruke, *hash* funkcija, biometrija, kriptografija, HMAC, NMAC, Python, hashlib

11. ABSTRACT

The topic of this paper is HMAC - a message authentication code that uses a cryptographic *hash* function and a secret cryptographic key. In the paper itself, the principle of functioning and components of the codes for verifying the authenticity of the message were initially presented. Then, following this analysis, two types of message authentication codes (NMAC and HMAC) were studied. During the study, special emphasis was placed on security analysis, and in addition, special attention was paid to the application of HMAC in the biometric system. At the very end, cryptographic libraries in Python that support the implementation of message authentication codes are demonstrated.

Keywords: message authentication code, *hash* function, biometrics, cryptography, HMAC, NMAC, Python, hashlib

12. POJMOVNIK I KRATICE

HMAC – Hash message authentication code

MAC – Message authentication code

MD5 – Message digest algorithm version 5

SHA – Secure Hash Algorithm

AE – Authenticated encryption

MIC – Message Integrity Code

RFC – Request for Comments

TLS – Transport Layer Security

FIPS – Federal Information Processing Standards

ISO/IEC – International Organization for Standardization/The International Electrotechnical Commission

DES – Data Encryption Standard

NMAC – Nested Message authentication code

CBC-MAC – Cipher block chaining message authentication code

DSS – Digital Signature Standard

SSH – Secure Socket Shell

IPsec – Internet Protocol secure

JSON – JavaScript Object Notation

RIPEMD – Race Integrity Primitives Evaluation Message Digest

PRF – Pseudo Random Function

ASCII – American Standard Code for Information Interchange

HAVAL – Hashing Algorithm with Variable Length of Output

IP – Internet Protocol

NPS – Nigerian Prisons Service

PIN – Personal identification number

ID – Identification

API – Application Programming Interface

OpenSSL – Open Secure Sockets Layer

13. LITERATURA

[1] Nechvatal, J. (1991), "Public-Key Cryptography", Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, s Interneta https://apps.dtic.mil/sti/pdfs/ADA408338.pdf, 1.7.2022.

[2] Merkle C. R., "One way hash functions and DES,", s Interneta, https://link.springer.com/content/pdf/10.1007/0-387-34805-0_40.pdf, 1.7.2022.

[3] Damgård I., "A design principle for hash functions," Advances in Cryptology – Crypto 89 Proceedings, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag, 1989.

[4] Tsudik G., "Message authentication with one-way hash functions,", s Interneta, https://ieeexplore.ieee.org/document/263477/authors#authors, 3.7.2022.

[5] van Oorschot P., Wiener M., "Parallel Collision Search with Applications to Hash Functions and Discrete Logarithms", Proceedings of the 2nd ACM Conf. Computer and Communications Security, Fairfax, VA, November 1994.

[6] Dobbertin H., "Cryptanalysis of MD4,", s Interneta, https://link.springer.com/content/pdf/10.1007/s001459900047.pdf, 4.7.2022.

[7] 802.11-2007 - IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, s Interneta, https://ieeexplore.ieee.org/document/4248378, 5.7.2022.

[8] Schneider B. F., „Hashes and Message Digests", s Interneta, http://www.cs.cornell.edu/courses/cs513/2005fa/NL20.hashing.html, 5.7.2022.

[9] Request for Comments: 4949, s Interneta, https://www.rfc-editor.org/rfc/rfc4949, 5.7.2022.

[10] Bellare M., Kilian J., Rogaway P., "The security of cipher block chaining.", s Interneta, https://cseweb.ucsd.edu/~mihir/papers/cbc.pdf, 7.7.2022.

[11] Bellare M., Canetti R., Krawczyk H., „Keying hash functions for message authentication", s Interneta, https://cseweb.ucsd.edu/~mihir/papers/kmd5.pdf, 8.7.2022.

[12] ISO/IEC FDIS 10118-3, Information Technology - Security Techniques - Hash Functions - Part 3: Dedicated hash functions, s Interneta, https://www.iso.org/standard/67116.html, 9.7.2022.

[13] Preneel B., van Oorschot P., "MD-x MAC and building fast MACs from hash functions," Advances in Cryptology – Crypto 95 Proceedings, Lecture Notes in Computer Science Vol. 963, D. Coppersmith ed., Springer-Verlag, 1995.

[14] Hirose S, "A note on the strength of weak collision resistance," IEICE Transactions on Fundamentals, vol. E87-A, no. 5, pp. 1092-1097, 2004.

[15] Ferguson N., Schneier B., Practical Cryptography, Chapter Hash Functions, pp. 83-96, John Wiley & Sons, 2003.

[16] Rivest R., "The MD5 message-digest algorithm", s Interneta, https://www.ietf.org/rfc/rfc1321.txt, 12.7.2022.

[17] Metzger P., Simpson W., „IP Authentication using Keyed MD5", IETF Network Working Group, RFC 1828, August 1995.

[18] Krawczyk H., Bellare M., Canetti R., „HMAC-MD5: Keyed-MD5 for Message Authentication", s Interneta, https://datatracker.ietf.org/doc/draft-ietf-ipsec-hmac-md5/00/, 15.7.2022.

[19] Rescorla E., Schiffman A., „The Secure HyperText Transfer Protocol", s Interneta, https://datatracker.ietf.org/doc/html/draft-ietf-wts-shttp-01, 17.7.2022.

[20] Freier A.O., Karlton P., Kocher P.C., „The SSL Protocol", s Interneta, https://datatracker.ietf.org/doc/html/draft-ietf-tls-ssl-version3-00, 18.7.2022.

[21] „SHA-1 Broken", s Interneta, https://www.schneier.com/blog/archives/2005/02/sha1_broken.html, 21.7.2022.

[22] Bellare M., Canetti R., Krawczyk H., „HMAC: Keyed-Hashing for Message Authentication", s Interneta, https://datatracker.ietf.org/doc/html/rfc2104, 21.7.2022.

[23] Bellare M., „New Proofs for NMAC and HMAC: Security without Collision-Resistance", s Interneta, https://eprint.iacr.org/2006/043.pdf, 21.7.2022.

[24] Toponce A., „Breaking HMAC", s Interneta, https://pthree.org/2016/07/29/breaking-hmac/, 21.7.2022.

[25] Jongsung K., Biryukov A., Preneel B. Hong S., „On the Security of HMAC and NMAC Based on HAVAL, MD4, MD5, SHA-0 and SHA-1“, s Interneta, https://eprint.iacr.org/2006/187.pdf, 22.7.2022.

[26] Wang X., Yu H., Wang W., Zhang H., Zhan T., „Cryptanalysis on HMAC/NMAC-MD5 and MD5-MAC“, s Interneta, https://www.iacr.org/archive/eurocrypt2009/54790122/54790122.pdf, 23.7.2022.

[27] Turner S., Chen L., „Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms“, s Interneta, https://www.rfc-editor.org/rfc/rfc6151, 25.7.2022.

[28] Bellare M., Guerin R., Rogaway P., „XOR MACs: New methods for message authentication using finite pseudorandom functions“, s Interneta, https://cseweb.ucsd.edu/~mihir/papers/xormacs.pdf, 26.7.2022.

[29] TeamKeccak, „Strengths of Keccak“, s Interneta, https://keccak.team/keccak_strengths.html, 27.7.2022.

[30] Bellare M., Canetti R., Krawczyk H., „Pseudorandom functions revisited: The cascade construction“, s Interneta, https://cseweb.ucsd.edu/~mihir/papers/cascade.pdf, 28.7.2022.

[31] Preneel B., van Oorschot P., „On the security of two MAC algorithms“, s Interneta, https://link.springer.com/content/pdf/10.1007%2F3-540-68339-9_3.pdf, 29.7.2022.

[32] Dobbertin H., „MD5 is not collision-free“ Manuscript, 1996.

[33] Dobbertin H., „The Status of MD5 After a Recent Attack“, RSA Labs' CryptoBytes,

Vol. 2 No. 2, Summer 1996., http://www.rsa.com/rsalabs/pubs/cryptobytes.html

[34] Wikipedia: „Otisak prsta“, s Interneta, https://bs.wikipedia.org/wiki/Otisak_prsta, 30.7.2022.

[35] Simon-Zorita D., Ortega-Garcia J., Cruz-Liamas S., Gonzale-Rodraguez, J., „Minutiae Extraction scheme for fingerprint Recognition system". In proceeding of the International conference on Image processing, 204-257.

[36] Abeng, E. E., „A hybrid Hash message Authentication code (HMAC)“ for authenticating and validating Biometrics Information.

[37] Aluko, M., „The strange case of Governor James Onanefe Ibori“, s Interneta, http://www.nigerdeltapeoplesworldcongress.org/articles/strange_case_of_gov.pdf, 1.8.2022.

[38] E. E. Abeng, W. A. Adesola, „Application of hybrid hash message authentication code approach in biometrics information system design“, s Interneta, https://www.ajol.info/index.php/gjpas/article/download/78924/69245/0, 2.8.2022.

[39] Chander K., Ranjender N., Shectal C., „Biometrics Security using steganography“, s Interneta, https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.302.8734&rep=rep1&type=pdf, 2.8.2022.

[40] Ross A., Jain A., Reisman J., „A hybrid fingerprint matcher”, s Interneta, https://www.sciencedirect.com/science/article/abs/pii/S0031320302003497, 3.8.2022.

[41] Python: „hashlib“, s Interneta, https://docs.python.org/3/library/hashlib.html, 4.8.2022.

[42] Python: „hmac“, s Interneta, https://docs.python.org/3/library/hmac.html, 5.8.2022.