



I. osnovna šola Celje  
Vrunčeva ulica 13  
3000 Celje

# Od 123456 do množic

Računalništvo

RAZISKOVALNA NALOGA

Avtor:  
Rudi Rodič

Mentor:  
Žan Močivnik, prof. mat. in rač.

Celje, marec 2025

### ***Zahvala***

Zahvaljujem se vsem, ki so pripomogli k nastanku te raziskovalne naloge. Pri raziskovalni nalogi mi je bil v največjo pomoč mentor Žan Močivnik, ki mi je bil vedno na voljo za morebitna vprašanja, za kar sem mu zelo hvaležen. Zahvaljujem se tudi vodstvu I. osnovne šole Celje, ki mi je omogočilo raziskovalno nalogo.

## **POVZETEK**

V raziskovalni nalogi sem preučeval varnost gesel in možnosti njihovega izboljšanja s pomočjo programiranja v jeziku Python. Glavni cilj naloge je bil razviti program, ki bi ustvarjal gesla, ki so enostavna za pomnjenje, a hkrati dovolj varna pred napadi, kot so brute-force in slovarski napadi.

V teoretičnem delu sem raziskal zgodovino kriptografije, pomen gesel in različne algoritme za njihovo zaščito, kot so SHA-256, bcrypt in Argon2. Nato sem v praktičnem delu razvil program v Pythonu, ki na podlagi uporabnikovih odgovorov generira prilagojena gesla. Ta gesla sem analiziral in primerjal z gesli, ki jih ustvariyo drugi sistemi, kot sta ChatGPT in Google. Analiza je pokazala, da popolne kombinacije med varnostjo in zapomljivostjo ni mogoče doseči, saj so varna gesla pogosto težje zapomnljiva.

Rezultati raziskave potrjujejo, da so daljša in kompleksnejša gesla varnejša, vendar je njihova uporabnost omejena. Najboljša praksa za varnost gesel vključuje uporabo geselskih upraviteljev, večfaktorske avtentikacije in algoritmov, ki zagotavljajo ravnovesje med varnostjo in praktičnostjo.

**Ključne besede:** gesla, varnost gesel, kriptografija, generiranje gesel, Python

## **ABSTRACT**

In this research paper, I examined password security and the possibilities for improving it using Python programming. The main goal was to develop a program that generates passwords that are both easy to remember and sufficiently secure against attacks such as brute-force and dictionary attacks.

In the theoretical part, I explored the history of cryptography, the importance of passwords, and various protection algorithms such as SHA-256, bcrypt, and Argon2. In the practical part, I developed a Python program that generates personalized passwords based on user responses. These passwords were analyzed and compared with those generated by other systems, such as ChatGPT and Google. The analysis revealed that a perfect balance between security and memorability is difficult to achieve, as secure passwords are often harder to remember.

The research results confirm that longer and more complex passwords are more secure, but their usability is limited. The best practice for password security includes using password managers, multi-factor authentication, and algorithms that provide a balance between security and practicality.

**Key Words:** passwords, password security, cryptography, password generation, Python

## *Kazalo vsebine*

1	Uvod.....	1
1.1	Opis raziskovalnega problema in namen .....	1
1.2	Hipoteze .....	1
1.3	Raziskovalne metode .....	1
1.4	Orodja in pripomočki.....	2
2	Teoretični del.....	3
2.1	Zgodovina .....	3
2.1.1	Krajše definicije.....	3
2.1.2	Zgodovina kriptografije .....	4
2.2	Gesla .....	5
2.3	Algoritmi za zaščito gesla po shranjevanju.....	8
3	Osrednji del .....	10
3.1	Okolje Python .....	10
3.1.1	Zgradba Pythona.....	11
3.2	Razvoj .....	12
3.2.1	Zasnova ideje .....	12
3.3	Zasnova ideje .....	13
3.4	Potek .....	14
3.4.1	Prvi program – zakodiranje gesl.....	14
4	Začetek programiranja .....	16
4.1	Analiza .....	27
5	Inženirski dnevnik .....	35
6	Sklepna ugotovitev .....	36
7	Zaključek.....	39
8	Viri in literatura .....	40

## ***Kazalo slik***

Slika 1: LogoTip Pythona.....	11
Slika 2: Poigriv logoTip Pythona. ....	11
Slika 3: Uvoz knjižnic.....	16
Slika 4: Klic funkcije za generiranje gesla. ....	17
Slika 5: Glavni del programa. ....	18
Slika 6: Naključen izbor 5 vprašanj. ....	20
Slika 7: Program za interakcijo z uporabnikom. ....	20
Slika 8: Odstranjevanje presledkov. ....	21
Slika 9: Ustvarjanje osnovnega gesla.....	22
Slika 10: Naključno spreminjanje velikosti črk. ....	22
Slika 11: Naključno spreminjanje velikosti črk. ....	22
Slika 12: Zamenjava posameznih črk s številkami. ....	23
Slika 13: Zamenjava posameznih črk s številkami. ....	23
Slika 14: Izboljšanje gesel. ....	25
Slika 15: Dodajanje velikih črk in posebnih znakov. ....	25
Slika 16: Generiranje naključnih posebnih znakov.....	26
Slika 17: Prikaz končnega gesla. ....	26

## ***Kazalo tabel***

Tabela 1: Primerjava generatorjev gesla.....	32
Tabela 2: Inženirski dnevnik – sledenje spremembam. ....	35

# 1 Uvod

## 1.1 Opis raziskovalnega problema in namen

Namen raziskovalne naloge je preučiti možnosti razvoja programa za generiranje gesel, ki združujejo dve ključni lastnosti: enostavno zapomnljivost za uporabnika in visoko stopnjo odpornosti proti razbitju. Cilj raziskave je bil identificirati najučinkovitejši algoritem ali kriptografsko metodo za ustvarjanje takšnih gesel ter oceniti njihovo varnost v primerjavi z obstoječimi rešitvami.

V okviru naloge sem analiziral različne pristope h generiranju gesel in njihovo odpornost na pogoste metode napadov, kot so groba sila (brute-force) in slovarski napadi (dictionary attacks). Na podlagi teh analiz sem ocenjeval primernost posameznih algoritmov za praktično implementacijo v varnostnih sistemih.

## 1.2 Hipoteze

H1: Daljša gesla so varnejša kot krajša gesla.

H2: Uporaba različnih znakov (številke, velike in male črke, posebni znaki) poveča varnost.

H3: Uporabniki intuitivno izbirajo predvidljive vzorce pri ustvarjanju gesel, medtem ko algoritmično generirana gesla zagotavljajo večjo varnost.

H4: Aplikacijska gesla je težje uganiti in so odpornejša na »brute-force« napade.

H5: Varna gesla pogosto niso uporabniku prijazna, kar vodi v kompromise pri varnosti.

H6: Možna je optimizacija gesel, ki hkrati zagotavlja visoko varnost in uporabnost.

## 1.3 Raziskovalne metode

Pri izdelavi raziskovalne naloge sem uporabljal naslednje metode dela:

- sistematični pregled literature,
- metoda možganske nevihte,
- eksperiment,
- metoda študija primera.

**Sistematični pregled literature** – metoda, s katero celovito preučimo raziskovalno področje in primerjamo svoje ugotovitve z obstoječimi dognanji. V okviru te metode sem analiziral razvoj kriptografskih metod, algoritmov za šifriranje in varnost gesel.

**Metoda možganske nevihte** – omogoča iskanje in raziskovanje različnih problemov ter iskanje inovativnih rešitev. S to metodo sem raziskoval možnosti za razvoj programa, ki bi omogočal generiranje varnih gesel s pomočjo šifrirnih tehnik.

**Eksperiment** – metoda, pri kateri razvijamo in testiramo posamezne dele programa ter odpravljamo morebitne napake. Ključna je bila pri razvoju programske rešitve, saj sem moral preizkusiti različne algoritme za šifriranje gesel in oceniti njihovo učinkovitost glede na varnost in uporabnost. Programiranje je iterativen proces, pri katerem rešitev redko najdemo že v prvem poskusu, zato lahko rečemo, da je razvoj šifrirnih algoritmov pravzaprav oblika eksperimentiranja.

**Metoda študije primera** – analitična metoda, pri kateri podrobno preučujemo posamezne dele programske kode in njihov vpliv na delovanje celotnega sistema. S pomočjo te metode sem analiziral delovanje različnih haširnih algoritmov, kot so SHA-256, bcrypt in Argon2, ter preučil njihovo primernost za varno shranjevanje gesel.

#### 1.4 Orodja in pripomočki

Za izvedbo raziskovalnih metod sem uporabljal naslednje programske rešitve:

Oblak 365 – povezuje Arnes AAI s storitvami Microsoft 365 in omogoča učencem, učiteljem ter drugim delavcem v izobraževalnih organizacijah enostaven dostop do različnih digitalnih orodij. Prav tako skrbnikom omogoča učinkovito upravljanje dostopov in licenc.

Office 365 – paket programske opreme podjetja Microsoft, ki vključuje urejevalnik besedil, planer, orodja za izdelavo predstavitev, shrambo dokumentov in druge funkcije. V raziskovalni nalogi sem ga uporabljal za pisanje, urejanje in organizacijo dokumentov.

OneDrive – Microsoftova storitev za shranjevanje datotek v oblaku, ki omogoča varno hrambo, skupno rabo in dostop do podatkov iz različnih naprav. OneDrive sem uporabljal za varnostno shranjevanje raziskovalnih podatkov in dokumentov.

Word – urejevalnik besedil, ki omogoča enostavno oblikovanje in sodelovanje pri dokumentih. Uporabljal sem ga za pisanje in urejanje raziskovalne naloge.

XMind – programska oprema za izdelavo miselnih vzorcev in možgansko nevihto, ki omogoča vizualizacijo idej, organizacijo kompleksnih informacij in lažje načrtovanje raziskovalnih korakov.

Visual Studio Code – poenostavljen urejevalnik kode s podporo za razvoj programske opreme, odpravljanje napak in nadzor različic. Uporabljal sem ga za razvoj in testiranje programov v okviru raziskave.

Python – visokonivojski programski jezik, ki se pogosto uporablja za razvoj programske opreme, spletnih aplikacij, avtomatizacijo nalog in analizo podatkov. V raziskovalni nalogi sem ga uporabil za razvoj programa za generiranje varnih gesel.

Have I Been Pwned – spletno orodje za preverjanje, ali je bilo določeno geslo že razkrito v preteklih vdorih. Uporabljeno za ozaveščanje o ranljivosti šibkih gesel.

## 2 Teoretični del

V teoretičnem delu sem najprej raziskal zgodovino kriptografije in njen razvoj skozi čas. Nato sem podrobneje predstavil gesla, njihov pomen pri zagotavljanju varnosti ter najpogostejše napake pri njihovi uporabi. Analiziral sem različne algoritme za zaščito gesel pri shranjevanju, med njimi tudi SHA-256, bcrypt in Argon2. Prav tako sem opisal postopek razvoja lastnih programskih rešitev za varno shranjevanje gesel in na koncu primerjal različne metode kodiranja in šifriranja gesel.

### 2.1 Zgodovina

#### 2.1.1 Krajše definicije

**Nomenklator** – šifrirna metoda, pri kateri se določene besede nadomestijo z drugimi besedami, posamezne črke pa z drugimi črkami, s čimer se prikrije vsebina sporočila.

**Tajnost** – lastnost informacije, ki zagotavlja, da ostane skrita širši javnosti. Če je nekaj tajno, do tega nepooblaščenih oseb nimajo dostopa.

**Kriptogram** – zašifrirano ali skrito besedilo, ki ga je mogoče razumeti le s poznavanjem ustrezne šifrirne metode ali ključa.

**Ključ** – niz znakov (beseda, geslo ali druga vrednost), ki se uporablja pri šifriranju in dešifriranju podatkov. Poznavanje ključa omogoča hitrejše in pravilno razvozlanje šifriranega sporočila.

**Šifra** – metoda skrivanja sporočila, pri kateri se vsaka črka ali simbol spremeni po določenem pravilu (ključu), da se prepreči enostavno branje vsebine brez dovoljenja.

**Šifriranje** – postopek preoblikovanja odprtega besedila v zakodirano obliko z uporabo šifrirnih algoritmov, tako da postane neberljivo za nepooblašcene osebe. Obratni proces imenujemo dešifriranje.

**Odprto besedilo** – besedilo, ki ni šifrirano ali zaščiteno. Imenujemo ga tudi prosto besedilo, saj ga lahko bere vsak brez uporabe posebnih metod dešifriranja.

**Haširanje** – enosmerni matematični postopek pretvorbe podatkov (npr. gesla) v niz znakov fiksne dolžine s pomočjo posebnih kriptografskih funkcij, kot so SHA-256, bcrypt in Argon2. Haširanje se pogosto uporablja za varno shranjevanje gesel, saj iz haširane vrednosti prvotnega podatka ni mogoče neposredno pridobiti.

### 2.1.2 Zgodovina kriptografije

Kriptografija – veda o skrivnem pisanju – ima dolgo zgodovino, ki sega v čase pred našim štetjem. Ena najstarejših znanih metod je pripisana Histiju iz Mileta, ki je v 5. stoletju pr. n. št. uporabil izvirno tehniko skrivnega sporočanja tako, da je sužnju na obrito glavo tetoviral sporočilo za Aristagorasa iz Grčije, ki je vsebovalo poziv k uporabi. Ko so lasje ponovno zrastle, je sužnja poslal Aristagorasu z navodilom, naj mu obrije glavo in prebere skrito sporočilo. Ta način prikrivanja informacij je bil učinkovit, dokler sporočilo ni bilo odkrito. Če bi kdo na poti sužnja predčasno obril, bi bilo sporočilo razkrito, kar kaže na pomanjkljivost zgolj fizičnega skrivanja podatkov – enkrat razkrita informacija ni več varna. (Kovačič, 2016).

Podobno se je z vprašanjem varnosti sporočanja v starem Rimu soočal Gaj Julij Cezar. Da bi preveril identiteto svojega vojščaka, sta pred odhodom na bojišče prelomila palico na pol, vsak je vzel svojo polovico, ob ponovnem srečanju pa sta ju staknila skupaj. Če sta se polovici popolnoma prilegali, sta lahko potrdila svojo pripadnost.

V srednjem veku je kriptografija doživela razcvet predvsem zaradi vojaških in cerkvenih potreb. Papež Klement VII. je naročil razvoj t. i. nomenklatorja, ki je vseboval kombinacijo simbolov, števil in pomenov – in se je kot metoda uporabljal skoraj 450 let (Kahn, 1996). Leta 1518 je Johannes Trithemius izdal Polygraphia, prvo tiskano knjigo o kriptografiji, ki velja za začetek sistematičnega obravnavanja šifriranja v pisni obliki (Kovačič, 2016).

Pomemben mejnik v razvoju kriptografije je izum Jeffersonovega šifrnega diska okoli leta 1790, katerega delovanje je bilo osnovano na sistemu kolesc, ki so omogočala mehansko

šifriranje in dešifriranje besedila. V nekoliko spremenjeni obliki ga je med drugo svetovno vojno uporabljala tudi ameriška mornarica.

Sčasoma se je izkazalo, da zgolj skrivanje sporočil ni dovolj, saj lahko nasprotnik razvozla uporabljeno šifrirno metodo. To je pripeljalo do oblikovanja Kerckhoffovega načela, ki pravi, da varnost kriptografske metode ne sme temeljiti na skrivnosti samega algoritma, saj ga lahko napadalec prej ali slej odkrije. Namesto tega mora varnost temeljiti na šifrirnem ključu, ki ostane neznan nasprotniku.

Poleg šifriranja so se v zgodovini uporabljali tudi skrivni jeziki. Če prejemnik ni poznal pravila ali jezika sporočila, ga ni mogel razvozlati. A enkrat, ko napadalec pridobi pravilo ali slovar, tak način izgubi varnost (Kovačič, 2016). To še dodatno potrjuje pomen varovanja ključa kot edine resnične zaščite pred zlorabo.

## 2.2 Gesla

Gesla so niz znakov, ki zagotavljajo zaščito za dostop do sistemov, naprav ali podatkov. Njihova glavna funkcija je preprečevanje nepooblaščenega dostopa in zagotavljanje, da lahko do določenih informacij dostopajo samo osebe z ustreznimi pravicami. Gesla so ena najpogosteje uporabljenih metod avtentifikacije in igrajo ključno vlogo pri varnosti v digitalnem okolju.

Za učinkovito zaščito morajo varna gesla vsebovati različne vrste znakov, in sicer:

1. velike in male črke (A–Z, a–z),
2. številke (0–9),
3. posebne znake (!, @, #, \$, %).

V današnjem digitalnem svetu je močno geslo temelj informacijske varnosti. Slaba gesla so najpogostejši vzrok za krajo spletnih računov, saj jih je mogoče hitro razkriti z napadi s slovarji, avtomatskimi orodji ali preko ranljivosti na spletnih straneh.

Po priporočilih portala Varni na internetu naj gesla vsebujejo vsaj 10 znakov, vključujejo velike in male črke, številke ter posebne znake. Takšna struktura pomembno oteži ugibanje gesla z avtomatskimi metodami (Varni na internetu, n.d.).

Poleg tega strokovnjaki opozarjajo, da:

- enakega gesla ne smemo uporabljati na več spletnih mestih, saj lahko v primeru vdora v eno storitev pride do nepooblaščenega dostopa tudi do drugih uporabnikovih računov;
- je smiselna uporaba upraviteljev gesel, saj ti omogočajo varno shranjevanje kompleksnih gesel, njihovo samodejno vnašanje in obvestila o šibkih ali že zlorabljenih geslih;
- je zelo priporočljivo vključiti dvofaktorsko avtentikacijo (2FA), kjer poleg gesla uporabnik vnese še kodo, poslano na telefon ali generirano v aplikaciji (npr. Google Authenticator);
- naj se uporabniki izogibajo zaporednim znakom in ponavljajočim se vzorcem (npr. »123456«, »asdfgh«), saj so ti najpogosteje uporabljeni in tako najlažje napadeni.

Zanimivo je tudi priporočilo o uporabi stavkov kot gesel – ti so lahko dolgi in kompleksni, a hkrati bolj zapomnljivi.

**Primer:**

»Po kaj gre mali Petja 2x tedensko na Slomškovo 7?«

Lahko postane: PkgmP2xtnS7? (Varni na internetu, n.d.).

S pomočjo teh tehnik lahko uporabniki bistveno povečajo varnost svojih računov in zmanjšajo možnost zlorabe svojih podatkov.

Za povečanje varnosti je ključno ustvarjati močna, naključna in unikatna gesla, ki vsebujejo kombinacijo velikih in malih črk, števil in posebnih znakov, ter so dovolj dolga (priporočeno več kot 12 znakov), kar otežuje njegovo ugibanje ali razbitje z avtomatiziranimi napadi.

Eden najučinkovitejših načinov za ustvarjanje močnega gesla je uporaba avtomatiziranih orodij, ki generirajo gesla visoke entropije. Primer take rešitve je Googlovo samodejno generiranje gesel, ki uporabniku med prijavo ali registracijo samodejno predlaga naključno ustvarjeno geslo. Ta gesla so običajno sestavljena iz dolgih nizov črk, števil in posebnih znakov ter se varno shranijo v Googlovem upravitelju gesel (Google Password Manager), kjer so zaščitena z glavnim Google računom in dodatno varnostno plastjo, kot je dvofaktorska avtentikacija (Google Help, n.d.).

**Primer Googlovega močnega gesla:** wS#1LZ7!tRVfQx9\$

Kljub tem priporočilom številni uporabniki še vedno uporabljajo šibka ali ponavljajoča se gesla, kar povečuje tveganje za nepooblaščen dostop. Najpogostejše napake pri uporabi gesel:

### **Uporaba prekratkih ali preprostih gesel**

Šibka gesla, kot so "123456", "password", "qwerty", "111111" ali celo uporabniško ime, predstavljajo veliko varnostno tveganje. Napadalci pogosto uporabljajo zbirke že razkritih gesel, imenovane slovarske baze (dictionary lists), ki vsebujejo milijone najpogosteje uporabljenih gesel. Te baze so rezultat podatkovnih vdorov v različne spletne storitve. Pri napadu napadalec najprej preveri, ali se geslo nahaja v takšni zbirki, s čimer močno poveča možnost uspešnega vdora.

**Primer:** Spletna storitev Have I Been Pwned omogoča preverjanje, ali je bilo določeno geslo že razkrito v preteklih podatkovnih vdorih. Če se geslo nahaja v bazi kompromitiranih gesel, obstaja velika verjetnost, da ga napadalci že poznajo in ga lahko uporabijo pri slovarskih napadih (dictionary attacks) ali credential stuffing napadih, kjer sistematično preizkušajo pridobljena gesla na različnih spletnih straneh. Zato je ključnega pomena, da uporabniki ne uporabljajo gesel, ki so bila že razkrita, in da redno preverjajo varnost svojih gesel s pomočjo orodij, kot je Have I Been Pwned.

### **Uporaba istega gesla za več računov**

Ponovna uporaba gesla na več platformah predstavlja veliko varnostno tveganje. Če napadalec pridobi geslo iz manj varne storitve, ga lahko uporabi za dostop do bolj občutljivih računov, kot so e-poštni naslovi, spletne banke ali družbena omrežja. Ta vrsta napada se imenuje credential stuffing, pri katerem napadalci sistematično preverjajo ukradena gesla na različnih spletnih straneh. Poleg tega shranjevanje gesel v nezaščitenih besedilnih datotekah ali na listih papirja ni varno, saj lahko do teh informacij pride vsakdo, ki ima fizični ali digitalni dostop do njih.

### **Shranjevanje gesel v nešifrirani obliki**

Nekateri uporabniki gesla zapisujejo na papir ali jih shranjujejo v nešifrirane besedilne datoteke (npr. *gesla.txt*) na računalniku. To pomeni, da lahko vsak, ki pridobi dostop do naprave, vidi vsa shranjena gesla, kar povečuje tveganje za krajo občutljivih podatkov. Priporočljiva je uporaba upravljalnikov gesel (password managers), ki omogočajo varno shranjevanje gesel in generiranje močnih gesel.

## Neuporaba dvofaktorske avtentikacije (2FA)

Dvofaktorska avtentikacija (2FA) bistveno poveča varnost gesel, saj ob prijavi zahteva dodatno preverjanje identitete, na primer prek enkratnega gesla (OTP), poslanega na telefon ali e-pošto. Tudi če napadalec pridobi geslo, mu 2FA lahko prepreči dostop do računa. Kljub njeni učinkovitosti številni uporabniki te funkcije ne aktivirajo, s čimer povečujejo svojo ranljivost za kibernetike napade.

Za večjo varnost je priporočljivo uporabljati dolgoročno močna gesla, ki niso predvidljiva, in jih zaščititi z večfaktorsko avtentikacijo ter varnimi metodami shranjevanja.

### 2.3 Algoritmi za zaščito gesla po shranjevanju

Algoritmi za zaščito gesel so posebej zasnovani za varno shranjevanje gesel v sistemih. Njihova glavna naloga je zaščita podatkov pred nepooblaščenim dostopom, pri čemer uporabljajo tehnike, kot so haširanje (pretvorba gesla v enosmerni niz) in soljenje (dodajanje naključnih podatkov pred haširanjem). Ti algoritmi so ključni za zagotavljanje varnosti gesel, saj shranjevanje gesel v običajni besedilni obliki predstavlja veliko varnostno tveganje.

Vrste algoritmov za zaščito gesel po shranjevanju:

#### **SHA-256 (Secure Hash Algorithm 256-bit)**

**Opis:** SHA-256 je ena najpogosteje uporabljenih kriptografskih haš funkcij, ki izračuna 256-bitno haš vrednost. Pogosto se uporablja za preverjanje integritete podatkov.

**Slabosti:** Samo haširanje gesel ni dovolj varno, saj gre za hitro metodo, ki je ranljiva na napade **brute-force** in **rainbow table** napade.

**Uporaba:** Primarno se uporablja za preverjanje podatkov, vendar ni priporočljiv za shranjevanje gesel brez dodatnih zaščit, kot sta soljenje in uporaba počasnejših algoritmov.

#### **Bcrypt**

**Opis:** Bcrypt je izboljšana hash funkcija, zasnovana posebej za zaščito gesel. Uporablja **soljenje** in omogoča nastavitve "**cost**" **faktorja**, ki poveča čas izvedbe in s tem zaščito pred napadi z uporabo zmogljive strojne opreme (npr. GPU).

**Prednosti:** Zaradi počasnejše obdelave je odpornejši na **brute-force** in **rainbow table** napade.

**Uporaba:** Pogosto se uporablja v sodobnih spletnih aplikacijah za varno shranjevanje gesel.

## **PBKDF2 (Password-Based Key Derivation Function 2)**

**Opis:** PBKDF2 večkrat hašira uporabniško geslo skupaj z naključnim "solom", s čimer oteži napade in zmanjša možnost uspešnega razbijanja gesla.

**Prednosti:** Večkratno ponavljanje haširanja (tisoče ali celo milijone iteracij) poveča varnost. Upočasni napade, tudi če napadalci uporabljajo zmogljivo strojno opremo.

**Uporaba:** Uporablja se v varnostnih sistemih, kot so **upravitelji gesel, šifriranje diskov** in avtentikacijski sistemi.

## **Argon2**

**Opis:** Argon2 je sodoben algoritem za zaščito gesel, ki je zmagal na tekmovanju **Password Hashing Competition (PHC)**. Njegova glavna prednost je možnost prilagoditve **porabe pomnilnika, časovne zahtevnosti in stopnje vzporednega izvajanja**.

**Prednosti:** Argon2 je zelo odporen proti napadom s sodobnimi strojno pospešenimi napravami (npr. GPU, FPGA). Omogoča prilagoditev zahtevnosti algoritma glede na računalniške zmogljivosti.

**Uporaba:** Priporočen za sodobne varnostne rešitve, kot so **sistemi za avtentikacijo, upravitelji gesel in shranjevanje občutljivih podatkov**.

## **Scrypt**

**Opis:** Scrypt je derivacijska funkcija ključa, podobna PBKDF2, ki uporablja povečano porabo pomnilnika za dodatno zaščito.

**Prednosti:** Zahteva veliko pomnilniških virov, kar otežuje napade s specializiranimi napravami, kot so **FPGA** ali **GPU**.

**Uporaba:** Pogosto se uporablja v kriptovalutah (npr. Litecoin) in sistemih za varno shranjevanje gesel.

Izbira ustreznega algoritma za zaščito gesel je ključnega pomena za zagotavljanje varnosti v digitalnem svetu. Medtem ko SHA-256 sam po sebi ni primeren za varno shranjevanje gesel, so algoritmi, kot so bcrypt, PBKDF2, Argon2 in scrypt, prilagojeni prav temu namenu, saj ponujajo dodatne varnostne mehanizme, kot sta soljenje in povečana računalniška zahtevnost. Argon2 je trenutno eden najučinkovitejših algoritmov za zaščito gesel in je priporočljiv za uporabo v novih varnostnih sistemih.

## 3 Osrednji del

### 3.1 Okolje Python

Python je visokonivojski, interpretiran in večnamenski programski jezik, ki je priljubljen zaradi svoje berljive sintakse, podobne angleškemu jeziku, ter bogatega nabora knjižnic za različne aplikacije, vključno z analizo podatkov, spletnim razvojem in umetno inteligenco.

Jezik je razvil nizozemski programer Guido van Rossum leta 1991 kot odgovor na omejitve programskega jezika ABC. Ime Python ni povezano s kačo, temveč z britansko humoristično televizijsko oddajo *Leteči cirkus Montyja Pythona*. Python je bil uradno izdan leta 1994 in je od takrat doživel več nadgradenj (Van Rossum, 1999).

Pomembne različice vključujejo:

- **Python 1.0 (1994)** – uvedel module in izjeme,
- **Python 2.0 (2000)** – prinesel podpore Unicode in zbiralnik smeti,
- **Python 3.0 (2008)** – uvedel številne spremembe, ki niso združljive s prejšnjimi različicami.

Do leta 2020 je Python postal eden najbolj priljubljenih programskih jezikov na svetu, s čimer se je uvrstil ob bok Javi, C++ in JavaScriptu. Njegova odprtokodna narava in velika skupnost razvijalcev omogočata hitro rast knjižnic in orodij.

Python je odprtokoden in ima veliko podporo skupnosti, kar omogoča hiter razvoj ter razširitev z več tisoč knjižnicami. Python Software Foundation ga opisuje kot "interpretiran, objektno usmerjen programski jezik na visoki ravni z dinamično semantiko."

Ena izmed največjih prednosti Pythona je interpretacija kode, kar pomeni, da programska koda ne potrebuje prevajalnika in se lahko izvaja neposredno. Python podpira več programskih paradig, vključno z:

- objektno usmerjenim programiranjem (OOP) – omogoča strukturiranje kode okoli objektov in njihovih lastnosti;
- proceduralnim programiranjem – uporablja zaporedje ukazov za izvajanje nalog;
- funkcionalnim programiranjem – omogoča pisanje funkcij, ki delujejo kot samostojne enote brez stranskih učinkov.

Python je večplatformski jezik, ki omogoča izvajanje aplikacij na Windows, macOS in Linux brez potrebe po ponovni kompilaciji. Širok nabor standardnih knjižnic omogoča uporabo jezika na različnih področjih, kot so strojno učenje, podatkovna znanost, spletni razvoj, avtomatizacija opravil in računalniško omrežje. Knjižnice tretjih oseb še dodatno širijo njegove zmogljivosti in omogočajo njegovo uporabo v različnih domenah, od znanstvenih raziskav do razvoja kompleksnih aplikacij.

Njegova priljubljenost izhaja iz berljive sintakse, ki izboljša produktivnost razvijalcev, saj omogoča hitro učenje in pisanje razumljive kode. Python tako ostaja eden najpogosteje uporabljenih jezikov v sodobnem programiranju.

### 3.1.1 Zgradba Pythona

Program v Pythonu je sestavljen iz zaporedja ukazov, ki se običajno pišejo v posamezne vrstice. Konec vrstice označuje konec ukaza, razen v primerih, kjer je ukaz daljši in ga želimo nadaljevati v naslednji vrstici – v tem primeru uporabimo poševnico (`\`).

Več ukazov lahko zapišemo v eno vrstico, če jih ločimo s podpičjem (;), vendar ta način ni priporočljiv zaradi slabše berljivosti kode.

Nekateri ukazi so sestavljeni iz glave in telesa:

- Glava je ukaz, ki se konča z dvopičjem (:).
- Telo vsebuje ukaze, ki so zamaknjeni za enako število presledkov (priporočljivo štiri presledke).

Sestavljeni ukazi lahko vsebujejo gnezdene ukaze, ki so dodatno zamaknjeni, kar omogoča boljšo strukturo kode.

Python prav tako omogoča pisanje komentarjev, ki služijo kot pojasnila v kodi. Vse, kar se nahaja za znakom #, interpretator ignorira. Komentarji so koristni za dokumentiranje delovanja programa in lažje vzdrževanje kode. Pythonov sistem zamikov namesto oklepajev izboljšuje berljivost kode in olajša strukturiranje programov.



Slika 1: LogoTip Pythona.



Slika 2: Poigriv logoTip Pythona.

## 3.2 Razvoj

### 3.2.1 Zasnova ideje

Preden razložim postopek razvoja ideje, je pomembno omeniti, da se je moja raziskovalna naloga trikrat spremenila, saj sem postopoma ugotavljal, da je prvotno izbrana tema preobsežna. Posledično sem moral dvakrat redefinirati raziskovalni problem, da bi bila raziskava bolj osredotočena in izvedljiva (Vsi programi so bili razviti in preizkušeni v okolju Visual Studio Code.).

Ko sem določil raziskovalno vprašanje, sem moral sprejeti dve ključni odločitvi:

1. Kateri programski jezik bom uporabil?
2. Kako bom oblikoval kodo in razvijal rešitev?

Za razvoj sem lahko izbiral med različnimi programskimi jeziki, kot so Java, JavaScript, C, C++, C# in Python. Med vsemi možnostmi sem se odločil za Python, saj:

- je enostaven za učenje in uporabo, kar omogoča hitrejši razvoj;
- je priljubljen na različnih področjih, kot so spletni razvoj, avtomatizacija, analiza in vizualizacija podatkov;
- je široko uporabljen v industriji, vključno s podjetji, kot so Google, Microsoft in Meta;
- ima bogato zbirko knjižnic, ki olajšajo razvoj različnih aplikacij.

Ko sem določil programski jezik, sem moral sprejeti še eno odločitev – način izvedbe programa. Na voljo sem imel dve možnosti:

1. Uporaba knjižnic/modulov (npr. Turtle za vizualizacijo), kar bi omogočilo vizualno privlačnejši končni izdelek, a bi hkrati prineslo tudi nekatere slabosti:
  - manj pregledna koda,
  - zahtevnejša analiza,
  - počasnejše izvajanje,
  - večja možnost napak pri generiranju in delovanju programa.
2. Izvajanje programa v konzolnem okolju, kar pomeni:
  - manj privlačen vizualni prikaz,
  - bolj berljivo in pregledno kodo,
  - lažjo analizo in hitrejše izvajanje,

- manj možnosti za napake pri generiranju in izvajanju.

Po preučitvi prednosti in slabosti sem se odločil za konzolno izvedbo programa, saj mi je omogočila večjo preglednost in učinkovitost kode, kar je bilo ključno za analizo in razvoj algoritmov.

### 3.3 Zasnova ideje

Ideja za to raziskovalno nalogo se je razvila med poletjem 2024, ko sem se prvič neposredno soočil s posledicami kibernetских napadov. Prišlo je do obsežnega vdora v različne sisteme, pri čemer je bilo razkritih na tisoče gesel – med njimi tudi moje Google geslo. Ta dogodek me je prisilil v takojšnjo spremembo gesla in razmislek o tem, kako pogosto uporabniki podcenjujejo pomen varnosti svojih prijavnih podatkov.

Nekaj tednov kasneje sem prejel elektronsko sporočilo s priloženo datoteko, ki se je izkazala za okuženo z virusom Trojan Horse. Ta vrsta zlonamerne programske opreme napadalcem omogoča prevzem nadzora nad napravo. Na srečo sem grožnjo pravočasno prepoznal, odstranil virus in preprečil večjo škodo. To me je pripeljalo do ključnega vprašanja:

Če lahko takšen incident prizadene mene, ki imam osnovno znanje o računalniški varnosti, kakšne posledice ima lahko za uporabnike z manj izkušnjami?

Po nadaljnjem raziskovanju sem naletel na strokovne članke, ki so opozarjali, da je eden največjih varnostnih problemov ponavljajoča se uporaba šibkih in predvidljivih gesel ter njihova delitev med več računi. Uporabniki pogosto izbirajo gesla, ki so enostavna za pomnjenje, a ravno zaradi tega lahko predvidljiva in ranljiva za napade. To me je pripeljalo do osrednjega raziskovalnega vprašanja:

Ali so gesla, ki jih priporočajo sistemi, pretežka za zapomniti? Bi bilo mogoče razviti program, ki bi ustvarjal gesla, ki so hkrati varna in enostavna za uporabnika?

Ob tem razmišljanju sem se odločil, da bom v okviru raziskovalne naloge poskusil razviti programsko rešitev, ki omogoča:

- preverjanje moči uporabniško izbranih gesel;
- generiranje prilagojenih gesel, ki so težko razbitljiva, a enostavna za pomnjenje;
- zagotavljanje, da generirana gesla niso shranjena ali poslana tretjim osebam.

Za realizacijo sem izbral programski jezik Python, saj ponuja široko paleto kriptografskih knjižnic in orodij za obdelavo podatkov. Poleg tega sem za analizo uporabil zbirko

najpogosteje uporabljenih gesel iz preteklih varnostnih incidentov, da bi preučil najpogostejše vzorce v šibkih geslih.

Med razvojem sem ugotovil, da ena sama rešitev ni dovolj za vse možne scenarije, zato sem moral razviti več različnih programov. Implementiral sem različne algoritme za zaščito gesel, kot so:

- SHA-256 – standardni kriptografski hash algoritem
- bcrypt – priljubljena metoda za varno shranjevanje gesel
- Argon2 – eden najsodobnejših algoritmov za zaščito gesel.

V določenih primerih obstoječe metode niso zagotavljale optimalne kombinacije varnosti in uporabnosti, zato sem bil primoran razviti lasten algoritem, ki je prilagojen ciljem te raziskave. Moj cilj je bil ustvariti rešitev, ki bi omogočala varno in hkrati uporabniku prijazno generiranje gesel, kar je še vedno eden največjih izzivov na področju računalniške varnosti.

### 3.4 Potek

V prvem delu bom predstavil postopek razvoja programa, ki deluje kot generator varnih gesel, prilagojenih posameznemu uporabniku. Program bo gesla ustvarjal na podlagi uporabnikovih odgovorov na specifična vprašanja, kar bo omogočilo oblikovanje gesel, ki so hkrati enostavna za zapomniti in odporna proti napadom.

Na koncu bom s pomočjo analize ocenil, ali sem dejansko poenostavil proces generiranja gesel, ki so intuitivna za pomnjenje, a hkrati dovolj kompleksna, da zagotavljajo visoko raven varnosti in odpornost proti napadom, kot sta brute-force in slovarski napadi.

#### 3.4.1 Prvi program – zakodiranje gesl

Cilj prvega programa je razvoj algoritma za generiranje gesel, ki so prilagojena uporabniku, a hkrati dovolj varna za zaščito pred napadi. Program združuje personalizacijo in naključne elemente, kar zagotavlja optimalno ravnovesje med uporabnostjo in varnostjo.

#### 1. Struktura programa

Program je zasnovan v več fazah:

- Inicializacija potrebnih knjižnic – uporaba ustreznih Python knjižnic za delo z naključnimi vrednostmi, nizi in varnostnimi algoritmi.

- Interakcija z uporabnikom – program uporabniku postavi niz naključnih vprašanj, povezanih z njegovim življenjem, kot so:
  - Kako je ime tvojemu prvemu hišnemu ljubljencu?
  - Kje si se rodil?
  - Katero številko si imel na dresu v šoli?
- Oblikovanje osnovnega gesla – program sestavi začetno besedilno geslo na podlagi uporabnikovih odgovorov.
- Dodatna zaščita gesla – program obdelano geslo izboljša z dodatnimi varnostnimi elementi:
  - velike in male črke (Rex → ReX),
  - številke in posebni znaki (Rex → ReX7#),
  - zamenjave določenih črk s številkami (Maribor → M4r1b0r).
- Prikaz generiranega gesla – program uporabniku prikaže končno geslo in poda oceno njegove varnosti.

## 2. Primer delovanja programa

Uporabnik odgovori na naslednja vprašanja:

- Ime prvega hišnega ljubljénčka → Rex
- Kje si se rodil? → Maribor
- Katero številko si imel na dresu? → 7
- Katerega meseca si bil rojen? → Maj

Osnovno geslo: RexMaribor7Maj

Obdelano geslo po algoritmu: ReX#M4r1b0r7m4J!

Program omogoča generiranje gesel, ki so še vedno povezana z uporabnikovimi odgovori, vendar so zaradi dodatne obdelave znatno težje razbitljiva s standardnimi napadi, kot so brute-force ali slovarski napadi. S tem se izboljša varnost gesel, hkrati pa ohrani njihova uporabnost in zapomljivost.

## 4 Začetek programiranja

Začetek programiranja sem začel s prvim programom, ki je zasnovan tako, da omogoča generiranje osebno prilagojenega, a varnega gesla. Program temelji na uporabnikovih odgovorih na niz vprašanj, ki se nanašajo na njegove osebne podatke. Na ta način se ustvari geslo, ki je uporabniku enostavno za pomnjenje, hkrati pa vključuje varnostne elemente, zaradi katerih je odporno proti napadam.

```
import random
import string
```

Slika 3: Uvoz knjižnic.

Ta vrstica kode uvaža dve vgrajeni knjižnici v Pythonu: `random` in `string`. Obe knjižnici sta ključni za delovanje programa, saj omogočata delo z naključnimi vrednostmi in znakovnimi nizi.

**Knjižnica `random`** – Pythonov modul, ki omogoča:

- generiranje naključnih vrednosti,
- izbiro naključnih elementov iz seznamov,
- ustvarjanje naključnih števil ter
- odločanje na podlagi verjetnosti.

V tem programu ima ključno vlogo, saj omogoča:

- naključno spreminjanje črk v geslu (velike/male črke),
- dodajanje naključnih posebnih znakov in
- zamenjavo določenih črk s številkami (npr. "o" → "0", "a" → "4").

**Knjižnica `string`** – vsebuje uporabne nize znakov, kot so:

- velike in male črke (`string.ascii_uppercase`, `string.ascii_lowercase`),
- številk (`string.digits`),
- posebni znaki (`string.punctuation`).

V tem programu se uporablja za dostop do predpripravljenih nizov znakov, kar omogoča bolj dinamično in učinkovito generiranje gesel.

Brez `random` in `string` bi bilo treba ročno določiti sezname znakov, kar bi bilo zamudno in manj učinkovito. Z njuno uporabo program deluje dinamično, hitreje in optimalno, kar omogoča boljše varnostne lastnosti generiranih gesel.

```
# Klic funkcije za ustvarjanje gesla  
create_password()
```

Slika 4: Klic funkcije za generiranje gesla.

Ta vrstica kode kliče funkcijo `create_password()`, kar pomeni, da se ob zagonu programa sproži proces generiranja gesla. Funkcija `create_password()` je osrednji del programa, saj vodi celoten postopek ustvarjanja varnega in prilagojenega gesla. Ključni koraki funkcije so:

- Naključna izbira vprašanj – program vsebuje seznam 30 vprašanj, iz katerih naključno izbere 5. Vprašanja so oblikovana tako, da uporabniku pomagajo izbrati osebno, a hkrati nepredvidljivo geslo.
- Uporabnikovi odgovori – program uporabniku prikaže izbrana vprašanja in počaka na vnos odgovorov. Vsak odgovor se shrani in obdela.
- Odstranjevanje presledkov – da bi bilo geslo kompaktno in težje predvidljivo, program iz odgovorov odstrani vse presledke.
- Ustvarjanje osnovnega gesla – združeni odgovori tvorijo osnovo gesla, ki pa je še vedno preveč preprosta za varno uporabo.
- Spreminjanje velikosti črk – vsaka črka v osnovnem geslu ima 50-odstotno možnost, da se spremeni v veliko ali malo črko, kar povečuje naključnost.
- Zamenjava določenih črk s številkami – program preveri, ali geslo vsebuje pogoste črke (npr. "e", "o", "a") in jih naključno nadomesti s številkami ("e" → "3", "o" → "0", "a" → "4" itd.), kar oteži napade z besednimi slovarji (dictionary attacks).
- Dodajanje dodatnih varnostnih elementov – program na koncu geslu doda:
  - naključno veliko črko
  - naključno številk
  - naključni poseben znak (!@#\$%^&...\*)

To dodatno poveča zapletenost gesla in ga naredi odpornejšega proti napadom.

- Prikaz končnega gesla – program uporabniku izpiše novo, izboljšano geslo, ki je bistveno varnejše kot osnovno besedilo in hkrati lažje za pomnjenje kot popolnoma naključno geslo.

Brez tega klica se funkcija `create_password()` ne bi nikoli izvedla, kar pomeni, da geslo ne bi bilo ustvarjeno in program ne bi imel nobenega vidnega učinka. Ta ukaz sproži celoten mehanizem programa in omogoča, da uporabnik prejme osebno prilagojeno, a varno geslo.

```

# Glavna funkcija za ustvarjanje gesla
def create_password():
    # Seznam vseh 30 vprašanj
    all_questions = [
        "Kako je ime tvojemu prvemu hišnemu ljubljenu?",
        "Kje si se rodil?",
        "Katero številko si imel na dresu v šoli?",
        "Kateri mesec si bil rojen?",
        "Kateri je tvoj najljubši film?",
        "Kateri je tvoj najljubši glasbeni izvajalec?",
        "Katero knjigo si prebral kot prvič?",
        "Kje si preživel svoj zadnji dopust?",
        "Kateri je tvoj najljubši šport?",
        "Kakšna je bila tvoja prva služba?",
        "Kateri je tvoj najljubši barvni odtenek?",
        "Kje si spoznal svoje najboljše prijatelje?",
        "Kateri je tvoj najljubši kraj za obisk?",
        "Katero živali bi želel imeti?",
        "Kateri je tvoj najljubši film iz otroštva?",
        "Kateri je tvoj najljubši umetnik?",
        "Kakšno je ime tvoje osnovne šole?",
        "Kako se imenuje tvoja najboljša prijateljica?",
        "Kaj je tvoja najljubša pijača?",
        "Katero je tvoje najljubše letno obdobje?",
        "Kako je ime tvoje sestre/brata?",
        "Katero je tvoje najljubše mesto na svetu?",
        "Kateri je tvoj najljubši počitniški kraj?",
        "Katera pesem te najbolj spomni na otroštvo?",
        "Kateri je tvoj najljubši predmet v šoli?",
        "Kako se imenuje tvoja prva ljubica?",
        "Kakšno ime ima tvoj prvi avto?",
        "Kateri je tvoj najljubši način preživljanja prostega časa?",
        "Kateri je tvoj najljubši športni klub?",
        "Katera hrana je tvoja najljubša?",
        "Kdo je tvoj največji vzornik?"
    ]

    # Naključno izberemo 5 vprašanj iz vseh 30
    selected_questions = random.sample(all_questions, 5)

```

Slika 5: Glavni del programa.

Ta del kode definira glavno funkcijo `create_password()`, ki je odgovorna za generiranje varnega in prilagojenega gesla. V začetnem delu funkcije program ustvari seznam 30 različnih vprašanj, ki jih bo kasneje uporabil za oblikovanje gesla.

Seznam all\_questions vsebuje osebna, a ne preveč predvidljiva vprašanja, kar povečuje unikatnost in varnost gesla. Ta vprašanja pokrivajo širok spekter osebnih podatkov, med drugim:

- osebne izkušnje (prvi hišni ljubljencek, prvo delovno mesto)
- osebne preference (najljubši film, glasbeni izvajalec, knjiga)
- družinske odnose (ime sestre/brata, prva ljubezen)
- lokacije (rojstni kraj, kraj zadnjega dopusta).

Uporabnikovi odgovori na ta vprašanja se uporabijo kot osnova za geslo, saj so unikatni za vsakega posameznika.

Namen teh vprašanj je uporabiti uporabnikove odgovore kot osnovo za geslo, s čimer se zagotovi prilagodljivost in edinstvenost gesla za vsakega posameznika. Ker vsak uporabnik poda različne odgovore, bo končno geslo unikatno in težko uganljivo, saj ne bo sledilo predvidljivim vzorcem, ki jih pogosto izkoriščajo napadalci pri slovarskih in brute-force napadih.

S tem pristopom program omogoča ustvarjanje gesel, ki so hkrati osebno relevantna in varna, saj kombinacija osebnih podatkov in dodatnih naključnih elementov preprečuje preprosto ugibanje gesla.

Ta postopek nam omogoča:

- Edinstvenost gesel – ker vsak uporabnik poda različne odgovore, je vsako generirano geslo unikatno.
- Težjo ugibljevost – kombinacija osebnih odgovorov in naključnih sprememb otežuje napade s slovarskimi bazami (dictionary attacks).
- Lažjo zapomnljivost – geslo temelji na osebnih podatkih, zato si ga uporabnik lažje zapomni, hkrati pa je dovolj zaščiteno pred napadi.

Brez tega seznama vprašanj program ne bi mogel ustvariti gesla, saj se odgovori uporabnika uporabljajo kot ključni elementi pri oblikovanju osnovnega niza znakov. Program nato ta niz dodatno preoblikuje s pomočjo:

- spreminjanja velikosti črk (npr. "maribor" → "MArIbOr")
- zamenjav črk s številkami (e → 3, o → 0, a → 4)
- dodajanja posebnih znakov (npr. "Maribor" → "M4r1b0r@7").

S tem se zagotovi, da je končno geslo unikatno, nepredvidljivo in odporno na standardne napade, kot so slovarski napadi (dictionary attacks) in brute-force napadi.

```
# Naključno izberemo 5 vprašanj iz vseh 30
selected_questions = random.sample(all_questions, 5)
```

Slika 6: Naključen izbor 5 vprašanj.

Ta vrstica kode naključno izbere podseznam petih vprašanj iz celotnega seznama 30 vprašanj z uporabo funkcije `random.sample()`, ki omogoča izbiro podseznama določenega števila elementov brez ponavljanja. To pomeni, da bo vsak niz vprašanj pri vsakem zagonu programa drugačen, saj se ista vprašanja ne morejo ponoviti v istem podseznamu.

S tem pristopom program zagotavlja, da so uporabnikova gesla vsakič unikatna, kar povečuje njihovo nepredvidljivost in varnost. Naključna izbira vprašanj zmanjšuje možnost, da bi gesla sledila ponavljajočim se vzorcem, zaradi česar bi postala ranljiva na napade s slovarjem (dictionary attacks) ali brute-force napade.

Funkcija `random.sample()` omogoča ustvarjanje dinamičnih podseznamov vprašanj, kar pomeni, da vsaka generacija gesla temelji na različnih kombinacijah osebnih podatkov uporabnika. S tem se poveča personalizacija gesel, saj uporabnik dobi vprašanja, ki so specifična zanj, hkrati pa ostanejo dovolj raznolika, da zagotavljajo visoko stopnjo varnosti.

Z uporabo `random.sample()` in podseznama vprašanj program zagotavlja, da je vsako generirano geslo unikatno, varno in uporabniku prilagojeno, kar povečuje ravnovesje med uporabnostjo in varnostjo.

```
# Zbiranje odgovorov uporabnika
answers = []
for question in selected_questions:
    answer = input(question + " ")
    # Odstranimo presledke iz odgovora
    answer = remove_spaces(answer)
    answers.append(answer)
```

Slika 7: Program za interakcijo z uporabnikom.

Ta del kode skrbi za interakcijo z uporabnikom in zbiranje njegovih odgovorov, ki se kasneje uporabijo za generiranje prilagojenega gesla. Program uporabniku prikaže izbrana vprašanja, shrani njegove odgovore ter odstrani presledke, da zagotovi kompaktno in varno obliko gesla.

Najprej program ustvari prazen seznam answers, kamor bo shranjeval uporabnikove odgovore. Nato s pomočjo zanke iterira skozi vseh pet naključno izbranih vprašanj iz seznama selected\_questions. Za vsako vprašanje program uporabniku prikaže poziv za vnos podatkov, uporabnik vnese svoj odgovor, ki se nato shrani v spremenljivko answer.

Po vnosu odgovora program pokliče funkcijo remove\_spaces(answer), ki odstrani vse presledke. Če uporabnik na primer vnese "MojaMuca", se odgovor ne spremeni, če pa vnese "Moja Muca", se presledek odstrani, rezultat pa postane "MojaMuca". Obdelan odgovor se nato doda v seznam answers. Ko uporabnik odgovori na vseh pet vprašanj, seznam vsebuje pet odgovorov, ki služijo kot osnova za nadaljnjo obdelavo gesla.

Na primer, če so bila vprašanja "Kako je ime tvojemu prvemu hišnemu ljubljenu?", "Kje si se rodil?", "Katera hrana je tvoja najljubša?", "Katere znamke je bil tvoj prvi avto?" in "Kdo je tvoj največji vzornik?", uporabnik pa je vnesel odgovore "Bobi", "Ljubljana", "Pica", "Golf 2" in "Tesla", bo po odstranitvi presledkov seznam answers videti takole: ["Bobi", "Ljubljana", "Pica", "Golf2", "Tesla"].

Ta korak je ključnega pomena, saj omogoča prilagojeno geslo na podlagi osebnih odgovorov uporabnika, odstrani presledke, da geslo ostane kompaktno, in ustvari seznam odgovorov, ki služi kot osnova za nadaljnjo obdelavo gesla. Tako generirano geslo temelji na uporabnikovih osebnih podatkih, a ostaja nepredvidljivo, kar izboljšuje njegovo varnost in uporabnost.

```
# Funkcija za odstranjevanje presledkov iz odgovorov
def remove_spaces(answer):
    return answer.replace(" ", "")
```

Slika 8: Odstranjevanje presledkov.

Funkcija remove\_spaces(answer) igra pomembno vlogo pri zagotavljanju čistosti in varnosti gesla, saj odstrani vse presledke iz uporabnikovih odgovorov. Ko prejme niz answer, ki predstavlja uporabnikov vnos, uporabi metodo .replace(" ", ""), s katero vse pojavitve presledkov nadomesti z ničelnim nizom. Na ta način zagotovi, da v končnem geslu ni nepotrebnih presledkov, ki bi lahko vplivali na njegovo varnost ali povzročili težave pri uporabi v različnih sistemih, kjer presledki niso dovoljeni.

Odstranjevanje presledkov je pomembno iz več razlogov. Prisotnost presledkov lahko zmanjša varnost gesla, saj lahko napadalci prepoznajo vzorce in lažje razbijejo geslo. Prav tako lahko določeni sistemi za preverjanje gesel ne podpirajo presledkov, kar lahko povzroči

težave pri prijavi. Poleg tega odstranitev presledkov omogoča lažje združevanje gesel s posebnimi znaki in številkami, kar izboljša njihovo kompleksnost.

Po odstranitvi presledkov funkcija vrne prečiščen niz, ki je pripravljen za nadaljnjo obdelavo pri generiranju gesla. Tako se zagotovi, da je geslo kompaktno, varno in združljivo s čim več avtentikacijskimi sistemi.

```
# Ustvarjanje osnovnega gesla
base_password = ''.join(answers)
```

Slika 9: Ustvarjanje osnovnega gesla.

Ta vrstica kode združi vse uporabnikove odgovore v enoten niz, s čimer ustvari osnovno geslo. To pomeni, da se posamezni odgovori, ki jih je uporabnik vnesel na podana vprašanja, združijo v neprekinjen niz znakov, ki nato predstavlja prvo različico gesla.

```
# Naključno spreminjanje velikosti črk
password_with_random_case = random_case_change(base_password)
```

Slika 10: Naključno spreminjanje velikosti črk.

Ta vrstica kode pokliče funkcijo `random_case_change(base_password)`, ki v osnovnem geslu naključno spremeni velikost črk. To pomeni, da ima vsaka črka v geslu naključno možnost, da se pretvori v veliko ali malo črko, kar povečuje nepredvidljivost gesla.

```
# Funkcija za naključno spreminjanje velikosti črk (male v velike in obratno)
def random_case_change(password):
    new_password = ""
    for char in password:
        # Naključno odločimo, ali bomo črko spremenili v veliko ali majhno
        if char.isalpha() and random.choice([True, False]):
            if char.islower():
                new_password += char.upper()
            else:
                new_password += char.lower()
        else:
            new_password += char # Ohranimo število ali poseben znak nespremenjen
    return new_password
```

Slika 11: Naključno spreminjanje velikosti črk.

Funkcija `random_case_change(password)` je zasnovana tako, da naključno spreminja velikost črk v geslu, s čimer povečuje njegovo nepredvidljivost in varnost. Njeno delovanje temelji na pregledu vsakega znaka v geslu ter naključni odločitvi o spremembi velikosti črke.

Najprej funkcija ustvari prazen niz `new_password`, kamor postopoma sestavlja spremenjeno različico gesla. Nato s pomočjo zanke `for` pregleda vsak znak v podanem geslu `password`. Če je znak črka, kar preveri z metodo `char.isalpha()`, funkcija z uporabo `random.choice([True, False])` naključno določi, ali bo črko spremenila iz male v veliko ali obratno. Če je črka mala, jo pretvori v veliko s `char.upper()`, če pa je velika, jo spremeni v malo s `char.lower()`.

V primeru, da znak ni črka, na primer številka ali poseben znak, ga funkcija pusti nespremenjenega in ga neposredno doda v `new_password`. Na koncu funkcija vrne novo različico gesla, kjer so nekatere črke naključno spremenjene v velike ali male, kar otežuje ugibanje gesla in zmanjšuje možnosti uspešnega napada s slovarjem (dictionary attack).

Na primer, če je izvorno geslo "maribor123", lahko po obdelavi postane "MaRiBor123", kjer je naključno spremenjena velikost črk. S tem se geslo ohrani v obliki, ki je uporabniku še vedno prepoznavna in relativno enostavna za pomnjenje, hkrati pa postane bolj odporna proti avtomatiziranim napadom.

```
# Zamenjava nekaterih črk s številkami (samo nekatere črke)
password_with_numbers = replace_with_numbers(password_with_random_case)
```

Slika 12: Zamenjava posameznih črk s številkami.

Ta vrstica kode kliče funkcijo `replace_with_numbers(password_with_random_case)`, katere naloga je zamenjava določenih črk v geslu s številkami, ki so jim vizualno podobne.

```
# Funkcija za zamenjavo črk s podobnimi številkami (spremenimo samo nekatere črke)
def replace_with_numbers(password):
    # Slovar za mapiranje črk na številkke
    char_to_number = {
        'e': '3', 't': '7', 'b': '8', 'q': '9', 'a': '4', 'o': '0', 's': '5', 'i': '1', 'l': '1'
    }

    new_password = ""
    for char in password:
        if char.lower() in char_to_number:
            # Naključno odločimo, ali bomo zamenjali črko z številko
            if random.choice([True, False]): # Naključno odločimo, ali bomo zamenjali
                # Zamenjamo črko s številko, ohranimo velikost črke
                new_password += char_to_number[char.lower()] if char.islower() else char_to_number[char.lower()].upper()
            else:
                new_password += char # Črka ostane nespremenjena
        else:
            new_password += char # Ohranimo črko, če ni v slovarju
    return new_password
```

Slika 13: Zamenjava posameznih črk s številkami.

Funkcija `replace_with_numbers(password)` je zasnovana za povečanje varnosti gesla z uporabo tehnike, znane kot "leet speak", pri kateri se nekatere črke naključno nadomestijo s

številkami, ki so jim vizualno podobne (npr. "e" → "3", "o" → "0", "s" → "5"). Ta pristop zmanjša predvidljivost gesla in ga naredi bolj odpornega proti napadom s slovarjem (dictionary attacks).

Postopek delovanja funkcije:

1. Ustvarjanje slovarja nadomestitev

Funkcija najprej ustvari slovar `char_to_number`, kjer določi zamenjave črk s številkami:

```
char_to_number = {'e': '3', 't': '7', 'o': '0', 's': '5', 'a': '4', 'i': '1'}.
```

V tem slovarju so črke uporabljene kot ključi, njihove številčne ustreznice pa kot vrednosti.

2. Inicializacija praznega niza

Funkcija ustvari prazen niz `new_password`, kamor bo shranjevala predelano geslo.

3. Iteracija skozi geslo in nadomeščanje črk

- Zanka `for` prehaja skozi vsak znak v podanem geslu `password`.
- Če je znak črka in se nahaja v slovarju `char_to_number`, funkcija z `random.choice([True, False])` naključno odloči, ali ga bo zamenjala s številko ali ga pustila nespremenjenega.
- Če je črka mala, jo nadomesti s številko iz slovarja.
- Če je črka velika, poskrbi, da tudi nadomestna številka ostane v veliki obliki s `.upper()`.

4. Dodajanje ostalih znakov brez sprememb

Če znak ni v slovarju (npr. številka, posebni znak), ga funkcija preprosto doda v `new_password` brez sprememb.

5. Vračanje nove različice gesla

Na koncu funkcija vrne predelano geslo, v katerem so nekatere črke naključno zamenjane s številkami, kar izboljša varnost in zapletenost gesla.

Primer delovanja:

Če je izvorno geslo "Maribor123", ga funkcija lahko spremeni v "M4r1b0r123". Podobno se "Password!" lahko spremeni v "P455w0rd!".

```
# Izboljšanje gesla (dodajanje posebnih znakov, številčk itd.)
enhanced_password = enhance_password(password_with_numbers)
```

Slika 14: Izboljšanje gesel.

Ta vrstica kode kliče funkcijo `enhance_password(password_with_numbers)`, ki izboljša varnost gesla z dodajanjem dodatnih varnostnih elementov, kot so velika črka, številka in poseben znak. To pomeni, da funkcija prejme že predelano geslo, v katerem so bile črke morda že naključno spremenjene v velike ali male ter zamenjane s številkami, nato pa mu doda dodatne varnostne sloje, ki otežijo napade z ugibanjem gesla. S tem pristopom geslo postane bolj kompleksno in težje predvidljivo.

```
# Funkcija za dodajanje velikih črk in posebnih znakov
def enhance_password(password):
    # Naključno dodaj veliko črko
    password += random.choice(string.ascii_uppercase)
    # Naključno dodaj številko
    password += random.choice(string.digits)
    # Naključno dodaj poseben znak
    password += generate_special_characters()
    return password
```

Slika 15: Dodajanje velikih črk in posebnih znakov.

Funkcija `enhance_password(password)` je namenjena izboljšanju varnosti gesla s tem, da mu doda dodatne znake, ki povečajo njegovo kompleksnost in odpornost proti napadom. Ko funkcija prejme vhodno geslo, mu najprej doda naključno veliko črko, ki jo pridobi z uporabo metode `random.choice(string.ascii_uppercase)`. To pomeni, da se izmed vseh črk angleške abecede od A do Z izbere ena in se pripne na konec gesla.

Nato funkcija na podoben način doda naključno številko, ki jo pridobi z `random.choice(string.digits)`, kar pomeni, da bo na koncu gesla dodana ena izmed številčk od 0 do 9. Zadnji korak funkcije vključuje še dodajanje naključnega posebnega znaka, ki ga pridobi s klicem `generate_special_characters()`. Ta funkcija iz nabora `string.punctuation` izbere enega izmed posebnih znakov, kot so `!`, `@`, `#`, `$`, `%`, s čimer še dodatno izboljša varnost gesla.

Ko so vsi elementi dodani, funkcija vrne novo, izboljšano različico gesla, ki zdaj vključuje vsaj eno veliko črko, eno številko in en poseben znak. S tem postane geslo varnejše in težje predvidljivo.

```
# Funkcija za generiranje naključnih posebnih znakov
def generate_special_characters():
    special_characters = string.punctuation # Vse posebne znake
    return random.choice(special_characters)
```

Slika 16: Generiranje naključnih posebnih znakov.

Funkcija `generate_special_characters()` je odgovorna za naključno izbiro enega posebnega znaka iz nabora vseh razpoložljivih ločil in posebnih simbolov, ki jih podpira programski jezik Python.

Najprej ustvari spremenljivko `special_characters`, ki prejme vrednost `string.punctuation`. Ta spremenljivka vsebuje niz vseh posebnih znakov, ki jih Python prepozna kot ločila, vključno z znaki, kot so `!@#$%^&*()_+=[{}];:",".<>?/` in drugi.

Nato funkcija uporabi `random.choice(special_characters)`, kar pomeni, da iz tega nabora naključno izbere en simbol. Ta pristop zagotavlja, da bo vsakič, ko se funkcija pokliče, vrnjen drugačen poseben znak, kar pripomore k večji varnosti gesla.

Dodajanje posebnih znakov oteži ugibanje gesla, saj poveča njegovo entropijo in naredi geslo manj predvidljivo, s čimer se poveča odpornost proti brute-force in slovarskim napadom.

Na koncu funkcija vrne izbrani poseben znak, ki se lahko nato uporabi pri sestavljanju gesla, s čimer geslo postane kompleksnejše, varnejše in težje razbitljivo.

Na primer, če bi bilo osnovno geslo "M4r1b0r", bi po dodajanju posebnega znaka postalo "M4r1b0r@", kar izboljša njegovo varnost.

```
# Prikaz končnega gesla
print("Tvoje izboljšano geslo je:", enhanced_password)
```

Slika 17: Prikaz končnega gesla.

Zadnja vrstica kode `print ("Tvoje izboljšano geslo je:", enhanced_password)` je odgovorna za prikaz končnega gesla uporabniku. Potem ko je geslo skozi različne funkcije izboljšano – očiščeno presledkov, naključno spremenjeno v velikosti črk, delno nadomeščeno s številkami ter dopolnjeno s posebnimi znaki – se na tej točki izpiše na zaslon.

Izraz "Tvoje izboljšano geslo je:" predstavlja besedilni del izpisa, ki uporabniku jasno sporoči, da gre za končni rezultat generacije gesla. Spremenljivka `enhanced_password`, ki sledi temu besedilu, vsebuje končno verzijo gesla, ki je skozi postopek postalo varnejše in edinstveno.

Če je bilo osnovno geslo "**Maribor123**", bi lahko po vseh izboljšavah postalo "**M4r1b0r!X9**", nato pa bi se prikazalo uporabniku kot:

Tvoje izboljšano geslo je: M4r1b0r!X9.

Ta vrstica kode je ključnega pomena, saj uporabniku omogoča, da **vidi svoje novo ustvarjeno geslo**, ki ga nato lahko uporabi za **prijavo v različne sisteme ali račune**. Z izpisom končnega rezultata se zagotovi tudi, da uporabnik ve, katero geslo je bilo ustvarjeno in si ga lahko po potrebi zabeleži ali shrani.

#### 4.1 Analiza

Pri tej analizi se bom osredotočil na primerjavo različnih metod generiranja gesel ter ocenil njihovo varnost, uporabnost in zapomljivost. Primerjal bom gesla, ki jih bo ustvaril moj lasten program, gesla, ki jih bo generiral ChatGPT, in gesla, ki jih bo naključno ustvaril Google. Namen analize je ugotoviti, katera od teh metod ponuja najboljše razmerje med varnostjo in praktičnostjo – torej ali je geslo dovolj kompleksno, da je odporno na različne vrste napadov, hkrati pa dovolj preprosto za pomnjenje in uporabo.

V prvem koraku bom preveril osnovne značilnosti vsakega generiranega gesla, pri čemer bom upošteval:

- dolžino gesla,
- prisotnost velikih in malih črk,
- vključitev števil in posebnih znakov.

Ti dejavniki so ključni za oceno kompleksnosti gesla in njegove odpornosti proti napadom z ugibanjem ali napadom s surovo silo (brute-force attacks). Daljša in bolj raznolika gesla so običajno varnejša, zato bom v analizi primerjal tudi povprečno dolžino gesel iz vsakega vira ter pogostost uporabe različnih tipov znakov.

Poleg same sestave gesla bom izvedel tudi analizo zapomljivosti, saj mora biti geslo ne le varno, temveč tudi praktično za vsakodnevno uporabo. Preveril bom, ali gesla sledijo vzorcem, ki jih človeški možgani lažje obdelajo in si jih hitreje zapomnijo – na primer, ali

vsebujejo smiselne besede, povezane črke ali preproste vzorce, ki jih je lažje priklicati iz spomina.

Za natančnejšo oceno varnosti gesel bom uporabil več metrik:

- entropijo gesla, ki meri, kako nepredvidljivo oziroma naključno je geslo – večja kot je entropija, težje ga je uganiti;
- čas potreben za razbijanje gesla z napadom surove sile, pri čemer bom upošteval različne hitrosti procesiranja gesel, ki jih omogoča sodobna strojna oprema;
- prisotnost besed iz slovarja, saj so takšna gesla bistveno bolj ranljiva na slovarske napade (dictionary attacks).

Za dodatno perspektivo bom preveril, ali katero od generiranih gesel spominja na gesla, ki so že bila kompromitirana v preteklih vdorih. Pri tem bom uporabil zbirke podatkov o uhajanju gesel in analiziral, ali katero izmed generiranih gesel vsebuje vzorce, podobne tistim, ki so že bili zlorabljeni.

Na koncu bom primerjal rezultate vseh teh analiz in oblikoval zaključek o tem, katera metoda generiranja gesel ponuja najboljše ravnovesje med varnostjo in uporabnostjo. Cilj analize ni zgolj ugotoviti, katera gesla so najvarnejša, temveč tudi raziskati, ali lahko varna gesla hkrati ostanejo dovolj preprosta za uporabo brez potrebe po geselskih upraviteljih.

Moje ugotovitve bodo lahko koristne za vsakogar, ki želi izboljšati svojo digitalno varnost, ne da bi pri tem žrtvoval praktičnost uporabe gesel.

## **1. M. Gesla:**

Najprej sem s pomočjo prej opisanega programa ustvaril pet gesel, ki so bila zasnovana tako, da bi bila prilagojena mojim potrebam, enostavna za pomnjenje in hkrati dovolj kompleksna, da bi jih bilo težko razbiti. Pri generiranju teh gesel sem upošteval načela varnosti in praktičnosti, pri čemer sem poskušal doseči ravnovesje med dolžino gesla, uporabo različnih vrst znakov ter oblikovanjem gesel na način, ki bi olajšal njihovo zapornitev.

Teh pet gesel bom uporabil kot predstavnike mojih generiranih gesel v nadaljnji analizi, kjer jih bom primerjal z gesli, ki jih generirata ChatGPT in Google. Cilj te analize bo ugotoviti, ali moj program resnično ustvarja gesla, ki so dovolj varna, a hkrati uporabna, ter ali se po varnosti in zapornivosti lahko kosajo z gesli, ki jih ponujajo druge metode generiranja.

Vprašanja in generirana gesla:

- Kateri je tvoj najljubši kraj za obisk? → **Ptuj**
- Kdo je tvoj najljubši umetnik? → **Bob Ross**
- Kako se imenuje tvoja najboljša prijateljica? → **Ana**
- Kateri je tvoj najljubši šport? → **košarka**
- Kaj je bila tvoja prva služba/poklic? → **zdravnik**

**Generirano geslo 1:** ptuJ80bROss4nakOšArkAZDravn1kL1#

- Kateri je tvoj najljubši športni klub? → **KKC**
- Kdo je tvoj največji vzornik? → **David Goggins**
- Kateri je tvoj najljubši kraj za obisk? → **Ptuj**
- Kako se imenuje tvoja najboljša prijateljica? → **Ana**
- Katero je tvoje najljubše letno obdobje? → **2004**

**Generirano geslo 2:** KKcDavidG0ggiNSP7uJANa2004L9^

- Kateri je tvoj najljubši predmet v šoli? → **računalništvo**
- Katero je tvoje najljubše letno obdobje? → **2004**
- Kateri je tvoj najljubši film? → **The Big Short**
- Kateri je tvoj najljubši šport? → **KKC**
- Kakšno ime ima tvoj prvi avto? → **Pepca**

**Generirano geslo 3:** r4čUn41n1š7v02004TH381gsH0RtkKcP3PC4G7~

- Kateri je tvoj najljubši barvni odtenek? → **zelena**
- Kateri je tvoj najljubši kraj za obisk? → **Ptuj**
- Kakšna je bila tvoja prva služba? → **zdravnik**
- Kakšno je ime tvoje osnovne šole? → **iosce**
- Kateri je tvoj najljubši predmet v šoli? → **računalništvo**

**Generirano geslo 4:** z3L3nAp7uJzdR4VNIK105cERAčUnA1N1ŠtV0Z4{

- Kaj je tvoja najljubša pijača? → **voda**
- Kateri je tvoj najljubši predmet v šoli? → **računalništvo**
- Kako se imenuje tvoja najboljša prijateljica? → **Ana**
- Kateri je tvoj najljubši film iz otroštva? → **The Big Short**
- Kje si se rodil? → **Celje**

**Generirano geslo5:** Vod4RAčUn41N1ŠTv0ANA7Heb1g5horTcelJ3X8\$

## 2. ChatGPT

Za pridobitev gesel, ki bi bila hkrati varna, prilagojena mojemu okusu in enostavna za zapomnitev, sem se obrnil na ChatGPT z vprašanjem: *"Hočem ustvariti 5 varnih gesel, ki bodo prilagojena mojemu okusu in si jih bom zapomnil zlahka. Mi lahko, prosim, pomagaš?"*

Moja ideja je bila, da umetna inteligenca na podlagi mojih osebnih preferenc generira gesla, ki bodo še vedno dovolj varna, a hkrati osebno relevantna, kar bi mi omogočilo, da si jih brez težav zapomnim brez pomoči geselskega upravitelja. Najprej je ChatGPT poskušal ugotoviti, kateri osebni elementi so mi blizu in bi jih bilo smiselno vključiti v geslo. Zato mi je zastavil vrsto vprašanj, ki so se nanašala na moje preference in osebne povezave z določenimi besedami ali številkami:

- Najljubša barva?
- Najljubša številka?
- Ime ali vzdevek, ki ti je pomemben?
- Najljubša hrana ali pijača?
- Katero leto ti veliko pomeni? (*npr. rojstvo, pomemben dogodek*)
- Katera beseda ali fraza te vedno nasmeji?

Odgovori na ta vprašanja so služili kot osnova za generiranje gesel, ki so bila prilagojena posebej meni, hkrati pa so vključevala različne varnostne elemente, kot so kombinacija velikih in malih črk, številke ter posebni znaki.

Na podlagi mojih odgovorov mi je ChatGPT predlagal naslednjih pet gesel:

- TiffanyKonj25!
- PumpUp2001\$
- Svalki&Blue25
- Konj\_Jam2001

- TiffSvalki#25.

Na prvi pogled so ta gesla precej drugačna od naključno generiranih gesel, saj vključujejo prepoznavne besede, ki so mi pomembne. To pomeni, da si jih lahko lažje zapomnim, saj temeljijo na osebnih referencah.

Kljub temu gesla vsebujejo varnostne elemente, kot so velike in male črke, številke ter posebni znaki, kar jih naredi bolj odporne proti preprostim napadom, kot so ugibanje gesel in slovarski napadi.

Gesla, ki jih je generiral ChatGPT, predstavljajo dobro ravnovesje med uporabnostjo in varnostjo. Vključujejo osebne reference, ki olajšajo pomnjenje, hkrati pa uporabljajo ključne varnostne elemente, zaradi katerih so bolj odporna proti napadom z ugibanjem in avtomatiziranim napadom.

### 3. Google

Za celovitejšo primerjavo sem v analizo vključil tudi pet gesel, ki jih naključno generira Google. Na ta način sem lahko neposredno primerjal gesla, ki jih ustvari moj program, gesla, ki jih prilagodi ChatGPT na podlagi osebnih preferenc, in gesla, ki jih Google generira popolnoma naključno, brez kakršnih koli prilagoditev posamezniku. Cilj te analize je pridobiti vpogled v razlike med različnimi metodami generiranja gesel in ugotoviti, katera od teh metod ponuja najboljše razmerje med varnostjo in praktičnostjo.

Googlova naključno ustvarjena gesla v teoriji predstavljajo najvišjo raven varnosti, saj temeljijo na popolni naključnosti in izpolnjujejo ključne varnostne kriterije, kot so zadostna dolžina, uporaba različnih vrst znakov ter nepredvidljivost vzorcev. Vendar pa se ob tem poraja vprašanje, ali so takšna gesla v praksi uporabna. Pomembno je ugotoviti, ali si jih je mogoče zapomniti brez pomoči geselskega upravitelja in ali jih je mogoče enostavno vnašati brez napak, kar je ključno za njihovo praktično uporabnost.

Googlova generirana gesla so bila naslednja:

- z#G2Zz,n!-bn5+K
- ZQ3d6kR+k68Z9y7
- 5CYAq6##EkW\_\*\*f
- eZF\_KFX6Wa3JYWZ
- g7NJU-@tNQ@FqWt.

S primerjavo treh vrst gesel – lastnoročno generiranih, prilagojenih s pomočjo ChatGPT-ja in naključno ustvarjenih s strani Googla – sem lahko ocenil, katera metoda ponuja najboljše ravnovesje med varnostjo in praktičnostjo. Cilj analize je bil ugotoviti, ali so naključno generirana gesla, ki zagotavljajo visoko varnost, hkrati tudi uporabna za vsakodnevno uporabo, ali pa so prilagojena gesla bolj praktična brez bistvenega zmanjšanja varnosti.

Poleg ocene varnosti gesel sem preučil tudi njihovo zapomljivost, saj mora biti geslo ne le varno, temveč tudi uporabniku prijazno. Pri tem sem preveril, ali se gesla skladajo z vzorci, ki jih človeški možgani lažje obdelajo in si jih hitreje zapomnijo. Prav tako sem analiziral, koliko časa bi trajalo, da bi se posamezno geslo razbilo s pomočjo napadov s surovo silo in ali vsebuje elemente, ki jih lahko izkoristijo napadi s slovarjem.

Zaključek analize bo pokazal, ali je smiselno uporabljati popolnoma naključna gesla, jih prilagoditi posamezniku ali razviti metodo, ki združuje prednosti obeh pristopov. Cilj raziskave je ugotoviti, ali je mogoče zasnovati algoritem, ki hkrati zagotavlja visoko stopnjo varnosti in uporabnost gesel v vsakodnevni praksi.

V tej tabeli so predstavljeni ključni varnostni in uporabnostni parametri gesel, ki jih je ustvaril moj program (M. gesla - RR), ChatGPT in Google. Analiza vključuje dolžino gesla, raznolikost znakov, prisotnost slovarskih besed, enostavnost pomnjenja in odpornost proti napadam.

Tabela 1: Primerjava generatorjev gesla.

Kriterij	M. gesla - RR	ChatGPT	Google
Vsaj 12–16 znakov	DA	DA	DA
Velike in male črke	DA	DA	DA
Številke	DA	NE	DA
Posebni znaki	DA	DA	DA
Ni slovarskih besed	NE	NE	DA
Ni ponavljajočih vzorcev	NE	NE	DA
Unikatnost	NE	DA	DA
Zapomljivost	NE	DA	NE
Brute-force odpornost (leta)*	1 tresvigintillion	806,8 trilijonov	5 kvadrilijonov
Entropija gesel (bitov)**	254,9	94,58	105,1

\*Primerjava Bruteforce napadov: M. Gesla:  $10^{72}$  let; ChatGPT:  $10^{2.91}$  let; Google:  $10^{15.7}$  let

\*\*Formula za Entropijo Gesel:  $E=L \times \log_2(N)$  (L = dolžina gesla; N = velikost znakovnega nabora)

## Varnost gesel

- Googlova gesla so najbolj naključna, ne vsebujejo slovarskih besed in ne sledijo ponavljajočim se vzorcem.
- Moj program ustvarja gesla z visoko entropijo in kompleksnostjo, vendar zaradi uporabe osebnih podatkov vsebujejo določene predvidljive vzorce.
- ChatGPT-jeva gesla so prilagojena uporabniku, vendar vsebujejo smiselne besede, zaradi česar so bolj ranljiva na slovarske napade.

## Zapomljivost gesel

- ChatGPT-jeva gesla so najenostavnejša za pomnjenje, saj vsebujejo znane besede in osebne reference.
- Googlova gesla so izjemno varna, vendar popolnoma naključna, kar pomeni, da jih je brez geselskega upravitelja skoraj nemogoče pomniti.
- Moj program zagotavlja boljše ravnovesje med varnostjo in uporabnostjo, vendar so gesla še vedno precej zapletena za ročno pomnjenje.

## Odpornost na brute-force napade

- Moj program (M. gesla - RR) ustvari najodpornejša gesla, ki bi s surovo silo potrebovala  $10^{72}$  let za razbitje.
- Googlova gesla so prav tako zelo odporna, saj bi jih bilo mogoče razbiti šele čez  $10^{15.7}$  let.
- ChatGPT-jeva gesla so najšibkejša glede na odpornost proti brute-force napadom, saj bi bila lahko razbita že po  $10^{2.91}$  letih.

Rezultati raziskave so jasno pokazali, da ustvarjanje univerzalnega programa za generiranje gesel, ki bi bil hkrati varen in prilagojen posamezniku, ni izvedljivo. Vsak uporabnik deluje v svojem specifičnem okolju, ima unikatne potrebe, navade in preference glede uporabe gesel, kar pomeni, da ni mogoče zasnovati ene same rešitve, ki bi ustrezala vsem.

Ugotovil sem, da si večina uporabnikov niti ne zapomni gesel, ki jih ustvarijo sami, zato bi bil razvoj personaliziranega sistema za generiranje gesel, ki bi bil hkrati varen in enostaven za uporabo, izredno zahteven. Glede na te ugotovitve se zdi najboljša praksa uporaba obstoječih mehanizmov za upravljanje gesel. Pri manj pomembnih spletnih storitvah, kjer uporabniki nimajo posebnih varnostnih zahtev, se zdi smiselno gesla vezati na e-poštne

naslove, kot je Gmail, saj to omogoča enostavno ponastavljanje gesel, če jih uporabnik pozabi. Pri računih, kjer je ključnega pomena, da si uporabnik geslo zapomni, pa bi bilo treba implementirati dodatne varnostne ukrepe, kot so biometrična avtentikacija (prstni odtis, prepoznavna obraza), pametne kartice ali dodatna osebna vprašanja, na katera pozna odgovor samo uporabnik.

V raziskavi sem prav tako ugotovil, da avtomatizirana gesla, ki jih ustvarjajo sistemi, kot je Google, zagotavljajo visoko stopnjo varnosti, vendar so pogosto nepraktična za pomnjenje. Po drugi strani gesla, ki jih uporabniki ustvarijo sami, pogosto ne dosegajo zadostne varnostne ravni, saj temeljijo na predvidljivih vzorcih ali osebnih podatkih. Idealna rešitev bi tako morala vključevati hibridni pristop, kjer uporabnik uporablja varnostne mehanizme za ključne račune in enostavnejše metode za manj pomembne prijave.

Moj lastni program za generiranje gesel se je v eksperimentu dobro odnesel, saj je ustvarjal gesla, ki so vsebovala ključne varnostne elemente. Vendar sem med programiranjem ugotovil, da sem pri zasnovi nekoliko zanemaril vidik zapomljivosti gesel. Sprva sem se trudil ustvariti gesla, ki bi bila hkrati varna in preprosta za pomnjenje, vendar se je izkazalo, da je ta kombinacija skorajda nemogoča. To potrjuje tudi analiza gesel, ki jih je ustvaril ChatGPT – ta gesla so sicer lažje zapomnljiva, a obenem manj varna, saj pogosto vsebujejo slovarske besede in predvidljive vzorce.

Moja raziskava je tako pokazala, da je razkorak med varnostjo in uporabnostjo gesel še vedno zelo velik in da popolnoma optimalne rešitve ni.

Zaključujem, da je najvarnejša praksa uporaba kombinacije različnih metod zaščite, pri čemer je treba za kritične račune uvesti dodatne plasti varnosti, kot so dvofaktorska avtentikacija, preverjanje z biometričnimi podatki in uporaba varnostnih ključev. Raziskava je pokazala, da univerzalne rešitve ni, vendar lahko uporabniki z razumevanjem tveganj in uporabo pravih orodij znatno izboljšajo svojo digitalno varnost.

## 5 Inženirski dnevnik

Vse spremembe svojih rešitev sem skrbno beležil v inženirskem dnevniku, kar mi je omogočilo natančno sledenje razvoju programa, ugotavljanje težav in zapisovanje rešitev. Na ta način sem lahko spremljal napredek, identificiral pomanjkljivosti in jih sproti odpravljaj.

Tabela 2: Inženirski dnevnik – sledenje spremembam.

Št.	Problem	Opis problema	Rešitev problema
#0	Začeti programirati	—	Začetek razvoja programa.
#1	Slaba naključnost	Modul random ni dovolj varen za generiranje gesel, saj ne zagotavlja kriptografsko varnih vrednosti.	Zamenjal sem random z secrets, ki je bolj primeren za varnostno občutljive operacije.
#2	Predolgo geslo	Uporaba 5 vprašanj je pogosto ustvarila geslo, ki je bilo težko zapomniti.	Zmanjšal sem število vprašanj iz 5 na 3, da je geslo še vedno varno, a lažje za pomnjenje.
#3	Predvidljivo zaporedje vprašanj	Vprašanja so bila statično določena in naključno izbrana, kar je lahko vodilo v ponavljajoče se vzorce.	Namesto ročno določenih vprašanj sem uporabil AI (OpenAI API), ki dinamično generira vprašanja.
#4	Ponavljajoči se znaki v geslu	Če so odgovori uporabnika vsebovali ponavljajoče se vzorce, so bila gesla manj varna.	Dodal sem naključno spreminjanje velikosti črk in zamenjavo nekaterih črk s številkami.
#5	Težava s presledki v geslih	Če je uporabnik v odgovorih uporabljal presledke, so ti ostali v geslu in ga naredili manj varnega.	Implementiral sem funkcijo <code>remove_spaces</code> , ki odstrani presledke iz odgovorov.
#6	Očitna zamenjava črk s številkami	Zamenjava črk s številkami je bila preveč dosledna, kar bi lahko napadalcem olajšalo razbijanje gesla.	Dodal sem naključni element, kjer se zamenjava črk s številkami ne zgodi vedno.
#7	Preveč zapleteno geslo za pomnjenje	Nekateri uporabniki so težko zapomnili gesla, saj so bila sestavljena iz preveč naključnih sprememb.	Uravnotežil sem kompleksnost z uporabo samo določenega števila naključnih sprememb, da geslo ostane močno, a zapomljivo.

## 6 Sklepna ugotovitev

Raziskava potrjuje, da ni univerzalne rešitve za varna in hkrati enostavno zapomnljiva gesla. Najzanesljivejši pristop ostaja uporaba upraviteljev gesel in večstopenjske avtentikacije, ki zagotavljata visoko varnost brez zmanjšanja uporabnosti. Analiza je pokazala, da uporabniki pogosto izbirajo predvidljive vzorce, medtem ko so algoritmično ustvarjena gesla varnejša, saj ne sledijo vzorcem in imajo višjo entropijo. Prav tako se je potrdilo, da so avtomatsko generirana gesla odpornejša proti napadam. Na podlagi teh ugotovitev bom v nadaljevanju potrdil ali ovrgel naslednje hipoteze.

H1: Daljša gesla so varnejša kot krajša gesla. – **Potrjena.**

Moja raziskava potrjuje, da so daljša gesla bistveno varnejša od krajših, saj imajo višjo entropijo in so odpornejša proti brute-force napadam. Izračuni so pokazali, da gesla z dolžino 16 znakov dosegajo približno 105 bitov entropije, medtem ko gesla z več kot 30 znaki presegajo 250 bitov, kar jih naredi praktično nezlomljiva. Daljša gesla eksponentno povečajo število možnih kombinacij, kar bistveno oteži njihovo razbijanje. Na podlagi teh ugotovitev lahko z gotovostjo potrdim, da so daljša gesla varnejša od krajših, kar potrjujejo tako teoretični izračuni kot praktični eksperimentalni rezultati.

H2: Uporaba različnih znakov (številke, velike in male črke, posebni znaki) poveča v varnost. – **Potrjena.**

Moja raziskava potrjuje, da uporaba različnih vrst znakov – številčk, velikih in malih črk ter posebnih simbolov – bistveno poveča varnost gesel. Analiza entropije je pokazala, da se z dodajanjem dodatnih vrst znakov eksponentno poveča število možnih kombinacij, kar otežuje napade z izčrpavanjem (brute-force) in slovarske napade. Na podlagi teh ugotovitev lahko z gotovostjo potrdim, da vključitev različnih vrst znakov v geslo bistveno poveča njegovo varnost, kar dokazujejo tako teoretični izračuni kot tudi eksperimentalni rezultati.

H3: Uporabniki intuitivno izbirajo predvidljive vzorce pri ustvarjanju gesel, medtem ko algoritmično generirana gesla zagotavljajo večjo varnost. – **Potrjena.**

Moja raziskava potrjuje, da uporabniki pri ustvarjanju gesel pogosto izbirajo predvidljive vzorce, medtem ko algoritmično generirana gesla zagotavljajo bistveno večjo varnost. Analiza gesel, ki jih uporabniki sami izberejo, je pokazala, da ta pogosto vključujejo osebne podatke, smiselne besede, zaporedja številčk ali ponavljajoče se vzorce, kar jih naredi ranljive za slovarske in usmerjene napade. Na podlagi teh ugotovitev lahko z gotovostjo potrdim, da

so gesla, ki jih izberejo uporabniki, pogosto šibka zaradi predvidljivosti, medtem ko algoritmično generirana gesla zagotavljajo bistveno višjo raven varnosti, saj odpravljajo človeške vzorce in povečujejo kompleksnost gesla.

H4: Aplikacijska gesla je težje uganiti in so odpornejša na »brute-force« napade. – **Potrjena.**

Moja raziskava potrjuje, da so aplikacijsko generirana gesla bistveno varnejša od ročno izbranih, saj so daljša, naključnejša in vključujejo raznolike znake, kar otežuje brute-force in slovarske napade. Analiza je pokazala, da ta gesla dosegajo višjo entropijo in ne sledijo predvidljivim vzorcem, ki bi jih napadalci lahko izkoristili. Medtem ko uporabniki pogosto izbirajo gesla na podlagi osebnih podatkov ali enostavnih kombinacij, aplikacije ustvarjajo naključna gesla, ki so zaradi svoje kompleksnosti bistveno težje razbitljiva. Zato lahko z gotovostjo trdim, da so algoritmično ustvarjena gesla varnejša, saj združujejo dolžino, naključnost in kompleksnost, kar zagotavlja visoko stopnjo zaščite pred napadi, s čimer pa seveda zgubi vso enostavnost, ki jo potrebujemo, da si gesla lahko zapomnimo.

H5: Varna gesla pogosto niso uporabniku prijazna, kar vodi v kompromise pri varnosti. – **Potrjena.**

Moja raziskava potrjuje, da visoka varnost gesel pogosto pomeni slabšo uporabniško izkušnjo, kar vodi v kompromise pri varnosti. Dolga in naključno generirana gesla so sicer odporna proti napadom, vendar si jih uporabniki težko zapomnijo, kar pogosto privede do shranjevanja gesel na nezavarovanih mestih ali uporabe enostavnejših, ponavljajočih se gesel. Analiza je pokazala, da aplikacije pogosto predlagajo izjemno varna, a nepraktična gesla, ki jih uporabniki težko uporabljajo brez upraviteljev gesel. Posledično se mnogi odločajo za krajša in manj varna gesla, kar povečuje tveganje za vdore. Zato lahko zaključim, da se mora učinkovita varnost gesel uravnotežiti z njihovo uporabnostjo, saj sicer uporabniki zaradi praktičnosti sprejemajo tveganja, ki zmanjšujejo varnost njihovih računov.

H6: Možna je optimizacija gesel, ki hkrati zagotavlja visoko varnost in uporabnost. – **Ovržena.**

Moja raziskava kaže, da optimizacija gesel, ki bi hkrati zagotavljala visoko varnost in enostavno uporabo, v praksi ni popolnoma izvedljiva. Dolga in kompleksna gesla so varna, vendar si jih uporabniki težko zapomnijo, medtem ko so preprostejša gesla uporabnejša, a bistveno manj odporna proti napadom. Čeprav nekateri algoritmi poskušajo generirati gesla, ki naj bi bila hkrati varna in enostavna za pomnjenje, analiza kaže, da vsak poskus

poenostavitve gesel vodi v zmanjšanje entropije in predvidljive vzorce, ki jih napadalci lahko izkoristijo. Zato je nujno, da uporabniki pri zagotavljanju varnosti svojih računov uporabljajo upravitelje gesel ali druge metode avtentikacije, saj popolnega ravnotežja med varnostjo in enostavnostjo pomnjenja gesel ni mogoče doseči.

## 7 Zaključek

Zaključimo lahko, da ustvarjanje univerzalnega programa za generiranje gesel ni preprosta naloga, saj vsak posameznik uporablja tehnologijo na svoj način, ima različne varnostne zahteve in edinstvene navade pri upravljanju gesel. To pomeni, da je težko razviti rešitev, ki bi hkrati zagotavljala najvišjo raven varnosti in bila povsem prilagojena uporabniški izkušnji.

Raziskava je pokazala, da si večina ljudi težko zapomni gesla, tudi če jih sami izberejo. Zato je pri manj pomembnih računih smiselno uporabiti sistemske rešitve, kot je povezava z e-pošto (npr. Gmail), kjer lahko uporabnik preprosto ponastavi geslo v primeru, da ga pozabi. Pri računih z višjo stopnjo varnostnega tveganja pa zgolj zapomnljivo geslo ni dovolj, zato je priporočljivo vključiti dodatne zaščitne ukrepe, kot so:

- biometrična avtentikacija (prstni odtis, prepoznavna obraza),
- pametne kartice ali varnostni ključi,
- preverjanje z osebnimi vprašanji, na katera pozna odgovor le uporabnik.

Poleg tega se je pokazalo, da popolnoma naključno ustvarjena gesla, kot jih generirajo nekateri sistemi, zagotavljajo visoko varnost, vendar so pogosto nepraktična za pomnjenje in uporabo. Mnogi uporabniki zaradi zapletenosti raje izberejo manj varna gesla, ki si jih lažje zapomnijo, ali pa si jih zapišejo na nezaščitena mesta, kar predstavlja dodatno varnostno tveganje. To potrjuje, da varnost in uporabnost gesel pogosto stojita v nasprotju, zato je pri oblikovanju varnostnih rešitev nujno iskati ravnotežje med obema.

Zato je najboljša praksa kombinacija različnih metod zaščite, ki uporabnikom omogočajo varen in enostaven dostop do podatkov, hkrati pa zagotavljajo najvišjo stopnjo varnosti. Za optimalno zaščito je priporočljivo uporabljati upravitelje gesel, ki omogočajo shranjevanje in generiranje močnih gesel, ter večstopenjsko avtentikacijo (2FA), ki preprečuje nepooblaščen dostop tudi v primeru razkritja gesla.

Varnost gesel je danes ključnega pomena, saj digitalni napadi postajajo vse bolj sofisticirani. Zato morajo uporabniki izbrati rešitve, ki ne le zagotavljajo visoko varnost, temveč so tudi dovolj praktične, da jih lahko brez težav uporabljajo v vsakdanjem življenju.

## 8 Viri in literatura

Google. *Use Chrome's password manager*. Pridobljeno 2. aprila 2025 s

<https://support.google.com/chrome/answer/95606>

Have I Been Pwned. Check if your email or phone is in a data breach. Dostopno na:

<https://haveibeenpwned.com/>

Kovačič, M. (2016). *Kriptografija*. Dostopno na:

[https://telefoncek.si/static/2016/06/Kriptografija\\_Kovacic.pdf](https://telefoncek.si/static/2016/06/Kriptografija_Kovacic.pdf)

PHC (Password Hashing Competition). (2015). Argon2: Specification and Benchmarking.

Dostopno na: <https://www.password-hashing.net/argon2-specs.pdf>

Python Software Foundation. Welcome to Python.org. Dostopno na:

<https://www.python.org/>

Security.org. How secure is my password?. Dostopno na: [https://www.security.org/how-](https://www.security.org/how-secure-is-my-password/)

[secure-is-my-password/](https://www.security.org/how-secure-is-my-password/)

Simplilearn. SHA-256 algorithm explained. Dostopno na:

<https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm>

Stytc. (2023). Argon2 vs Bcrypt vs Scrypt: Which password hashing algorithm should

you use?. Dostopno na: <https://stytc.com/blog/argon2-vs-bcrypt-vs-scrypt/>

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko. (2023). Uvod v

kriptografijo – drugi del. Dostopno na: [https://ucilnica.fri.uni-](https://ucilnica.fri.uni-lj.si/pluginfile.php/32408/mod_resource/content/0/drugi_del.pdf)

[lj.si/pluginfile.php/32408/mod\\_resource/content/0/drugi\\_del.pdf](https://ucilnica.fri.uni-lj.si/pluginfile.php/32408/mod_resource/content/0/drugi_del.pdf)

Varni na internetu. *Močno geslo? Ne? Pa naj postane!*. Pridobljeno 2. aprila 2025, s

<https://www.varninainternetu.si/mocno-geslo/>

Van Rossum, G. (1999). *The History of Python*. Dostopno na: [https://python-](https://python-history.blogspot.com/)

[history.blogspot.com/](https://python-history.blogspot.com/)

Wikipedia. (2024). *Cezarjeva šifra*. Pridobljeno 2. aprila 2025 s spletnega mesta:

[https://sl.wikipedia.org/wiki/Cezarjeva\\_šifra](https://sl.wikipedia.org/wiki/Cezarjeva_šifra)

Wikipedia. (2023). SHA-2. Dostopno na: <https://en.wikipedia.org/wiki/SHA-2>