Algoritmi za sažimanje lozinki

Marić, Luka

Sveučilište Josipa Jurja Strossmayera u Osijeku
Filozofski fakultet Osijek
Preddiplomski studij informatologije

Luka Marić

Algoritmi za sažimanje lozinki

Završni rad

Mentor: doc. dr. sc. Tomislav Jakopec

Osijek, 2023.

Sveučilište Josipa Jurja Strossmayera u Osijeku

Filozofski fakultet Osijek

Odsjek za informacijske znanosti

Preddiplomski studij informatologije

Luka Marić

Algoritmi za sažimanje lozinki

Završni rad

Društvene znanosti, informacijske i komunikacijske znanosti,

informacijski sustavi i informatologija

Mentor: doc. dr. sc. Tomislav Jakopec

Osijek, 2023.

IZJAVA

Izjavljujem s punom materijalnom i moralnom odgovornošću da sam ovaj rad samostalno napisao/napisala te da u njemu nema kopiranih ili prepisanih dijelova teksta tuđih radova, a da nisu označeni kao citati s navođenjem izvora odakle su preneseni.

Svojim vlastoručnim potpisom potvrđujem da sam suglasan/suglasna da Filozofski fakultet u Osijeku trajno pohrani i javno objavi ovaj moj rad u internetskoj bazi završnih i diplomskih radova knjižnice Filozofskog fakulteta u Osijeku, knjižnice Sveučilišta Josipa Jurja Strossmayera u Osijeku i Nacionalne i sveučilišne knjižnice u Zagrebu.

U Osijeku, 16. kolovoza, 2023.

Lucka Matić, 0122235752

Ime i prezime studenta, JMBAG

SAŽETAK

Sažimanje engl. *hashing* je jedan od ključnih zaštitnih mehanizama u svijetu sigurnosti na internetu kada su u pitanju osjetljivi podaci i lozinke. Primjenom funkcija sažimanja se osiguravaju podaci u bazama podataka. Cilj ovog rada je objasniti na koji način algoritmi za sažimanje funkcioniraju i gdje se sve mogu primjenjivati te koji su to algoritmi. *Message Digest* i *Secure Hashing Algorithm* su dva najčešća algoritma kada su u pitanju razni sigurnosni protokoli ili sigurnost poruka, računalna forenzika i kriptovalute. Međutim SHA i MD algoritmi se ne preporučuju za sažimanje lozinki. Trenutno se koriste tri glavna algoritma za sažimanje lozinki, a oni su Bcrypt, Scrypt i Argon2. U radu se nastoji objasniti na koji način funkcioniraju različiti algoritmi za sažimanje lozinki i koje su prednosti određenog algoritma. Nadalje, u radu se pokazuju primjeri uporabe *hash* algoritama za sažimanje lozinki kroz Node.js programski jezik, te cijeli proces kojim se mogu *hash* vrijednosti probiti od strane napadača te kako se najbolje zaštititi od takvih vrsta napada.

Ključne riječi: sažimanje, hashing, algoritmi za sažimanje, lozinke

Sadržaj

1. UVOD

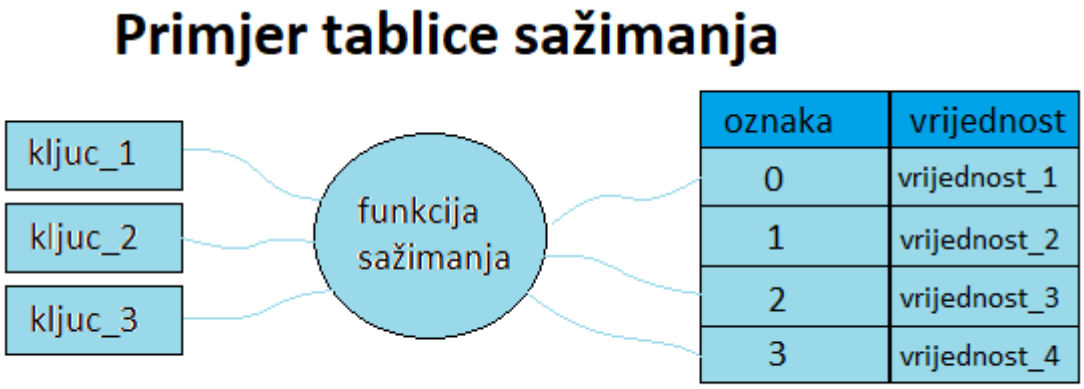
Razvitkom Interneta i tehnologije ljudi su svakodnevno izloženi digitalnom okruženju. Ljudi koriste Internet u razne svrhe od zabave do obavljanje poslova. Za većinu tih aktivnosti koje ljudi rade su potrebni korisnički računi, na primjer za korištenje društvenih mreža, email, igranje online igrica itd. Za svaki korisnički račun potrebna je i lozinka koja treba biti dovoljno jaka kako bi računi bili sigurni. Za dodatnu sigurnost koriste se funkcije za sažimanje lozinki koji pomoću matematičkih algoritama od lozinke naprave drugu vrijednost koja se naziva sažetak ili *hash* vrijednost. Sažimanje ili *hashing* se koristi svuda po Internetu i uključen je u sigurnosne protokole u razne svrhe. Koristi se za pronalaženje podataka, kriptografiju, kibernetičku sigurnost i digitalne potpise. Za razne protokole za sigurnost najviše se koriste SHA (*Secure Hashing Algorithm*) algoritmi, točnije SHA-256 algoritam, a MD (*Message Digest*) algoritmi se koriste za sigurnost poruka, računalne forenzike i kriptovalute.[1] Specifično za sigurnost lozinki najviše se ističu Bcrypt, Scrypt i Argon2 algoritmi. Razlog zašto su baš ti algoritmi dobri za zaštitu lozinki je zato što im je potrebno puno više memorije i sporiji su od ostalih algoritama upravo zbog toga kako bi napadačima trebalo puno više vremena za probijanje lozinke. Bitna stvar koju koriste ti algoritmi je *salt* što bi bila dodatna vrijednost koja se dodaje na konačnu *hash* vrijednost što još više otežava napadačima da dođu do originalne lozinke, prema dostupnim tehnologijama trenutno gotovo pa nemoguće.[2] Najpoznatiji napadi na lozinke su *brute force* napadi i napadi *rainbow* tablicama koji mogu biti bezopasni ako se koriste dovoljno dobri sigurnosni protokoli i algoritmi za zaštitu.

2. OPĆENITO O HASHINGU

Hashing je proces u kojem se neki niz znakova pretvara u drugu vrijednost odnosno u drugi niz znakova. Najčešće druga vrijednost bude kraća ili jednostavnija za pronalazak nego što je to bila izvorna vrijednost. *Hashing* se provodi pomoću matematičkih algoritama koji su najčešće jednosmjerni, to jest ne postoji mogućnost da se *hash* vrijednost ponovo pretvori u originalnu vrijednost, te najviše se implementira preko *hash* tablice preko koje odabrana vrijednost prođe

[1] Usp. TechTarget. URL: https://www.techtarget.com/searchsecurity/definition/MD5 (2023-06-25)
[2] Usp. Okta developer. URL: https://developer.okta.com/blog/2019/07/29/hashing-techniques-for-password-storage#evaluating-hashing-algorithms (2023-06-27)

kroz *hash* funkciju i smjesti se u odabrani indeks po kojemu će *hashirana* vrijednost biti dostupna.[3] Primjer *hash* tablice se može vidjeti na slici 1.



Slika 1.[4]

Upotreba *hashinga* najčešće je vezana uz indeksiranje i pronalaženje podataka, kibernetičku sigurnost, kriptografiju i za digitalne potpise.[5]

2.1. Povijest *hashinga*

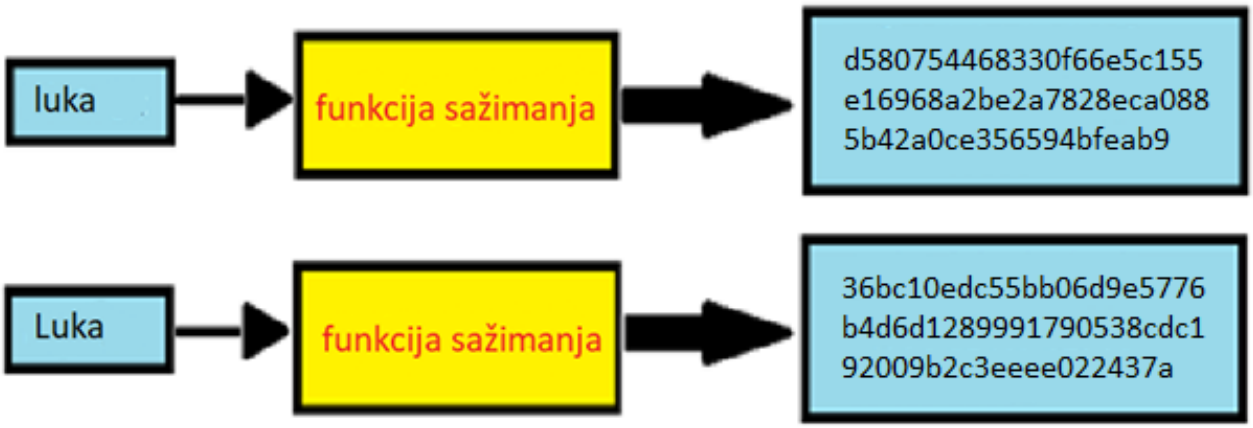
Povijest *hashinga* započela je još 1953. godine kada ju je njemački izumitelj Hans Peter Luhn prvi upotrijebio nakon što se pridružio IBM-u kao specijalist za komunikaciju, skladištenje i pronalaženje informacija u tekstu. On je osmislio algoritam za brže pronalaženje brojeva u telefonskom imeniku koji je bio poznat pod nazivom „Luhnov algoritam" iz kojeg su se kasnije razvili ostali algoritmi za *hashing* poput: MD4, MD5, DMDC, te razne verzije SHA algoritma.[6] Luhnov algoritam funkcionirao je tako da se umjesto pretraživanja svih telefonskih brojeva, kojih je bilo preko milijun, u jednoj takozvanoj „kanti" da se brojevi rasporede u numerirane „kante". Na primjer ako je broj 389-571-5115 broj bi se podijelio u parove (38, 95, 71, 51, 15), te bi se te znamenke međusobno zbrojile (12, 14, 8, 6, 6), a ako bi broj bio s dvije znamenke

[3] Usp. TechTarget. URL: https://www.techtarget.com/searchdatamanagement/definition/hashing (2023-06-19)
[4] Isto.
[5] Isto.
[6] Usp. Charmaine Gaffud San Jose, Christine. NHAF-512: New Hash Algorithm Applying Feistel Cipher Structure, 2023. URL: https://www.researchgate.net/publication/367434037_NHAF-512_New_Hash_Algorithm_Applying_Feistel_Cipher_Structure (2023-06-19)

onda bi se samo gledala zadnja znamenka tako da bi u ovom slučaju broj 389-571-5115 bio smješten u „kantu“ s oznakom 24866 što bi olakšalo pronalazak tog broja.[7]

3. SHA HASHING ALGORITMI

SHA je akronim od *secure hashing algorithm* što bi u prijevodu značilo siguran algoritam sažimanja i on je jedan od algoritama koji se koristi za *hashing* podataka. SHA pretvara određene podatke u niz znakova koji se ne može dešifrirati poput nekih metoda kao što su funkcije kompresije i bitovne operacije zato što je jednosmjeran, što bi značilo kada se upotrijebi *hashing* na podacima više se ne može podatak „probiti“. Algoritam funkcionira tako kad se jedno slovo promijeni u riječi ili ako se promijeni veličina slova da se generira potpuno drugačija *hashing* vrijednost. Primjer se može vidjeti na slici 2 gdje je korišten SHA-256 algoritam.[8]



Slika 2.[9]

Spominje se više SHA oblika, ali postoji samo četiri osnovna, a to su SHA-0, SHA-1, SHA-2 i SHA-3 koji je objavljen 2015. godine, ali za njegovo korištenje još nema potrebe. Neki od

[7] Usp. Medium. URL: https://medium.com/@dieswaytoofast/the-invention-of-the-hashing-algorithm-600933d7f845 (2023-06-19)
[8] Usp. Encryption Consulting. URL: https://www.encryptionconsulting.com/education-center/what-is-sha#intro-sha (2023-06-20)
[9] Isto.

ostalih oblika su SHA-256, SHA-512 i SHA-224, ali oni su samo verzije od SHA-2 oblika. Na primjer SHA-256 je verzija SHA-2 koja u imenu ima duljinu bitova koje može proizvesti.[10]

3.1. SHA-0

Nacionalni institut za standarde i tehnologiju (NIST) 1993. godine izdao je SHA-0 koji je bio kao federalni standard za obradu informacija. Temeljio se po načelima MD4 i MD5. Funkcionirao je tako da preuzme bilo koju informaciju do 264 bita na primjer riječ „Luka“ i pretvori ga u *hashing* vrijednost od 160 bitova koji bi izgledao ovako: „28abf0c17048d18eb767d7849ba01e5ea2b51071“.[11] SHA-0 je imao problema s kolizijama *hashing* vrijednosti da su bile gotovo iste ili čak potpuno iste. Pronalazak takvih kolizija nije jednostavno, ali kroz nekoliko napada koji su služili kako analiza algoritma otkrile su se slabosti i pogrešne funkcije. Zbog takvih nedostataka bila je potrebna revizija algoritma koji je bio promijenjen isto od strane Nacionalnog instituta za standarde i tehnologiju 1995. godine kada je SHA-0 promijenjen u SHA-1 s naprednijim algoritmom za *hashing* informacija.[12]

3.2. SHA-1

SHA-1 može sažeti poruku čija će vrijednost iznositi 160 bitova odnosno 20 bajtova. *Hashirana* vrijednost će uglavnom biti pretvorena u heksadecimalni broj koji je dugačak 40 znakova. To je određeno po federalnom standardu za obradu informacija koji je dizajnirala Nacionalna sigurnosna agencija SAD-a. SHA-1 se smatrao sigurnim sve do 2005. godine kada se otkrilo da je postao nesiguran i da je potrebno kreirati novi algoritam to jest SHA-2.[13] SHA-1 je korišten od strane velikih tvrtki poput Googlea, Microsofta i Applea sve do 2017. godine kada su prestali prihvaćati njegove *secure socket layer* (SSL) certifikate. Dok je bio u uporabi bio je korišten u razne svrhe. U kriptografiji je bio korišten kao zaštita komunikacije od vanjskih strana gdje mu je bila zadaća da se potvrdi da podaci nisu bili izmijenjeni tijekom njihovog prijenosa preko *hash* vrijednosti koju je prvenstveno generirao. Koristio se u svrhu sigurnosti digitalnih potpisa

[10] Isto.
[11] Toolkit. URL: https://toolkit.vercel.app/sha-0/ (2023-07-29)
[12] Usp. Biham, Eli; Chen, Rafi. Near-Collisions of SHA-0, 2004. URL: https://link.springer.com/chapter/10.1007/978-3-540-28628-8_18 (2023-06-20)
[13] Usp. Geeksforgeeks. URL: https://www.geeksforgeeks.org/sha-1-hash-in-java/ (2023-06-21)

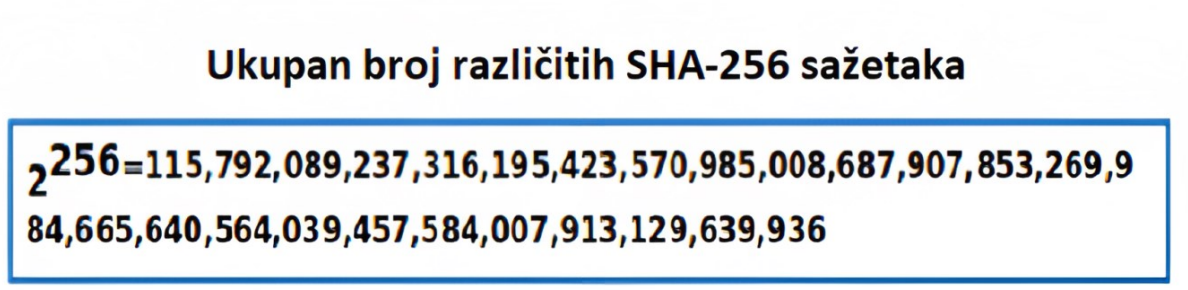
te na isti način ako je prvenstvena *hash* vrijednost koju je generirao ista to znači da je digitalni potpis valjan. Isto tako i u digitalnom forenzici, pohranjivanju lozinki i ažuriranju *softwera*, ako se prvenstveno generirana *hash* vrijednost i dalje podudara to znači da su podaci sigurni i nisu izmijenjeni.[14]

3.3. SHA-2

Secure hash algorithm 2 izdan je 2002. godine od strane NIST-a, Nacionalnog instituta za standarde i tehnologiju. Tada su objavljene tri nove verzije *hash* algoritama koje su spadale u SHA-2, a to su SHA-256, SHA-384 i SHA-512. Sljedeće godine u prosincu objavljena je i četvrta verzija SHA-2, a to je bila SHA-224.[15] SHA-2 je sve do danas najkorišteniji *hash* algoritam koja ima puno funkcija, a najviše je korištena kao sigurnosni mehanizam za zaštitu podataka. Koristi se u mnogim protokolima za sigurnost koji su rašireni po Internetu. Neki od njih su: *Transport Layer Security* (TLS), *Internet Protocol Security* (IPSec), *Pretty Good Privacy* (PGP) i *Secure/Multipurpose Internet Mail Extensions* (S/MIME). TLS se koristi za web stranice, a kada stranica završava s https to znači da je SHA-2 uključen u sigurnost te stranice. IPSec je najčešće korišten kao sigurnost veze u VPN-ovima. PGP se koristi u šifriranju e-pošte odnosno služi kao sigurnost da poruka nije vidljiva potencijalnim napadačima nego samo primatelju te poruke, a S/MIME je također uključen u enkripciju e-pošte. Kao što je već ranije rečeno u radu, SHA verzije u nazivu imaju brojeve poput 224, 256 i 384, a to je broj bitova koje mogu proizvesti kada generiraju *hash* vrijednost. Osim toga verzije se još razlikuju i po veličini bloka, veličini riječi kroz koje se obrađuje *hash* i kroz koliko krugova prolaze dok se ne generira konačni *hash*. Broj bitova je bitan zbog toga da se ne dogodi kolizija između *hash* vrijednosti. Ako je broj bitova veći u *hash* funkciji to znači da je manja vjerojatnost za koliziju to jest manja je vjerojatnost da dvije različite poruke imaju jednaku *hash* vrijednost. Veličina bloka je također potrebna za sigurnost *hashinga*. Ako je blok manji onda će trebati više blokova za obradu unosa podataka, najsigurnijim se smatraju blokovi veličine od 512 i 1024 bita. Duljina riječi ovisi o veličini bloka, a veličina bloka je uvijek šesnaest puta veća od duljine riječi. Na primjer algoritam koji ima veličinu bloka 512 bita riječ će biti duljine 32 bita, a kod bloka veličine 1024 bita riječ će biti duljine 64 bita. Također bitna stvar u SHA-2

[14] Isto.
[15] Usp. Preneel, Bart. The First 30 Years of Cryptographic Hash Functions and the NIST SHA-3 Competition, 2010. URL: https://www.esat.kuleuven.be/cosic/publications/article-1532.pdf (2023-06-21)

algoritmu su brojevi krugova kroz koje prolaze riječi. Svaki krug kroz koji prođe poruka znači da se riječ dodatno izmiješa. To znači da što više krugova ima u algoritmu to je teže da se dogodi kolizija između *hash* vrijednosti i sigurnija je.[16] Jednim od najsigurnijih *hashing* algoritama smatra se SHA-256. SHA-256 ima sve komponente koje bi siguran *hashing* algoritam trebao imati, a to je na primjer da je jednosmjeran algoritam, šanse za koliziju su male ili ih nema, ima efekt lavine odnosno ako se promjeni samo jedna znamenka u poruci koja treba biti *hashirana*, vrijednost će se potpuno promijeniti i što je najbitnije još nikad nije bio probijen odnosno hakiran. Na slici 3. može se vidjeti koliko SHA-256 ima različitih *hash* vrijednosti koje može generirati, a šanse za koliziju su jedan naprama 2^256.[17]



Slika 3.[18]

4. MESSAGE DIGEST HASHING ALGORITMI

Message digest to jest sažetak poruke je funkcija koja tvori *hashing* generiranjem niza znamenki kroz jednosmjerni algoritam. Algoritmi se koriste za sigurnost podataka ili medija kako bi se autor osigurao da nisu pravljene nikakve izmjene i promjene nekog dijela podatka. Generirana *hash* vrijednost predstavlja datoteke koje sadrže zaštićena djela, svaka datoteka ima svoju *hash* vrijednost i ako je ona promijenjena to znači da je sadržaj datoteke promijenjen namjerno ili slučajno. To pomaže autoru da uoči gdje je učinjena izmjena i potencijalne osobe koje su tu izmjenu izvršile. Sažetak poruke čine algoritamski brojevi. *Message digest* algoritmi

[16] Usp. Comparitech. URL: https://www.comparitech.com/blog/information-security/what-is-sha-2-how-does-it-work/ (2023-06-22)
[17] Usp. Code signing store. URL: https://codesigningstore.com/what-is-the-most-secure-hashing-algorithm (2023-06-23)
[18] Isto.

umjesto što mogu otkriti promjene u datoteci također mogu i otkriti to jest locirati duplikate datoteka kao i upozoravanje korisnika ako je preuzeo iste datoteke i porijeklo dupliciranih preuzimanja. Za probleme kada je u pitanju kolizija to jest kada je *hash* vrijednost ista za dva različita podatka tu se *message digest* više ne koristi zato što je po tome pitanju nepouzdan. Sažeci poruka su šifrirani privatnim ključevima koji stvaraju digitalni potpis što osigurava da odgovarajući korisnik pristupa zaštićenim informacijama.[19]

4.1. Message digest 2

Message digest 2 funkcija je razvijena od strane Ronalda Rivesta 1989. godine. Funkcija generira 128-bitnu vrijednost od početne poruke. Korištena je i optimizirana za 8-bitna računala, a primarno je stvorena za aplikacije digitalnog potpisa koje su zahtijevale osigurane datoteke potpisane privatnim ključem. MD2 se još uvijek koristi u infrastrukturi javnih ključeva, ali to je rijedak slučaj zato što je funkcija zastarjela i potrebno joj je puno vremena za računanje i nije sigurna za korištenje. Proces generiranja *hash* vrijednosti u MD2 je da se najprije dodaju bajtovi kako bi vrijednost bila veličine 128 bita što bi se slagalo s tim što je veličina bloka u kojem se obrađuje poruka 16 bajtova, a taj proces se naziva *padding*. Nakon *paddinga* dodaje se kontrolni broj, inicijalizacija, obrada poruke u blokovima te generiranje konačne vrijednosti. Prednost MD2 je to što je jednostavan, ali nedostatci su to što je spor zato što je namijenjen za 8-bitna računala i to što može procuriti informaciju o ključevima u slučaju napada.[20]

4.2. Message digest 4

Message digest 4 generira bilo koji podatak u 128-bitnu vrijednost koja je otisak prsta ili sažeta poruka s ciljem da ne generira identičnu vrijednost dva puta. MD4 algoritam se može primijeniti u aplikacijama digitalnog potpisa što bi značilo da se datoteka može sigurno komprimirati pomoću MD4 prije nego što se osigura javnim ključem. U odnosu na MD2, MD4 je napravljen da radi na 32-bitnim strojevima što mu daje veću brzinu koja može biti oko 1.450.000 bajtova u sekundi što bi iznosilo 1,45 megabajta po sekundi. Algoritam je napravljen da ne zahtjeva

[19] Usp. Techopedia. URL: https://www.techopedia.com/definition/4024/message-digest (2023-06-23)
[20] Usp. Techopedia. URL: https://www.techopedia.com/definition/31699/message-digest-2-md2 (2023-06-23)

velike supstitucijske tablice i da se može kompaktno kodirati. Proces generiranja *hash* vrijednosti je sličan kao i u MD2 algoritmu. Najprije se radi *padding* od kojeg dobivamo vrijednost dužine 448 bitova, zatim se tom rezultatu dodaje 64-bitni prikaz poruke koji se sastoji od dvije 32-bitne riječi i još jedne dodatne riječi, nakon toga dolazi inicijalizacija pa obrađivanje poruke u blokovima koji se također sastoji od 16 bajtova i na kraju dobijemo konačnu vrijednost to jest sažetak poruke.[21]

4.3. Message digest 5

Message digest 5 je kreiran 1991. godine i on je nastao kao proširenje za *message digest* 4. Isto kao i MD4, MD5 uzima poruku bilo koje dužine i od nje napravi 128-bitnu sažetu poruku ili otisak prsta. MD5 dizajniran je za 32-bitne mašine, poprilično je brz i ne zahtjeva velike supstitucijske tablice te se može kompaktno kodirati. Razlika u MD4 i MD5 je ta što je MD4 nešto brži od MD5, ali sigurnost mu je bila upitna kada su u pitanju kriptoanalitički napadi. Proces izrade sažetka poruke je identičan kao i kod MD4. Prvo se radi *padding* to jest dodavanje bajtova kako bi vrijednost bila veličine 448 bitova zatim slijedi dodavanje 64-bitnog prikaza poruke kroz dvije 32-bitne riječi i jedne dodatne riječi, nakon toga slijedi inicijalizacija, procesuiranje vrijednosti kroz blokove i na kraju dobivamo output to jest konačni sažetak poruke.[22] *Message digest* 5 kreiran je za provjeravanje autentičnosti kriptografske poruke na Internetu, ali se više ne smatra pouzdanim zbog toga što može doći do kolizije odnosno može doći do toga da dvije datoteke imaju istu *hash* vrijednost. Zbog toga se MD5 koristi samo za sigurnost poruka, lozinki, računalne forenzike i kriptovalute. Izvršeno je nekoliko napada na MD5 2011. godine i neki su u manje od minute uspjeli generirati koliziju i zbog toga se MD5 ne treba više koristiti u sigurnosnim protokolima, a pogotovo u onima koji se služe za digitalne potpise. Idealna zamjena za MD5 kada je u pitanju sigurnost na Internetu i sigurnost digitalnih potpisa je SHA-2.[23] Ostale stvari u kojima se koristi MD5 je provjera integriteta datoteka to jest provjera njene autentičnosti, enkripcija podataka i provjera lozinki. Prednosti su to što je jednostavan i brz, može generirati sigurnu lozinku od 16 bajtova i potrebno mu je malo memorije, a neki od nedostataka su ti što može generirati iste *hash* vrijednosti što znači da

[21] Usp. Rivest, L. Ronald. The MD4 Message Digest Algorithm, 1991. URL: https://link.springer.com/content/pdf/10.1007/3-540-38424-3_22.pdf (2023-06-24)
[22] Usp. Rivest, L. Ronald. The MD5 Message-Digest Algorithm, 1992. URL: https://www.ietf.org/rfc/rfc1321.txt (2023-06-25)
[23] Usp. TechTarget. URL: https://www.techtarget.com/searchsecurity/definition/MD5 (2023-06-25)

dolazi do kolizije, znatno je slabiji od SHA1 u pitanju sigurnosti i nema ni simetričan ni asimetričan algoritam.[24]

4.4. Razlog sažimanja lozinke

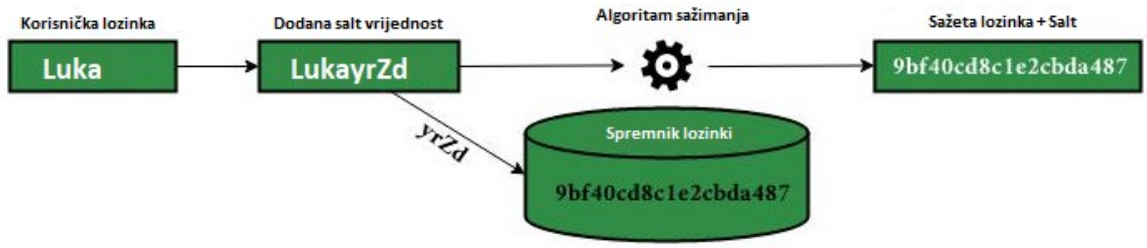
Sigurnost lozinki je od iznimne važnosti i jako je bitno da se korisnicima određenih stranica pruži najbolja sigurnost za njihove korisničke račune. U slučaju da napadač dobije pristup bazi podataka gdje su spremljene lozinke, korisnički računi bi bili u opasnosti i postojala bi velika mogućnost da će napadač imati pristup osobnih podacima. Zato je od velike važnosti da se koriste algoritmi za sažimanje lozinki kako bi bile sigurno pohranjene i kako napadač ne bi imao uvid u tzv. *plain text* lozinke.[25] Iako su lozinke pohranjene kao sažete vrijednosti i dalje postoji mogućnost od probijanja lozinke ako se poklope dvije hash vrijednosti i zbog toga se koristi *salt* za dodatnu sigurnost. *Salt* je dodana vrijednost na *hash* vrijednost prije nego što je generirana i zbog toga napadač ne može otkriti koja je bila originalna lozinka zbog dodatnih znakova koji su dodani.[26] Osim toga što su lozinke zaštićene sažetom vrijednošću i *saltom* svejedno bi korisnici trebali pažljivo postavljati lozinke zato što ako je lozinka kompliciranija to će napadačima biti teže razaznati lozinku. Neke od preporuka NIST-a (National Institute of Standards and Technology) su: duljina (broj znakova) je bitna, ne kompleksnost, ne dozvoliti slabe lozinke (riječi iz rječnika, ponavljanje znakova, naziv usluge, korisnika, usluge), lozinka ne smije sadržavati korisničko ime, trebaju se koristiti različite lozinke na različitim računima i da se lozinka ne bi trebala lako pogoditi.[27]

5. ALGORITMI ZA HASHING LOZINKI

Kao što je već navedeno, najbolji algoritmi za *hashing* su SHA algoritmi i MD algoritmi, ali za *hashing* lozinki nisu najbolji. Ono što je prednost kod SHA i MD algoritama je njihova brzina, ali kada je u pitanju *hashing* lozinki potrebno je suprotno, algoritmi ne bi trebali biti brzi zbog toga što to čini napade na lozinke puno skupljim za napadače i proces je dugotrajniji. Kada bi

[24] Usp. Geeksforgeeks. URL: https://www.geeksforgeeks.org/what-is-the-md5-algorithm/ (2023-06-26)
[25] Usp. Culttt. URL: https://culttt.com/2013/01/21/why-do-you-need-to-salt-and-hash-passwords (2023-07-18)
[26] Usp. StackExchange. URL: https://security.stackexchange.com/questions/36833/why-should-i-hash-passwords (2023-07-18)
[27] Usp. Jakopec, Tomislav. Lozinke naše svagdašnje. Cyber Security Confrence 2021. Sveučilište J. J. Strossmayer, Fakultet elektrotehnike, računarstva i informacijskih tehnologija. Osijek, 15.10.2021. [Predavanje]

algoritmi za *hashing* lozinki bili brzi poput SHA i MD onda bi napadači mogli koristiti više računalnih komponenata poput procesora i grafičke kartice i tada bi mogli izvršiti više pokušaja da pogode lozinku u kratkom vremenskom razdoblju. Još jedna prednost koju imaju algoritmi za *hashing* lozinki je ta što koriste *salt* u generiranju vrijednosti što dodatno otežava napadaču da otkrije lozinku. *Salt* puno pomaže kada korisnik koristi jednostavne lozinke. Kada se ne bi koristio algoritam koji ima *salt*, napadač bi mogao pronaći podudaranje lozinki i uspio bi probiti lozinku, ali pomoću salta napadač ne može imati podudaranje lozinki upravo zbog te dodane vrijednosti na lozinku. Najbolja tri algoritma za *hashing* lozinki su Bcrypt, Scrypt i Argon2. Ti algoritmi koriste *salt* i još uz to otežavaju napade time što je potrebno koristiti puno više memorije i puno više snage procesora za izračunavanje *hash* vrijednosti što napadačima stvara veći trošak i vremenski duže treba da se izračuna *hash* vrijednost.[28]



Slika 4.[29]

5.1. Bcrypt hashing algoritam

Bcrypt su kreirali Niels Provos i David Mazières 1999. godine po uzoru na *Blowfish* blok šifru. *Blowfish* blok šifra je značajna po svojoj skupoj fazi postavljanja ključa. Brza je i ima simetričan algoritam. Bcrypt je algoritam koji koristi *salt* i na taj način se štiti od napada, također može povećati broj iteracija što bi značilo da je tada sporiji što bi usporilo otkrivanje *hash* vrijednosti čak i ako se poveća računalna snaga. *Hash* vrijednost koju *Bcrypt* generira je dugačka 192 bita koja je predstavljena kao niz od 31 znaka i još uz nju *salt* dugačak 128 bitova odnosno niz od 22 znaka. Zbog svoje sigurnosti Bcrypt se smatra standardom za *hashing* lozinki. Jedina slabost kod Bcrypta su FPGA procesori preko kojih napadi na algoritam mogu biti učinkoviti, ali ako

[28] Usp. Okta developer. URL: https://developer.okta.com/blog/2019/07/29/hashing-techniques-for-password-storage#evaluating-hashing-algorithms (2023-06-27)
[29] Usp. GeeksforGeeks. URL: https://www.geeksforgeeks.org/what-is-salted-password-hashing/ (2023-07-18)

se kreira dovoljno jaka lozinka onda je i tada nemoguće dešifrirati lozinku.[30] Bit će prikazano kako *hashirati* i dodati *salt* na lozinku koristeći Bcrypt u Node.js.

Kako bi se mogao koristiti Bcrypt treba se uključiti modul s linijom koda:

```
const bcrypt = require ('bcrypt');.
```

Nakon toga odabire se broj krugova za salt. Što je veći broj to će salt biti sigurniji od napada, ali ako se odabere preveliki broj onda će trebati više vremena da algoritam generira *salt*. Idealna vrijednost za *salt* bi bila 10 tako da se piše sljedeća linija koda:

```
const saltRounds = 10;.
```

Nakon što je upisan broj krugova za *salt* sljedeće što se treba upisati je lozinka koju korisnik želi *hashirati* s linijom koda

```
var password = "Luka".
```

Nakon upisane lozinke generira se *salt* s linijom koda:

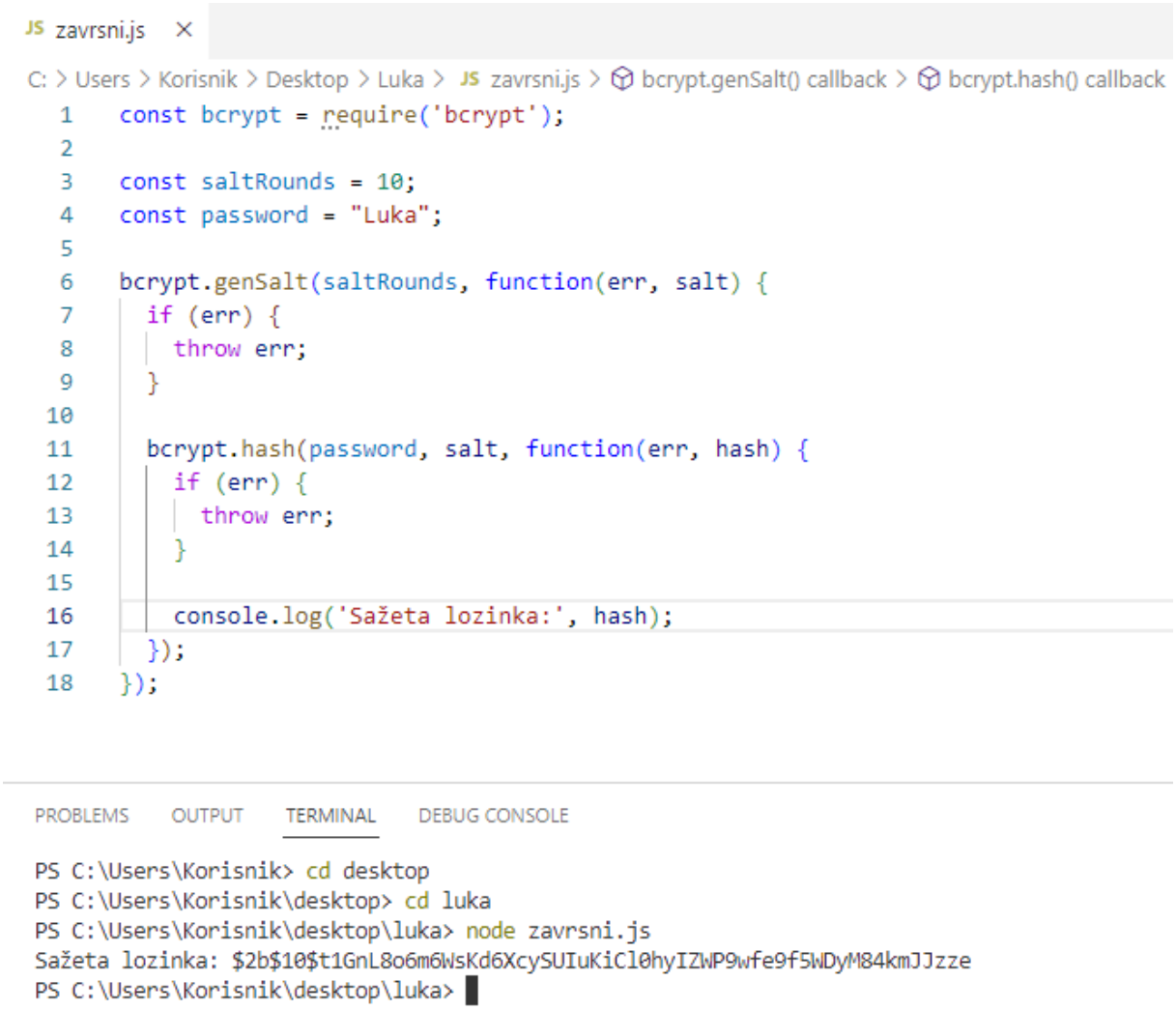
```
bcrypt.genSalt(saltRounds, function(err, salt) { // returns salt
});.
```

Na kraju se dodaje *hash* funkcija unutar getSalt funkcije i tada se može ispisati konačna *hash* vrijednost

```
bcrypt.genSalt(saltRounds, function(err, salt) {
bcrypt.hash(password, salt, function(err, hash) { // returns
hash console.log(hash); }); });
```
[31]

Nakon što se preko terminala došlo do putanje gdje je spremljena datoteka upiše se node te ime datoteke, u ovome slučaju node zavrsni.js, te tada se ispiše *hash* vrijednost koja je u ovome slučaju:
$2b$10$dsPJmsCrH.pT0XQ0U.Zq6..PZOK35KMXLxi6gbFBvLkktOC0GWnpu

[30] Usp. Sriramya, P.; Karthika, R.A. Providing password security by salted password hashing using Bcrypt algorithm, 2015. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=9d82620f0866ec773b12c30f90d70b74661f972f (2023-06-28)
[31] Usp. Hey Node. URL: https://heynode.com/blog/2020-04/salt-and-hash-passwords-bcrypt/ (2023-06-29)



Slika 5.

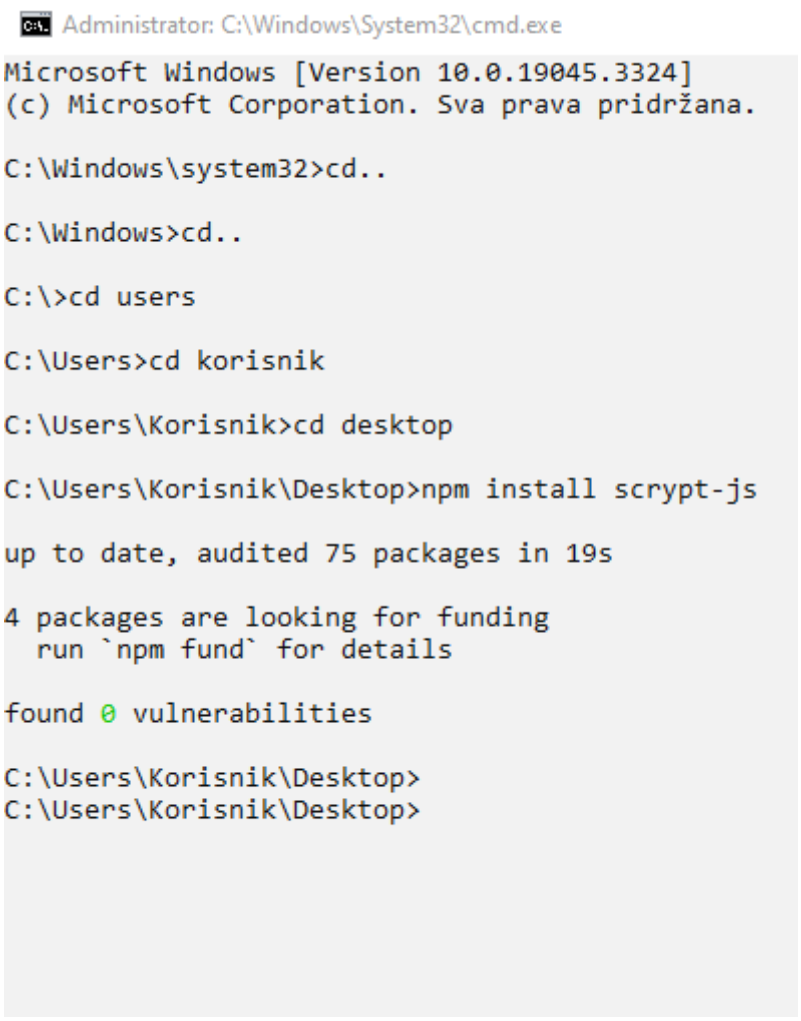
5.2. Scrypt hashing algoritam

Scrypt je kreirao Colin Percival 2009. godine, a predstavio ga je u radu „"*Stronger Key Derivation Via Sequential Memory-Hard Functions*." Kreiran je kao algoritam koji troši puno memorije kako bi poboljšao mrežnu sigurnost od napadača koristeći prilagođene hardvere, a kasnije se počeo koristiti za kriptovalute poput Litecoina, Dogecoina i Einsteiniuma.[32] Za Scrypt je potrebna velika količina memorije i uporaba procesora, a zbog generiranja brojeva u cijelom procesu koji se pohranjuju u memoriju s izravnim pristupom (RAM) zauzima veliku količinu memorije. Scrypt je jako skup algoritam zato što kada se generira element tijekom *hashiranja* zahtijeva više memorije i računanja i zbog toga je vrlo siguran od napadača zbog

[32] Usp. Komodo. URL: https://komodoplatform.com/en/academy/scrypt-algorithm/ (2023-06-29)

nedostatka resursa i memorije.[33] Jedan od načina da bi se *hashiralo* u Scryptu treba se preko Naredbenog retka ili *Visual Studio Codea* instalirati *Scrypt library*, u ovome slučaju to će biti učinjeno preko JavaScripta. Najprije se odabere putanja gdje korisnik želi instalirati *Scrypt library* zatim se upisuje linija koda:

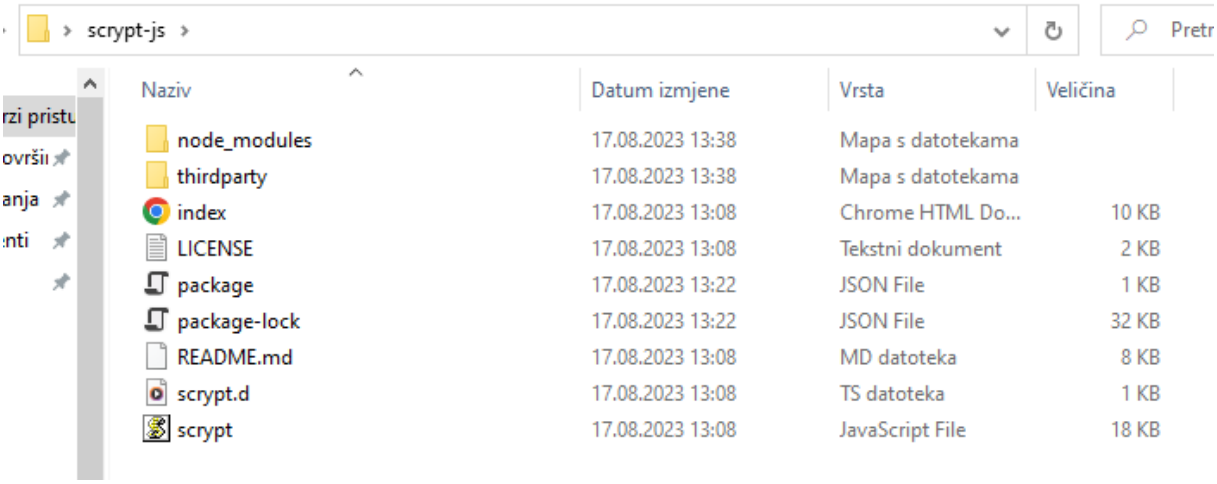
```
npm install scrypt-js
```



Slika 6.

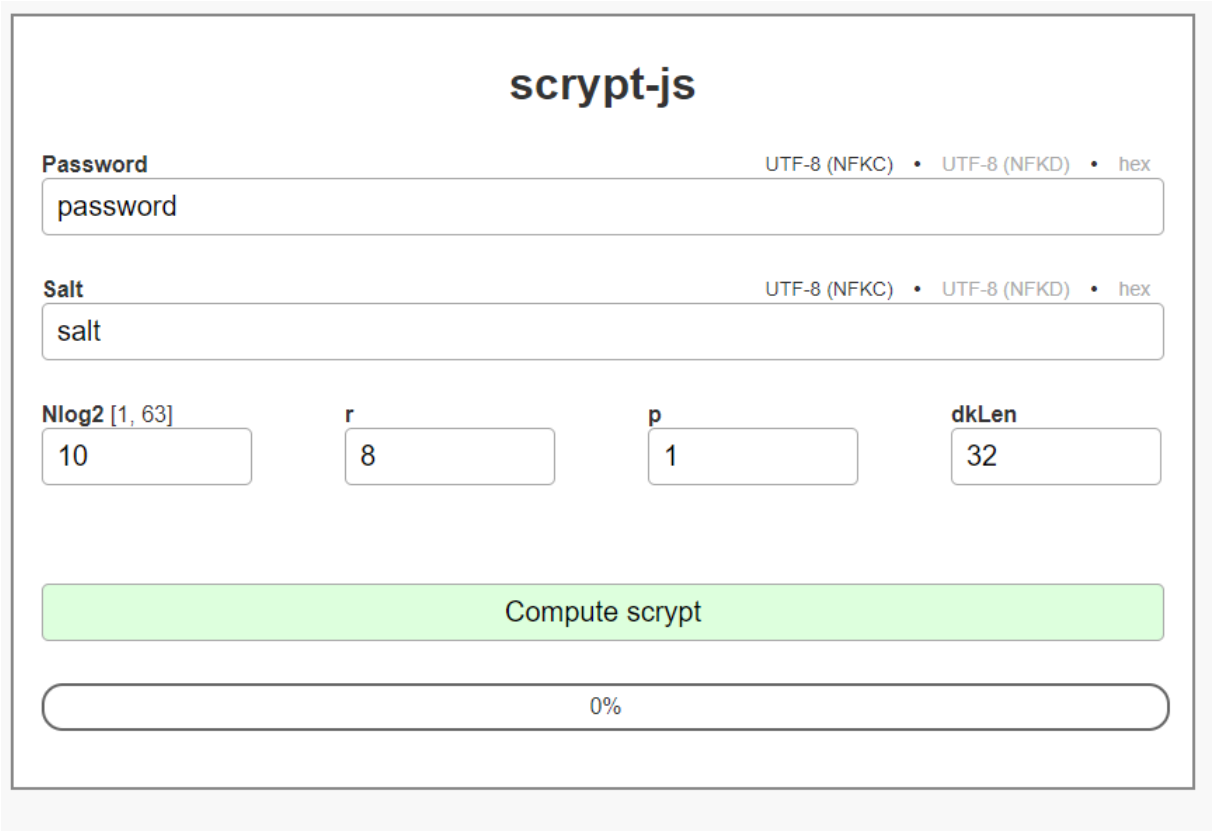
Nakon upisane linije koda na računalo se instaliraju sve potrebne datoteke pomoću kojih se može sažimati lozinka u Scryptu.

[33] Usp. Ertaul, Levent; Kaur, Manpreet; Arun Kumar R Gudise, Venketa. Implementation and Performance Analysis of PBKDF2, Bcrypt, Scrypt Algorithms, 2016. URL: http://borg.csueastbay.edu/~lertaul/PBKDFBCRYPTCAMREADYICWN16.pdf (2023-06-29)



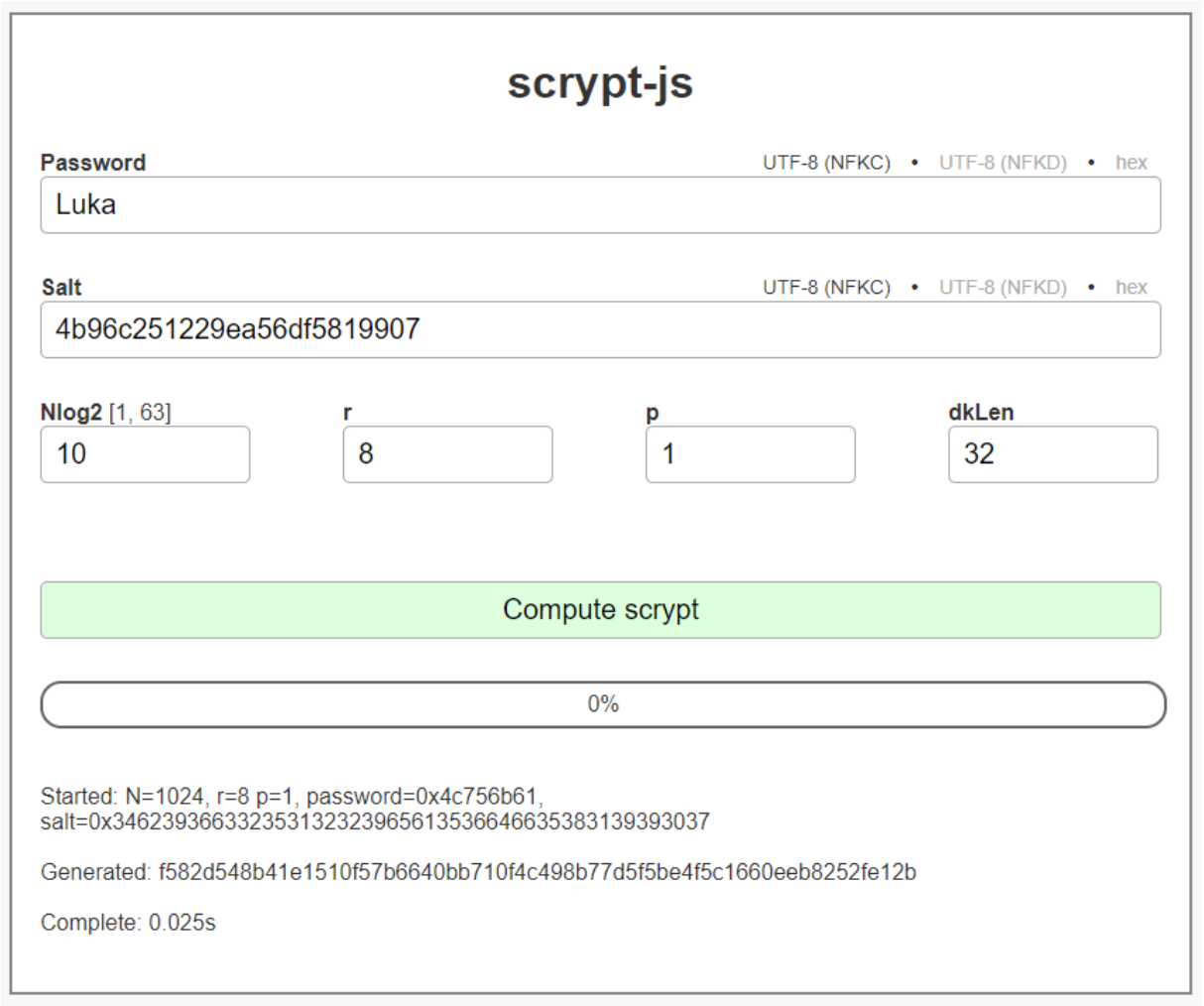
Slika 7.

Mjesto na kojemu se može sažeti lozinka je da se uđe na index.html nakon čega će se otvoriti sljedeći prozor:



Slika 8.

Potom se upisuje lozinka od koje korisnik želi dobiti sažetu vrijednost, *salt*, te ako korisnik želi promijeniti parametre sažimanja.

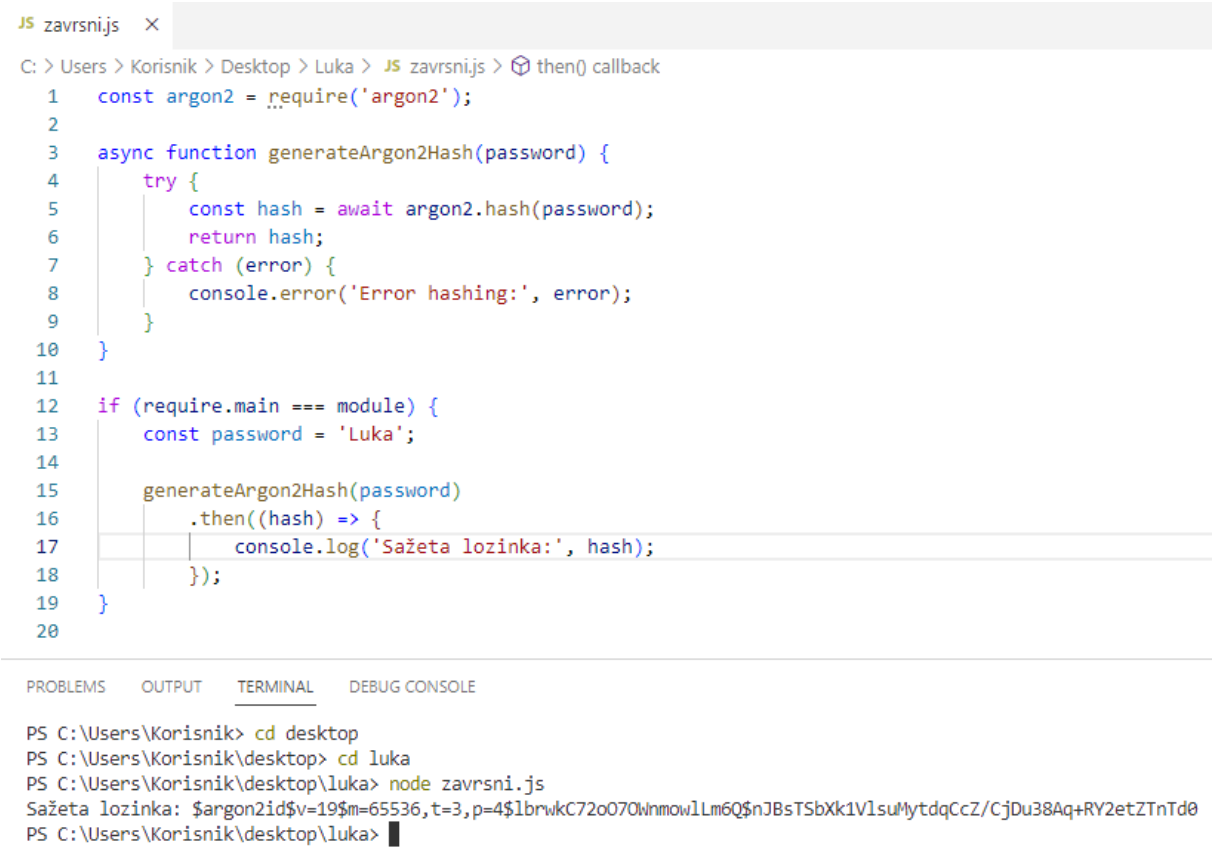


Slika 9.

5.3. Argon2 hashing algoritam

Argon2 je algoritam za *hashiranje* kojemu je jedina zadaća enkripcija lozinki. Algoritam je jedan od najboljih za korištenje zbog njegove sigurnosti koji pruža obranu od svih vrsta napada nebitno je li to GPU napad ili neki drugi. Osvojio je nagradu za najbolji algoritam za *hashiranje* lozinki u srpnju 2015. godine. Argon2 opravdava prvo mjesto na natjecanju zbog njegove neupitne sigurnosti i prilagođene brzine. Postoje tri parametra pri generiranju lozinki s Argon2 algoritmom, a to su: vrijeme, memorija i niti. Kako su u pitanju vrste Argona2 to su: Argon2i, Argon2d i Argon2id. Argon2i služi prvenstveno za obranu od napada s bočnih kanala, Argon2d

služi za zaštitu od strane GPU napada, a Argon2id je hibrid dva Argona2.[34] *Hashiranje* lozinki s Argon2 biti će prikazan u Node.js.



Slika 10.

Jedino što se zapisuje u bazu podataka je:

`$lbrwkC72oO7OWnmowlLm6Q$nJBsTSbXk1VlsuMytdqCcZ/CjDu38Aq+RY2etZTnTd0`

kako ne bi napadači znali s kojim je algoritmom *hashirana* lozinka i kako im se ne bi olakšalo napad na nju.

6. PROBIJANJE HASHINGA

Jedan od najpoznatijih načina za probijanje *hashinga* je napad *rainbow* tablicama. *Rainbow* tablica je baza podataka koja se koristi kao rječnik lozinki i njihovih odgovarajućih *hash*

[34] Usp. Boldare. URL: https://www.boldare.com/blog/how-to-improve-user-password-security-with-argon2/ (2023-06-30)

vrijednosti pomoću kojeg se može otkriti koja lozinka odgovara kojoj *hash* vrijednosti. *Rainbow* tablica radi na taj način da brzo i učinkovito provodi kriptoanalizu. Način na koji funkcionira je da se prvo kreira tablica tako da se napiše neka lozinka koja se često koristi i naziv algoritma kojim se *hash* vrijednost generirala, a piše se kroz sljedeće linije koda koje su u PHP programskom jeziku:

```
$string = 'Luka';

$hash = md5($string);

echo $hash;
```

Zatim se dobije hash vrijednost:

`25d55ad283aa400af464c76d713c07ad`.

Zatim se uzme prvih osam znakova iz dobivenog *hasha* i opet se napišu linije koda:

```
$string = 'Luka';

$hash = md5($string);

echo $hash;
```

Dobije se hash vrijednost:

`5c41c6b3958e798662d8853ece970f70`.

Taj proces se ponavlja sve dok se ne dobije dovoljno *hash* vrijednosti u *output chainu*. Kada se dobije dovoljno vrijednosti one se spremaju u tablicu. Nakon što ima dovoljno vrijednosti u tablici sljedeći korak je probijanje lozinke. Počinje se s *hashiranom* vrijednošću odnosno lozinkom tako da provjeravamo je li postoji u bazi podataka. Ako postoji tada se ide na početak *chaina* i *hashira* se sve dok ne dođe do podudaranja. Prednost napada *rainbow* tablicama je ta što se ne mora znati točna lozinka kako bi se probila, nego se mora podudarati *hash* vrijednosti, a nedostatak je taj što tablice zauzimaju puno prostora na uređaju. Najbolja obrana od napada *rainbow* tablicama je dodavanjem salta na *hashing* vrijednosti.[35] Još jedan od poznatih napada je *brute force attack* koji funkcionira tako da napadač upisujemo lozinku sve dok ne pogodi točnu lozinku. Postoji više načina za takve vrste napada. Jedan način je da se upisujemo naziv korisničkog imena kao lozinke, drugi način je da se koriste osnovne lozinke poput „12345678“,

[35] Usp. Geeksforgeeks. URL: https://www.geeksforgeeks.org/understanding-rainbow-table-attack/ (2023-07-01)

a treći način je da se upisuju sve riječi iz rječnika sve dok se ne pogodi lozinka. Neki od načina za obranu od takvih vrsta napada su: povećanje kompleksnosti lozinke, ograničenje neuspjelih pokušaja prijave, korištenje CAPTCHE, uvođenje dvofaktorske provjere autentičnosti i šifriranje i *hashiranje* lozinki.[36] Pretvaranje *hash* vrijednosti u originalnu vrijednosti lozinke tj. u *plain text* može gotovo svatko učiniti, takvi alati su dostupni diljem Interneta i postoje za neke slabije algoritam poput MD i SHA algoritama tako da i ako napadač dođe do *hash* vrijednosti lozinke može lako otkriti originalnu lozinku korisnika.[37]

7. BUDUĆNOST ALGORITAMA ZA SAŽIMANJE LOZINKI

Budućnost algoritama za sažimanje lozinki je zasad sigurna. Tri algoritma koja se najčešće koriste: Bcrypt, Scrypt i Argon2 su osigurali da budućnost bude svjetla kada je u pitanju *hashiranje* lozinki. Svaki od ta tri algoritma ima nešto u čemu je najbolji i što je najbitnije koriste se *saltom* što dodatno otežava probijanje *hash* vrijednosti. Još od 2015. godine otkad je Argon2 pobijedio na natjecanju za najbolji *hash* algoritam nije se ništa promijenilo i Argon2 je ostao na vrhu kao najbolji i najsigurniji algoritam i vjerojatno će tako ostati u skorijoj budućnosti zato što se zasad ne spominje neki novi algoritam koji bi ga mogao zamijeniti. Iz dana u dan se algoritmi poboljšavaju i unapređuju usporedno kako se i tehnologija razvija i sve dok je tako algoritmi za sažimanje lozinki će biti sigurni od napada svih vrsta.[38]

[36] Usp. TechTarget. URL: https://www.techtarget.com/searchsecurity/definition/brute-force-cracking (2023-07-02)
[37] MD5 Descryption. https://www.md5online.org/md5-decrypt.html (2023-08-04)
[38] Usp. Medium. URL: https://medium.com/@rauljordan/the-state-of-hashing-algorithms-the-why-the-how-and-the-future-b21d5c0440de (2023-08-06)

8. ZAKLJUČAK

Hashing je jednosmjerni proces u kojem se neki niz znakova pretvara u drugu vrijednost odnosno u drugi niz znakova kroz uporabu matematičkih algoritama. Područja u kojima se *hashing* najčešće koristi su kriptografija, indeksiranje i pronalaženje podataka, kibernetička sigurnost i za digitalne potpise. Hans Peter Luhn je 1953. godine prvi upotrijebio *hashing* kada je za lakše pronalaženje brojeva u telefonskom imeniku stvorio algoritam pod nazivom „Luhnov algoritam". Od toga trenutka *hashing* algoritmi su se unapređivali i njihova uporaba se širila na više područja. Nacionalni institut za standarde i tehnologiju (NIST) 1993. godine izdao je SHA-0 koji je postao standard za sigurnost, ali nakon nekog vremena zbog njegove nesigurnosti kreirana je njegova druga verzija pod nazivom SHA-1 koji je bio puno sigurniji i napredniji. Bio je korišten u razne svrhe, a najviše u kriptografiji u sigurnosti prijenosa poruke putem komunikacijskih kanala. Do 2005. je smatran sigurnim, a onda je kreiran SHA-2 koji je sve do danas najsigurniji *hash* algoritam korišten prvenstveno kao sigurnosni mehanizam za zaštitu podataka. Primijenjen je u raznim protokolima za sigurnost poput Transport Layer *Security* (TLS), *Internet Protocol Security* (IPSec*), Pretty Good Privacy* (PGP) i *Secure/Multipurpose Internet Mail Extensions* (S/MIME). MD algoritmi koriste se za sigurnost podataka ili medija kako bi se osiguralo da autor nije napravio nikakve izmjene ili dodatke u bilo kojem dijelu podataka. Također može locirati duplikate datoteka te upozoriti ako je preuzeta iste datoteka i porijeklo dupliciranih preuzimanja. Kada su u pitanju algoritmi za *hashing* lozinki relevantni predstavnici su Bcrypt, Scrypt i Argon2 koji je osvojio nagradu za najbolji algoritam 2015. godine. Sva tri algoritma su pouzdana za korištenje i sva tri algoritma imaju svoje specijalnosti. Bcrypt može povećati broj iteracija što ga čini sporiji što bi također usporilo otkrivanje *hash* vrijednosti čak i pri povećavanju računalne snage. Scrypt troši puno memorije što predstavlja problem za napadače zato što je potrebno puno vremena da se izračuna *hash* vrijednost i zbog nedostatka resursa. Argon2 pruža obranu od GPU i ostalih vrsta napada i vrlo je jednostavan za korištenje. Kada je u pitanju sigurnost od napada poput *bruce force* napada i napada *rainbow* tablicama najsigurniji način da se obrani je uporaba *salt* vrijednosti koja se dodaje na *hash* vrijednost koju je proizveo algoritam. Uz *salt* za dodatnu sigurnost lozinke preporučljivo je i povećanje kompleksnosti lozinke, ograničenje neuspjelih pokušaja prijave, korištenje CAPTCHE, uvođenje dvofaktorske provjere autentičnosti i šifriranje i *hashiranje* lozinki. Trenutno se ne spominju novi algoritmi za sažimanje lozinki zbog velike sigurnosti već postojećih algoritama, ali može se reći sve dok se razvijaju i unapređuju algoritmi za sažimanje lozinki u skladu s razvitkom tehnologije sigurnost naših lozinki bit će osigurana.

9. LITERATURA

1. Biham, Eli; Chen, Rafi. Near-Collisions of SHA-0, 2004. URL: https://link.springer.com/chapter/10.1007/978-3-540-28628-8_18 (2023-06-20)

2. Boldare. URL: https://www.boldare.com/blog/how-to-improve-user-password-security-with-argon2/ (2023-06-30)

3. Charmaine Gaffud San Jose, Christine. NHAF-512: New Hash Algorithm Applying Feistel Cipher Structure, 2023. URL: https://www.researchgate.net/publication/367434037_NHAF-512_New_Hash_Algorithm_Applying_Feistel_Cipher_Structure (2023-06-19)

4. Code signing store. URL: https://codesigningstore.com/what-is-the-most-secure-hashing-algorithm (2023-06-23)

5. Comparitech. URL: https://www.comparitech.com/blog/information-security/what-is-sha-2-how-does-it-work/ (2023-06-22)

6. Culttt. URL: https://culttt.com/2013/01/21/why-do-you-need-to-salt-and-hash-passwords (2023-07-18)

7. Encryption Consulting. URL: https://www.encryptionconsulting.com/education-center/what-is-sha#intro-sha (2023-06-20)

8. Ertaul, Levent; Kaur, Manpreet; Arun Kumar R Gudise, Venketa. Implementation and Performance Analysis of PBKDF2, Bcrypt, Scrypt Algorithms, 2016. URL: http://borg.csueastbay.edu/~lertaul/PBKDFBCRYPTCAMREADYICWN16.pdf (2023-06-29)

9. Geeksforgeeks. URL: https://www.geeksforgeeks.org/sha-1-hash-in-java/ (2023-06-21)

10. GeeksforGeeks. URL: https://www.geeksforgeeks.org/what-is-salted-password-hashing/ (2023-07-18)

11. Geeksforgeeks. URL: https://www.geeksforgeeks.org/what-is-the-md5-algorithm/ (2023-06-26)

12. Geeksforgeeks. URL: https://www.geeksforgeeks.org/understanding-rainbow-table-attack/ (2023-07-01)

13. Hey Node. URL: https://heynode.com/blog/2020-04/salt-and-hash-passwords-bcrypt/ (2023-06-29)

14. Jakopec, Tomislav. Lozinke naše svagdašnje. Cyber Security Confrence 2021. Sveučilište J. J. Strossmayer, Fakultet elektrotehnike, računarstva i informacijskih tehnologija. Osijek, 15.10.2021. [Predavanje]

15. Komodo. URL: https://komodoplatform.com/en/academy/scrypt-algorithm/ (2023-06-29)

16. MD5 Descryption. https://www.md5online.org/md5-decrypt.html (2023-08-04)

17. Medium. URL: https://medium.com/@dieswaytoofast/the-invention-of-the-hashing-algorithm-600933d7f845 (2023-06-19)

18. Medium. URL: https://medium.com/@rauljordan/the-state-of-hashing-algorithms-the-why-the-how-and-the-future-b21d5c0440de (2023-08-06)

19. Okta developer. URL: https://developer.okta.com/blog/2019/07/29/hashing-techniques-for-password-storage#evaluating-hashing-algorithms (2023-06-27)

20. Preneel, Bart. The First 30 Years of Cryptographic Hash Functions and the NIST SHA-3 Competition, 2010. URL: https://www.esat.kuleuven.be/cosic/publications/article-1532.pdf (2023-06-21)

21. Rivest, L. Ronald. The MD4 Message Digest Algorithm, 1991. URL: https://link.springer.com/content/pdf/10.1007/3-540-38424-3_22.pdf (2023-06-24)

22. Rivest, L. Ronald. The MD5 Message-Digest Algorithm, 1992. URL: https://www.ietf.org/rfc/rfc1321.txt (2023-06-25)

23. Sriramya, P.; Karthika, R.A. Providing password security by salted password hashing using Bcrypt algorithm, 2015. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=9d82620f0866ec773b12c30f90d70b74661f972f (2023-06-28)

24. StackExchange. URL: https://security.stackexchange.com/questions/36833/why-should-i-hash-passwords (2023-07-18)

25. Techopedia. URL: https://www.techopedia.com/definition/31699/message-digest-2-md2 (2023-06-23)

26. Techopedia. URL: https://www.techopedia.com/definition/4024/message-digest (2023-06-23)

27. TechTarget. URL: https://www.techtarget.com/searchdatamanagement/definition/hashing (2023-06-19)

28. TechTarget. URL: https://www.techtarget.com/searchsecurity/definition/brute-force-cracking (2023-07-02)

29. TechTarget. URL: https://www.techtarget.com/searchsecurity/definition/MD5 (2023-06-25)

30. Toolkit. URL: https://toolkit.vercel.app/sha-0/ (2023-07-29)