**SVEUČILIŠTE JURJA DOBRILE U PULI**

**FAKULTET INFORMATIKE**

**KOMPARACIJA VIŠE KRIPTOGRAFSKIH ALGORITAMA NA PRIMJERU SIGURNE POHRANE LOZINKI**

**BOŽENA MATAK**

Kolegij: Kriptografija

Mentor: doc.dr.sc. Siniša Miličić

Studijski smjer: Informatika – Diplomski studij

Pula, 2022. godina

**Sadržaj**

[1. Općenito o lozinkama 1](#_Toc95828173)

[2. Pohrana lozinki 2](#_Toc95828174)

[2.1. Kako sigurno pohraniti lozinke? 3](#_Toc95828175)

[2.2. Najpoznatiji napadi na lozinke 3](#_Toc95828176)

[3. Kriptografski algoritmi za sigurnu pohranu lozinki 4](#_Toc95828177)

[4. Argon2 5](#_Toc95828178)

[4.1. Značajke Argon2 algoritma 6](#_Toc95828179)

[5. Bcrypt 9](#_Toc95828180)

[6. Argon2 vs Bcrypt 12](#_Toc95828181)

[7. PBKDF2 12](#_Toc95828182)

[8. Argon2 vs Bcrypt vs PBKDF2 15](#_Toc95828183)

[9. Zaključak 17](#_Toc95828184)

[10. Literatura 18](#_Toc95828185)

**Uvod**

U današnje vrijeme brzim razvojem i napretkom informacijske tehnologije dostupnima postaju različite vrste zabave te pristupi zabavnim, edukativnim i profesionalnim sadržajima na Internetu. Kako bi se sve to moglo koristiti najčešće je potrebno izraditi korisnički račun i profil koji također podrazumijevaju i izradu lozinki. Pošto lozinke predstavljaju tzv. ključ do informacija kao takve su često izložene različitim napadima od strane *cyber* kriminalaca. *Cyber* kriminalci pokušavaju probiti lozinke u svrhu preuzimanja korisničkih računa, krađe identiteta, krađe novca, prodaje podataka, manipulacije i sl. Neki od najčešćih napada na lozinke, poput napada grube sile, rječnika i dugine tablice će biti objašnjeni i u ovom radu. Kako bi zaštitili svoje lozinke i tako osigurali podatke koje iste čuvaju jako je važno izraditi jaku i kvalitetnu lozinku te osigurati da je ona sigurno pohranjena.

Postoji mnogo različitih načina pohrane lozinki od kojih se *hashiranje* ističe kao najsigurnija metoda. Postupak *hashiranja* lozinki uključuje transformaciju lozinki u *hash* vrijednost u obliku niza bajtova fiksne duljine. Danas postoji jako veliki broj modernih algoritama za *hashiranje* koji su posebno dizajnirani za pohranu lozinki, no koji je od njih zaista najbolji za sigurnu pohranu lozinki?

Kroz ovaj rad će biti opisana tri algoritma – Bcrypt, Argon2 i PBKDF2. Također će biti prikazana implementacija i komparacija ovih algoritama kako bi se moglo determinirati koji je od njih najprikladniji i najsigurniji za korištenje u svrhu sigurne pohrane lozinki.

# **Općenito o lozinkama**

Lozinke predstavljaju ključni dio našeg svakodnevnog života. Primjerice kako bi se pristupilo raznim stvarima na internetu potrebna je lozinka. Zbog toga se može reći kako iste često predstavljaju ključ za dolazak do informacija. Različiti servisi i tvrtke koje upravljaju osobnim podacima inzistiraju kreiranje vlastitog „potpisa“ za sve čemu želimo pristupiti. Koristimo ih u različite svrhe no neke od najčešćih su: prijava na vlastito računalo, čitanje e-pošte, upravljanje financijama, kupovanje stvari online, a u nekim slučajevima čak i gledanje filmova.

Pošto lozinke predstavljaju ključ za dolazak do informacija kao takve su često ugrožene od strane *cyber* kriminalaca. Najčešće situacije: čitanje tuđe e-pošte, financijski gubitak – krađa novca, preuzimanje tuđeg profila/računa, prodaja podataka ili čak i krađa identiteta.

Kako bi se korisnik zaštitio od mogućih napada *cyber* kriminalaca, bitno je odabrati jaku lozinku i slijediti određena pravila prilikom tog odabira.

Korisni savjeti prilikom odabira lozinke:

* Upotreba jedinstvene lozinke za svaki račun. Ne koristiti istu lozinku na više računa
* Lozinka treba imati najmanje 8 znakova. Treba se sastojati od malih, velikih slova, brojeva i simbola. Duža lozinka će pružiti veću zaštitu od kraće lozinke ukoliko je pravilno sastavljena
* Ne koristiti osobne podatke kao što su ime, dob, datum rođenja i sl.
* Uvijek se odjaviti nakon završetka rada na računalu
* Izbjegavati unos lozinki na računala koja se ne mogu kontrolirati (npr. računalo u knjižnici ili kafiću)
* Ne dijeliti lozinku

# **Pohrana lozinki**

Autentifikacija korisnicima pruža skup vjerodajnica kao što su korisničko ime i lozinka te ih provjerava svaki put kada korisnici žele pristupiti aplikaciji. Kako bi se korisničko ime i zaporka mogli uspoređivati u budućnosti, iste je potrebno pohraniti u bazu podataka.

Postoji nekoliko načina pohrane lozinke:

* Lozinka u obliku plaintexta – Jednostavan pristup pohranjivanju lozinki je stvaranje tablice u bazi podataka koja mapira korisničko ime s lozinkom. Kada se korisnik prijavi, poslužitelj dobiva zahtjev za autentifikaciju s danim korisničkim imenom i lozinkom. Traži korisničko ime u tablici i uspoređuje priloženu lozinku sa pohranjenom lozinkom. Ukoliko se lozinke podudaraju korisniku je omogućen pristup aplikaciji.

Problem kod ovakvog načina pohrane lozinki jest taj da ukoliko napadač provali u bazu podataka i ukrade tablicu lozinki mogao bi pristupiti bilo kojem korisničkom računu. Bez obzira koliko je lozinka složena, ona je čitljiva.

* Enkripcija – enkripcija i dekripcija– bolji pristup od prethodnog jest postupak šifriranja, tj. enkripcija lozinke prije pohranjivanja u bazu podataka. Enkripcija je proces transformacije informacija korištenjem algoritma kako bi one postale nečitljive svakome osim onima koji posjeduju ključ. Rezultat enkripcije su šifrirane informacije – *ciphertext*. U mnogim kontekstima riječ enkripcija se također implicitno odnosi na obrnuti proces, dekripcija, kako bi se šifrirana informacija ponovno učinila čitljivom.

Problem kod postupka šifriranja jest taj što je isti dvosmjeran, odnosno ako napadač dobije ključ zna koji se algoritam koristi te lako može dešifrirati tekst kako bi dobio izvornu lozinku.

* *Hashiranje*– predstavlja sigurniji način pohrane lozinki. Postupak kojim se lozinke ne mogu pretvoriti natrag u izvornu lozinku zato što je jednosmjeran. *Hashing* je složena matematička funkcija koja pretvara podatke proizvoljne veličine u podatke fiksne veličine.

## **2.1. Kako sigurno pohraniti lozinke?**

Ranije su spomenuti načini pohrane lozinki u bazu podataka od kojih se *hashiranje* izdvaja kao najsigurnija metoda. Postupak *hashiranja* uključuje transformaciju lozinke u *hash* vrijednost u obliku niza bajtova fiksne duljine. Na taj se način onemogućava dohvaćanje duljine ili složenosti lozinke pošto će *hash* vrijednost uvijek biti iste duljine, čak i mala promjena lozinke će potpuno promijeniti *hashiranu* vrijednost. Ključna značajka postupka *hashiranja* jest ta da je isti jednosmjeran, odnosno transformirana lozinka se ne može pretvoriti natrag u izvornu lozinku. Za razliku od enkripcije iz *hash* vrijednosti je nemoguće izdvojiti lozinku na temelju pogađanja. Ukoliko se dogodi da je baza podataka kompromitirana, važno je da su svi podaci pohranjeni na siguran način kako bi se spriječilo *cyber* kriminalce od direktnog korištenja informacija. Postupak *hashiranja* lozinke bi trebao biti temeljen na standardnoj implementaciji isprobane i testirane *hash* funkcije koja je dizajnirana posebno za lozinke.

Prilikom *hashiranja* kao dodatni sloj sigurnosti se dodaje tzv. „*salt*“. *Salt* je slučajna vrijednost koja se dodaje na lozinku prije *hashiranja* te tako osigurava da je rezultirajuća vrijednost uvijek jedinstvena čak i ako su lozinke potpuno identične. *Salt* se treba čuvati odvojeno, a ne zajedno s lozinkom.

## **2.2. Najpoznatiji napadi na lozinke**

Pošto lozinke predstavljaju najrašireniji oblik potvrde autentičnosti kao takve postaju mete mnogih *cyber* kriminalaca.

Postoji nekoliko poznatih napada za probijanje *hashiranih* lozinki:

* Rječnik (engl. *Dictionary*) – najbrži i najjednostavniji način za otkrivanje lozinki. Napad rječnikom predstavlja tehniku grube sile u kojoj napadači prolaze kroz uobičajene riječi i izraze kako bi pogodili lozinke. Poznato je da mnogi ljudi koriste jednostavne i lako pamtljive lozinke na više računa što napadačima omogućava lakši dolazak do željenih informacija.
* Dugina tablica (eng. *Rainbow table*) – predstavlja bazu podataka koja se koristi za provjeru autentičnosti probijanjem *hash* lozinke. Dugina tablica je unaprijed izračunati rječnik lozinki otvorenog teksta i njihovih odgovarajućih *hash* vrijednosti koji se koristi kako bi se saznalo koja lozinka otvorenog teksta može proizvesti isti *hash*.
* Napad grube sile (eng. *Brute force*)– vremenski je najzahtjevniji napad. Napad grube sile uključuje testiranje svih mogućih kombinacija slova, znamenki i simbola. Kako bi napad grube sile bio uspješan potrebno je snažno računalo.

# **Kriptografski algoritmi za sigurnu pohranu lozinki**

Danas postoji jako veliki broj modernih algoritama za *hashiranje* koji su posebno dizajnirani za sigurnu pohranu lozinki. Uspješni i kvalitetni algoritmi koji se koriste za sigurnu pohranu lozinki trebaju biti spori. Kriptografska *hash* funkcija koja se koristi za *hashiranje* lozinki mora biti spora za izračunavanje jer bi u suprotnom brzo izračunati algoritam mogao učiniti napade grubom silom izvedljivijim, posebno uz brzo razvijajuću snagu modernog hardvera. Usporavanje *hash* funkcije se provodi dodavanjem većeg broja internih iteracija. Važno je postići dobru ravnotežu brzine i upotrebljivosti za funkcije *hashiranja*. Dobronamjerni korisnik neće imati primjetan učinak na performanse kada pokuša ući u aplikaciju s jednom valjanom prijavom.

Dobri algoritmi za kriptografske *hash* funkcije trebaju imati sljedeća svojstva:

* *Pre – Image Resistance* – ulazna vrijednost k treba biti teža za pronaći iz poznate *hash* vrijednosti v
* *Second Pre – Image Resistance* – za ulaznu vrijednost k koja ima *hash* vrijednost v, trebalo bi biti teško pronaći drugu ulaznu vrijednost koja također izbacuje istu *hash* vrijednost v
* *Collision Resistance* – trebalo bi biti teško pronaći par ulaznih vrijednosti koji ima iste izlazne vrijednosti

Kao što je ranije spomenuto postoji puno modernih kriptografskih algoritama koji se koriste u svrhu pohrane lozinki, a među najpopularnijima i najkorištenijima su: Argon2, PBKDF2 te Bcrypt. Na sljedećim stranicama će biti detaljnije prikazana svojstva ovih algoritama na temelju kojih će se moći doći do zaključka o tome koji je najsigurniji i najbolji za korištenje u svrhu sigurne pohrane lozinki.

# **Argon2**

Argon2 predstavlja kriptografski algoritam kojeg su dizajnirali Alex Biryukov, Daniel Dinu i Dmitry Khovratovich. Napravljen je za brzo punjenje memorije i korištenje više računalnih jedinica učinkovito, ali u isto vrijeme i dalje omogućava pružanje obrane od napada. Argon2 je optimalan i dizajniran za hardver sa x86 arhitekturom, a izmišljen je kako bi se postigla osobina teškog pamćenja *hash* funkcije.

Postoje tri dostupne verzije Argon2 algoritma, a to su:

* Argon2d – fokusira se na brzo računanje te koristi podatke ovisno o pristupu memoriji. Pogodan je za aplikacije bez prijetnji od vremenskih napada sa bočnih strana kao što su kriptovalute.
* Argon2i – koristi podatke neovisno o pristupu memoriji. Poželjniji je za *hashiranje* lozinki i za izvođenje ključa na temelju lozinke. Argon2i je nešto sporiji pošto radi više prijelaza preko memorije kako bi se zaštitilo od napada kompromisa.
* Argon2id – se ponaša kao Argon2i za prvu polovicu prve iteracije preko memorije, a za ostalo radi kao Argon2d čime se osigurava ušteda troškova napada grubom silom i napada sa bočnih strana kao što su *meltdown* i *spectre,* zaštita zbog kompromisa s vremenskom memorijom.

Postoje dva tipa ulaza za Argon2, a to su primarni i sekundarni. Primarni ulazi su:

* Poruka P – može imati bilo koju duljinu od 0 do bajtova
* Lozinka
* Jednokratni S za *salt.*- može imati bilo koju duljinu od 8 do bajtova

Kako bi se Argon2 mogao koristiti primarni inputi su obavezni. Sekundarni ulazi su izborni i sastoje se od nekoliko parametara kao što su:

* Stupanj paralelizma P – bilo koji broj od 1 do bajt
* Duljina oznake τ – bilo koji cijeli broj bajtova od 4 do bajt
* Veličina memorije m – bilo koji cijeli broj kilobajta od 8 do
* Broj iteracija t – bilo koji cijeli broj od 1 do bajt
* Broj verzije v – jedan bajt 0x13
* Tajna vrijednost K – bilo koji broj od 0 do bajt
* Povezani podatak X – može imati bilo koju duljinu od 0 do bajtova
* Tip y za Argon2 – 0 za Argon2d, 1 za Argon2i i 2 za Argon2id

## **Značajke Argon2 algoritma**

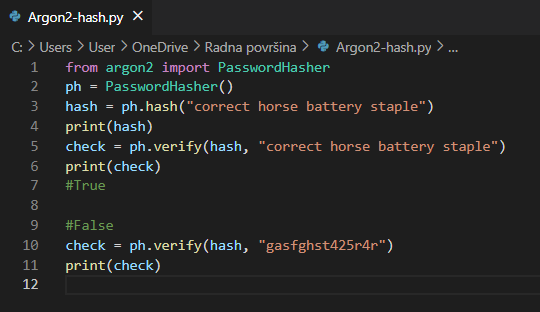
Argon2 je višenamjenska obitelj shema za *hashiranje* koja je prikladna za *hashiranje* lozinki, izvođenje ključa, kriptovalute i druge aplikacije koje zahtijevaju dokazivo veliku upotrebu memorije. Kao što je već ranije spomenuto Argon2 je optimiziran za x86 arhitekture, ali ne usporava puno na starijim procesorima. Najvažnija značajka Argon2 je njegova izvedba i mogućnost korištenja više računalnih jezgri na način koji zabranjuje kompromise između vremena i memorije.

Postoji nekoliko značajki koje nisu uključene u trenutnu verziju, ali se lako mogu dodati kasnije. Moguće buduće ekstenzije – Argon2 se prilično lako može podesiti da podržava druge funkcije kompresije, *hash* funkcije i veličine blokova. ROM se može integrirati jednostavnim uključivanjem u područje odakle su blokovi referencirani.

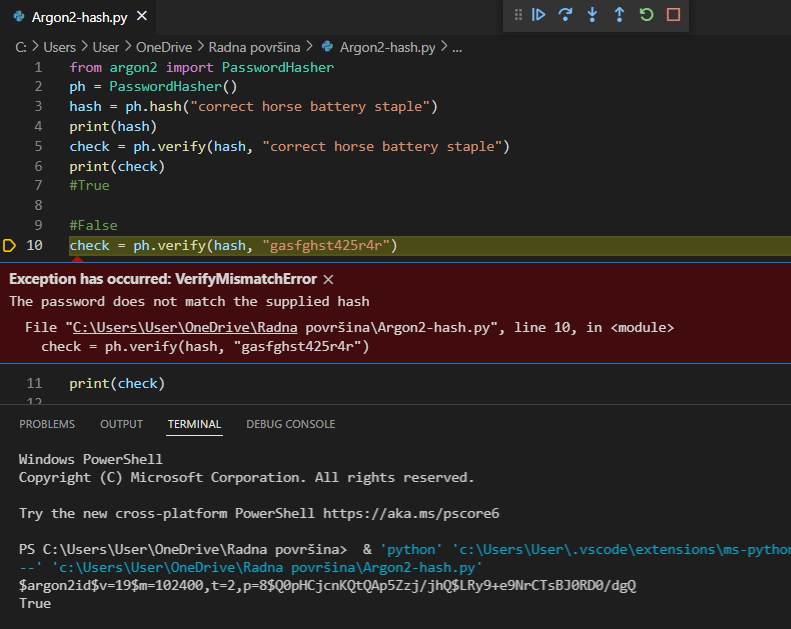
Između ostaloga bitno je za naglasiti kako je u natjecanju koje se održalo između 2012. i 2015. godine, kako bi se pronašao siguran algoritam za *hashiranje* lozinki, Argon2 proglašen pobjednikom.

U nastavku ovog poglavlja će biti prikazana implementacija Argon2 algoritma u Python kodu u svrhu prikaza *hashiranja* lozinke te će biti objašnjeni ključni elementi tog koda.

Prvo je potrebno instalirati Argon2 ukoliko to nije napravljeno ranije, korištenjem naredbe *pip install argon2.* Nakon što je Argon2 instaliran potrebno je importati *PasswordHasher* klasu. Klasa već sadrži sve potrebne parametre, a to su: *time\_cost* – broj iteracija, *memory\_cost* – memorija za korištenje u KB, paralelizam – koliko paralelnih niti koristi, *hash\_len* – veličina izvedenog ključa, *salt-len* – duljina nasumične soli koja se generira za svaku lozinku. Kod sadrži prikaz *hashiranja* lozinke „*correct horse battery staple*“ te se ispisuje njezina *hash* vrijednost. Nakon ispisa se vrši provjera koja u prvom slučaju vraća *true* vrijednost pošto *hash* odgovara lozinki, a u drugom slučaju vraća *false* pošto se lozinka ne podudara s *hashiranom* vrijednosti. Na slici 2 se nalazi ispis programa nakon pokretanja.



Slika 1. Primjer hashiranja lozinke pomoću Argon2 algoritma



Slika 2. Rezulat ispisa koda za hashiranje - Argon2

# **Bcrypt**

Bcrypt je *hash* funkcija koju su dizajnirali Niels Provos i David Mazieres, predstavljena na USENIX-u 1999. godine. Razvijena je u svrhu učvršćivanja pohrane lozinki Unix autentifikacijskog sustava. Bcrypt ima nekoliko parametara, a najvažniji od njih je broj iteracija. Broj iteracija se može konfigurirati na određenu vrijednost koja usporava proces probijanja zbog sigurnosti, ali je opet dovoljno brza za provjeru zadane lozinke. Zbog mogućnosti konfiguracije broja iteracija Bcrypt može pratiti rastuću snagu hardvera.

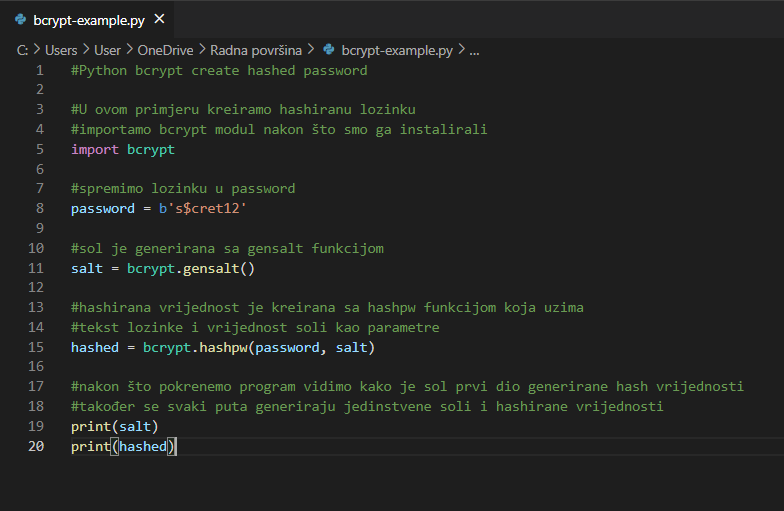
Bcrypt se temelji na *Blowfish* šifri koju je dizajnirao Bruce Schneier, a poznata je još i kao složena simetrična blok šifra. Prilikom rada Bcrypt koristi prednosti skupog podešavanja ključa *Eksblowfish.* koji se odnosi na skupi raspored ključeva *blowfish*, što je zapravo parametizirana i „posoljena“ varijacija *Blowfish* šifre. *Eksblowfish* se sastoji od tri parametra, a to su: trošak, sol i ključ. Parametar troška je ono što ovaj algoritam čini prilagodljivim u računskoj snazi jer isti kontrolira koliko je skupo izračunavanje rasporeda ključeva. Povećanjem vrijednosti troška stvara se skuplji raspored ključeva za izračunavanje. Termin ključ zapravo predstavlja lozinku koju korisnik odabere.

Prilikom odabira lozinke potrebno je pripaziti da njezina duljina ne bude više od 72 bajta ili će u suprotnome ista biti skraćena.

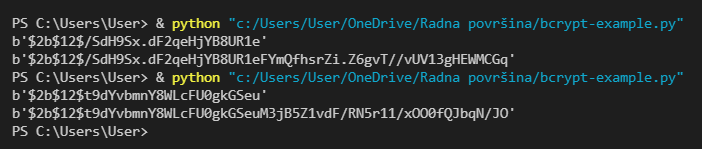
Bcrypt je implementiran u mnogim programskim jezicima kao što su: C, C++, C#, Java, JavaScript, Perl, PHP, Python, Rubi i dr.

U nastavku ovog rada će biti prikazana jednostavna implementacija Bcrypt algoritma koja će sadržavati primjer kreiranja *hashiranja* lozinke, provjeru lozinke u odnosu na *hashiranu* vrijednost i definiranje faktora troška koji povećava sigurnost usporavanjem postupka *hashiranja*.

U sljedećem primjeru prikazan je postupak kreiranja *hashirane* lozinke pomoću Bcrypt algoritma. Pomoću naredbe *import bcrypt* prvo unosimo bcrypt modul koji je potreban za rad, željena lozinka se sprema u varijablu *password* te se generira ranije spomenuta sol pomoću *gensalt* funkcije. Nakon što je definirano sve što je potrebno *hashirana* vrijednost se generira pomoću *hashpw* funkcije koja kao parametre uzima tekst lozinke i sol. Nakon što se kod pokrene vidljivo je da je sol prvi dio generirane *hash* vrijednosti. Također bitno je za primjetit kako se svaki puta prilikom pokretanja programa generiraju jedinstvene soli i hashirane vrijednosti što je vidljivo iz Slike 4.

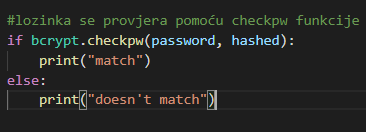


Slika 3. Bcrypt primjer kreiranja hashirane lozinke

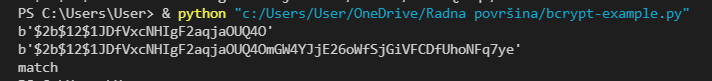


Slika 4. Ispis hashirane vrijednosti pomoću Bcrypt-a

Lozinku je moguće provjeriti pomoću *checkpw* funkcije na Slici 5, a na Slici 6 je vidljivo kako se u konzoli ispisuje poruka *match* što znači da se u ovom slučaju lozinka i *hash* podudaraju.

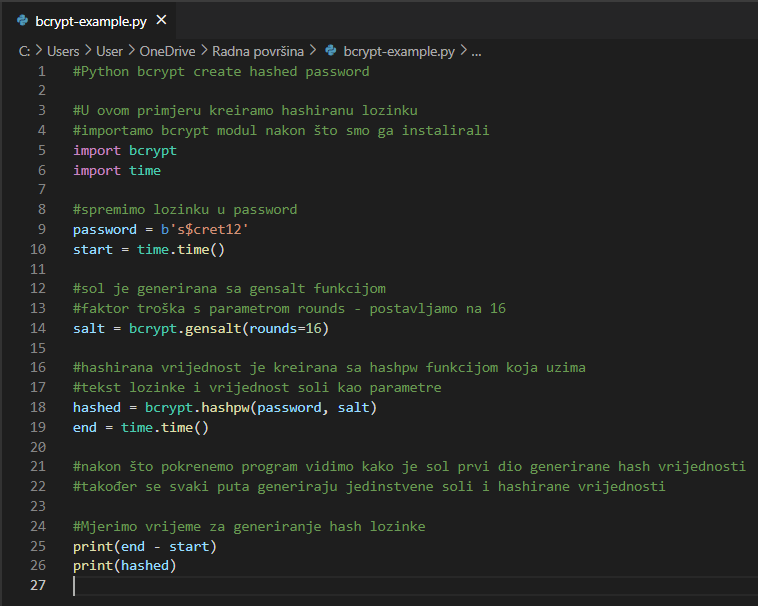


Slika 5. Provjera pomoću checkpw funkcije – Bcrypt



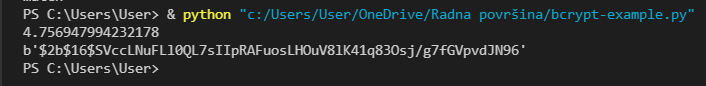
Slika 6. Ispis checkpw funkcije – Bcrypt

Na sljedećoj slici prikazan je isti kod za *hashiranje* lozinke, razlika je u tome što je dodan faktor troška koji je definiran pomoću parametra *rounds* te je isti postavljen na 16. Pomoću faktora troška moguće je izmjeriti vrijeme potrebno za generiranje *hash* vrijednosti.



Slika 7. Mjerenje faktora troška – Bcrypt

Slika 8 prikazuje koliko je bilo potrebno vremena za generiranje *hash* vrijednosti s navedenim troškom faktora te je vidljivo da je potrebno nešto više od četiri sekunde.



Slika 8. Ispis mjerenja faktora troška - Bcrypt

# **Argon2 vs Bcrypt**

Poznato je kako se danas specijalizirani računalni čipovi kao što su FPGA, ASIC i GPU, lako mogu nabaviti od strane napadača, puno lakše nego ogromne količine memorije. Zbog ove činjenice se otvara pitanje slabosti Bcrypt-a koji se fokusira na troškove računanja. Bolja i kvalitetnija *hash* funkcija bi trebala biti dovoljno fleksibilna za razmjenu velike memorije za brzo računanje i male memorije za sporo računanje. Argon2 pruža najveću fleksibilnost. Ranije je već spomenuto kako je pobijedio na natjecanju za *hashiranje* lozinki u srpnju 2015. godine. Argon2 se smatra poboljšanjem algoritama Bcrypt i Scrypt, ima unaprijed definiranu proizvoljnu i podesivu veličinu memorije, element CPU vremena i paralelizam, dok Bcrypt nema posebno podesivu upotrebu memorije nego je ista fiksirana za zadanu količinu CPU vremena. Kombinacija ovih elemenata kod Argon2 algoritma čini napad grubom silom jako skupim jer zahtijeva da napadači imaju veliku memoriju za računanje i memorijske resurse.

Uzimajući u obzir današnju računsku snagu i specifične hardvere, za ublažavanje općih napada u kriptografiji, Argon2 pruža znatno poboljšanje u odnosu na Bcrypt.

# **PBKDF2**

PBKDF2 (*eng. Password Key derivation function*) je funkcija deriviranja ključa lozinke koja je osmišljena da bude algoritamski spora kako bi učinkovitost „*Rainbow*“ tablica bila reducirana. Broj iteracija je jedan od najvažnijih parametara PBKDF2 algoritma. Odabir većeg broja ponavljanja usporava napadača, ali može negativno utjecati na upotrebljivost. NIST preporučuje minimalno 1000 iteracija za primjene opće namjene, ali sugerira da se odabere što je moguća veća vrijednost broja iteracija. Mnoge aplikacije ovu vrijednost definiraju a priori. Kako bi se pružila bolja otpornost na napade grube sile, PBKDF2 uvodi CPU-intenzivne operacije. Navedene operacije se temelje na ponavljanju pseudoslučajne funkcije (PRF) koja preslikava ulazne vrijednosti u izvedeni ključ. Najvažnije svojstvo koje treba osigurati je to da iterirana pseudoslučajna funkcija bude bez ciklusa jer ako to nije tako, zlonamjerni korisnik može izbjeći CPU intenzivne operacije i dobiti izvedeni ključ izvršavanjem skupa funkcionalno ekvivalentnih uputa. PBKDF2 algoritam je opisan i uveden u internetski standard RFC 2898 (PKCS #5).

PBKDF2 ima pet ulaznih parametara, a to su:

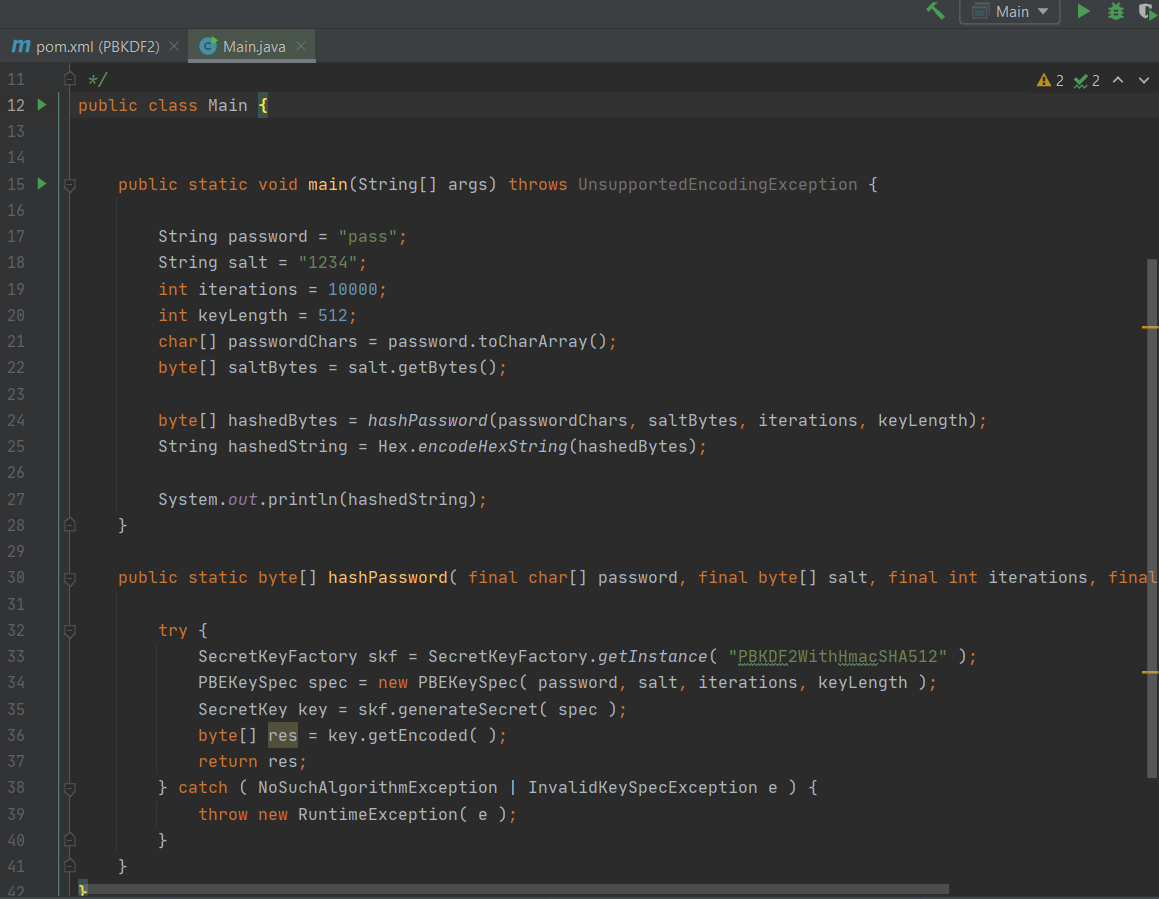
* PRF – pseudoslučajna funkcija dvaju parametara s izlaznom duljinom hLen
* Lozinka – glavna lozinka iz koje se generira izvedeni ključ
* Sol – niz bitova
* c – željeni broj iteracija
* dkLen – željena duljina bita izvedenog ključa
* DK – generirani izvedeni ključ

Ovaj algoritam najčešće dolazi u kombinaciji sa SHA algoritmima gdje se može postići i do preko sto tisuća iteracija, kao npr. u PBKDF2-SHA512 inačici, koja koristi 512-bitni ključ, a svoju primjenu nalazi u Dropbox i iCloud sustavima. Poznato je još da se primjenjuje i u MAC operacijskim sustavima kod kriptiranja lozinki te u upotrebi privjesaka (*eng. Keychains*).

U sljedećem dijelu rada će biti prikazan jednostavan primjer korištenja PBKDF2 algoritma za *hashiranje* lozinki. Program je napisan u *IntelliJ* programu u Java programskom jeziku.

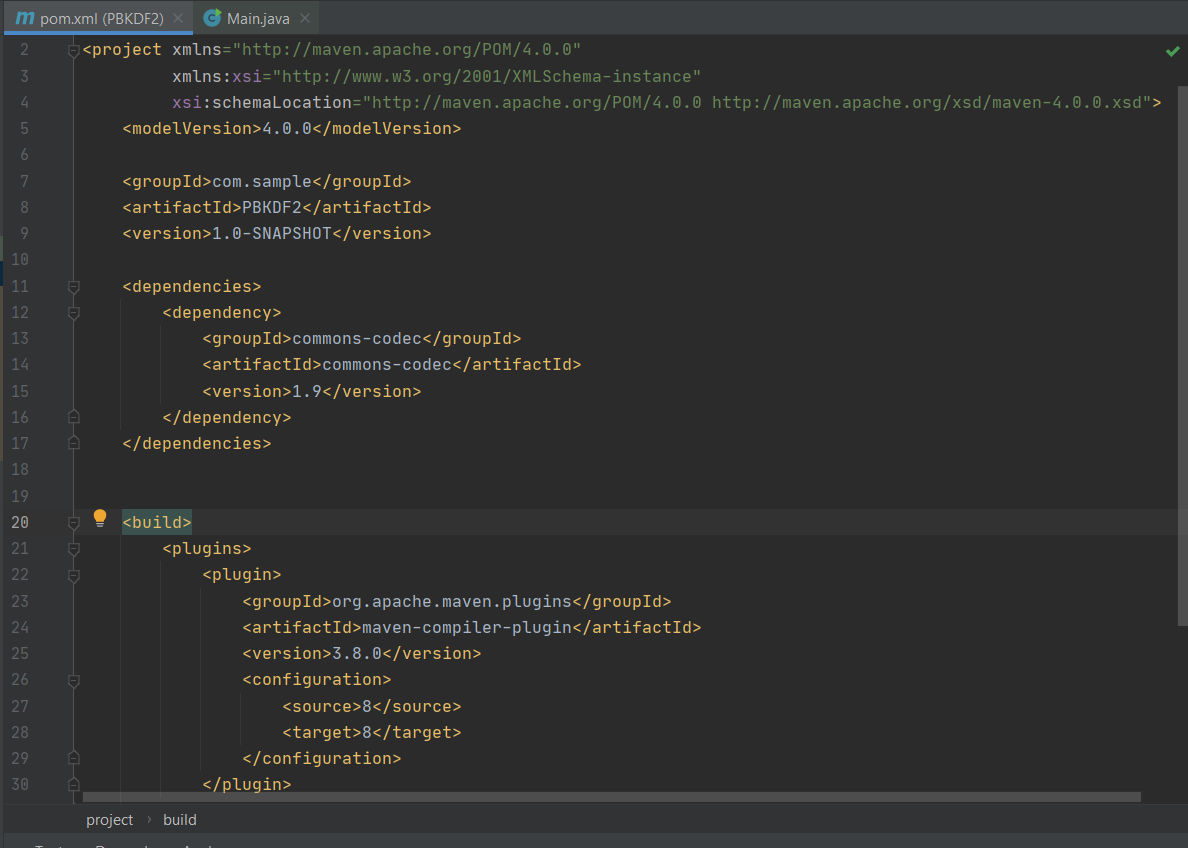
Za početak je potrebno kreirati *maven* projekt i *Main.java* klasu unutar koje će se nalaziti programski kod. Datoteka pom.xml je kreirana automatski prilikom kreiranja projekta te ista sadrži različite *dependencies*. *Dependencies* je prethodno potrebno definirati, a u ovom slučaju se definira *dependency* za pretvaranje niza bajtova koje vraća *hashPassword()* u String.

U Main.java klasi je kreirana metoda *hashPassword()* koja sadrži funkcionalnosti PBKDF2 te se može vidjeti način na koji su definirane četiri ulazne varijable. Lozinka – vrijednost koju je potrebno *hashirati*, sol – vrijednost koja se dodaje lozinki, broj iteracija – broj iteracija koje se trebaju izvršiti, služi podešavanju brzine izvođenja algoritma i duljina ključa – duljina outputa *hashirane* vrijednosti. Prethodno navedeno je moguće vidjeti na slici 9.



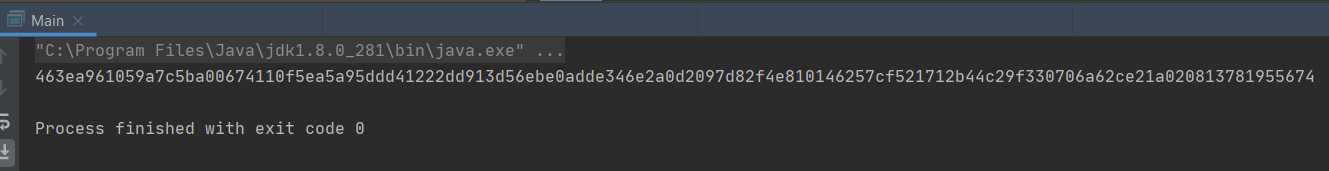
Slika 9. Sadržaj Main.java klase

Metoda *hashPassword()* vraća niz bajtova koje je potrebno pretvoriti u niz pomoću prikladnog heksadecimalnog kodera, u ovom programu je korištena biblioteka *Apache Commons Codec* koja se može vidjeti pod ranije spomenutim *dependencies* u pom.xml datoteci. Na slici 10 se nalazi sadržaj pom.xml datoteke.



Slika 10. Sadržaj pom.xml datoteke

Nakon izvršenja programa dobije se output kao na slici 11.



Slika 11. Rezultat hashiranja lozinke - PBKDF2

# **Argon2 vs Bcrypt vs PBKDF2**

Na temelju detaljnijeg opisa svojstava i načina rada ova tri algoritma u ranijim poglavljima, može ih se usporediti te se dolazi do sljedećih zaključaka. Što se tiče PBKDF2 algoritma glavni nedostatak je taj što je isti implementiran kao mali sklop s malo RAM-a što s druge strane omogućuje zlonamjernim korisnicima jeftine napade grube sile na ASIC i GPU-ove. Bcrypt algoritam s druge strane koristi 4KB RAM-a, manje je paralelan te je nešto jači od PBKDF2 u obrani od ASIC i GPU-ova, ali ipak zahtjevi za memorijom dopuštaju učinkovite napade na FPGA. Također, u odnosu na PBKDF2, Bcrypt je znatno sporiji jer koristi *Blowfish* složenu simetričnu blok šifru, koja koristi iteracije kao dva na potenciju broja stupnja parametara iteracija. Argon2 s druge strane pripada posebnoj kategoriji *hash* funkcija koje se nazivaju „*memory hard functions*“. Odnosi se na funkcije koje su dizajnirane tako da koriste proizvoljno veliku i podesivu količinu memorije čineći tako veličinu i cijenu njihove hardverske implementacije mnogo skupljom, a samim time i ograničavaju količinu paralelizma koju napadač može koristiti. PBKDF2 i Bcrypt nemaju posebno podesivu upotrebu memorije kao Argon2 nego je ista fiksirana za zadanu količinu CPU vremena. Usporedbom Bcrypt i PBKDF2 s Argon2 algoritmom brzo se dolazi do zaključka kako je Argon2 memorijski sigurnija *hash* funkcija, optimizirana je za jasnoću i učinkovitost. Glavna značajka i prednost Argon2 algoritma u odnosu na druge algoritme jest ta da pruža zaštitu od napada kompromisa nametanjem značajne kazne na vrijeme trajanja algoritma za bilo kakvo smanjenje dostupne memorije od strane napadača.

Usporedbom ovih triju algoritama jasno je kako Argon2 predstavlja najbolje rješenje kada se radi o sigurnoj pohrani lozinki.

# **Zaključak**

Obavljanje različitih radnji na Internetu od korisnika zahtjeva unos i korištenje lozinki koje najčešće čuvaju određene privatne podatke i informacije te iz tog razloga postaju meta mnogih *cyber* kriminalaca. Kako pohranjeni podaci ne bi bili kompromitirani od strane napadača potrebno je omogućiti sigurnu pohranu lozinki. Proces sigurne pohrane lozinki uključuje postupak *hashiranja*, a odnosi se na transformaciju lozinke u *hash* vrijednost u obliku niza bajtova fiksne duljine.

Poznato je kako postoji mnogo različitih kriptografskih algoritama za *hashiranje* koji su posebno dizajnirani za sigurnu pohranu lozinki. Najpoznatiji te ujedno i obrađeni u ovom radu su: Argon2, Bcrypt i PBKDF2.

Temeljem komparacije ovih triju algoritama dolazi se do zaključka kako je Argon2 najprikladnija i najsigurnija opcija prilikom odabira algoritma za sigurnu pohranu lozinki. Argon2 u odnosu na Bcrypt i PBKDF2 je memorijski sigurnija *hash* funkcija, optimiziran je za jasnoću i učinkovitost te ono što je najbitnije, pruža najveću zaštitu od različitih napadača. Nakon Argon2 kao sigurnija varijanta slijedi PBKDF2 koji pruža veću zaštitu u obrani od ASIC-a i GPU-ova u odnosu na Bcrypt koji je već pomalo zastario te je znatno sporiji jer koristi *Blowfish* složenu simetričnu blok šifru.

Iz prethodno navedenog se zaključuje kako se u svrhu sigurne pohrane lozinki preporuča korištenje Argon2 algoritma, kao alternativa se preporuča PBKDF2 dok Bcrypt preostaje kao zadnja opcija.

# **Literatura**

1. CyberHound, Password security research paper, <https://www.superloop.com/documents/ebooks/CyberHound%20Research%20Paper%20-%20Password%20Security.pdf>

2. Guide - Password security, Password guide for IT users, developers, system administrators and senior management, <https://www.cfcs.dk/globalassets/cfcs/dokumenter/vejledninger/en/CFCS-guidance-password-security-en-.pdf>

3. Argon2, the memory-hard function for password hashing and other applications, Alex Biryukov, Daniel Dinu and Dmitry Khovratovich, University of Luxembourg, Luxembourg, <https://www.password-hashing.net/argon2-specs.pdf>

4. Choose Argon2 Parameters for Secure Password Hashing and Login, 2020, <https://www.ory.sh/choose-recommended-argon2-parameters-password-hashing/>

5. Argon2, 2021 - <https://cryptobook.nakov.com/mac-and-key-derivation/argon2>

6. Argon2: The Better Password Hashing Function Than Bcrypt, Daniel Ryan Levyson, Institut Teknologi Bandung, Indonesia 2019, <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2019-2020/Makalah2019/13516132.pdf>

7. Passlib Documentation, Release 1.7.4 - Assurance Technologies, LLC, 2020, <https://buildmedia.readthedocs.org/media/pdf/passlib/stable/passlib.pdf>

8. Evaluation of Password Hashing Schemes in Open Source Web Platforms, Christoforos Ntantogian, Stefanos Malliaros, Christos Xenakis, Department of Digital Systems, University of Piraeus, Piraeus, Greece, <https://www.cyberwatching.eu/sites/default/files/Passwords-sub-v26.pdf>

9. Algorithms, Key Size and Protocols Report, Editor: Nigel P. Smart, Contributors: Michel Abdalla, Carlos Cid, Benedikt Gierlichs, Andreas H¨ulsing, Atul Luykx, Kenneth G. Paterson, Bart Preneel, Ahmad-Reza Sadeghi, Terence Spies, Martijn Stam, Michael Ward, Bogdan Warinschi, Gaven Watson, 2018, <https://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf>

10. Examining PBKDF2 security margin - case study of LUKS \*, Andrea Visconti, Ondrej Mosnaček, Milan Brož, and Vashek Matyaš, Department of Computer Science, Universita degli Studi di Milano, <https://www.researchgate.net/publication/332428424_Examining_PBKDF2_security_margin_---_case_study_of_LUKS?fbclid=IwAR3_OGuY8CL7cacst85Q5ysWTWqRXEl0LF2bgcZ744SwQ70-clDG3_vfww0>

11. Understanding Optimizations and Measuring Performances of PBKDF2 \*, Andrea Francesco Iuorio and Andrea Visconti\*\*, Department of Computer Science, Università degli Studi di Milano, <https://www.researchgate.net/publication/331198410_Understanding_Optimizations_and_Measuring_Performances_of_PBKDF2?fbclid=IwAR2nNMKqNarFZHZdOns8tnYxgXtahdUghfg-_yOUDmEi5hn7LFYBkYZx_aw>

12. Password Hashing Competition - Survey and Benchmark - George Hatzivasilis, Ioannis Papaefstathiou and Charalampos Manifavas, Dept. of Electronic & Computer Engineering, Technical University of Crete, Chania, Crete, Greece, Dept. of Informatics Engineering, Technological Educational Institute of Crete, Heraklion, Crete, Greece, <https://www.researchgate.net/publication/273959978_Password_Hashing_Competition_-_Survey_and_Benchmark>

13. Implementation of Performance Analysis of PBKDF2, Bcrypt, Scrypt Algorithms - Levent Ertaul, Manpreet Kaur, Venkata Arun Kumar R Gudise CSU East Bay, Hayward, CA, USA, <http://borg.csueastbay.edu/~lertaul/PBKDFBCRYPTCAMREADYICWN16.pdf>

14. On the weaknesses of PBKDF2 \*, Andrea Visconti, Simone Bossi, Hany Ragab, and Alexandro Calò,Cryptography and Coding Laboratory (CLUB), Department of Computer Science, Università degli Studi di Milano, <http://homes.di.unimi.it/visconti/PBKDF2.pdf>

15. Password Storage Cheat Sheet, 2021, <https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Password_Storage_Cheat_Sheet.md>

16. PBKDF2, 2021, <https://cryptobook.nakov.com/mac-and-key-derivation/pbkdf2>

17. Neurotechnics, Password-Based Key Derivation Function 2 (PBKDF2), <https://neurotechnics.com/tools/pbkdf2-test>

18. PBKDF2 Hash a secure password, Hashing a secure password to safe storage, 2019, <https://dev.to/demg_dev/pbkdf2-hash-a-secure-password-5f8l?fbclid=IwAR3_OGuY8CL7cacst85Q5ysWTWqRXEl0LF2bgcZ744SwQ70-clDG3_vfww0>

19. SSLTrust, PBKDF2: Password Based Key Derivation, <https://www.ssltrust.com.au/blog/pbkdf2-password-key-derivation?fbclid=IwAR2Hh7pQK-haUH_Djw2m931qa46M_-BYOGDOsqUR6H0J9KZFbvsX0P9GuMQ#PBKDF2-hash>

20. How to store passwords securely with PBKDF2, Kasun Dharmadasa, 2017, <https://medium.com/@kasunpdh/how-to-store-passwords-securely-with-pbkdf2-204487f14e84>