Art. 267 KK

§*Kto bez uprawnienia uzyskuje dostęp do całości lub części systemu informatycznego podlega karze pozbawienia wolności do lat 2.*

Art. 269a KK

§*Kto, nie będąc do tego uprawnionym w istotnym stopniu zakłóca pracę systemu systemu teleinformatycznego lub sieci teleinformatycznej, podlega karze pozbawienia wolności od 3 miesięcy do lat 5.*

Art.269c KK

§*Nie podlega karze za przestępstwo określone w art. 267 §2 lub art. 269a, kto działa wyłącznie w celu zabezpieczenia systemu teleinformatycznego albo opracowania metody takiego zabezpieczenia i niezwłocznie powiadomił dysponenta tego systemu lub sieci o ujawnionych zagrożeniach, a jego działanie nie naruszyło interesu publicznego lub prywatnego i nie wyrządziło szkody.*

Informacje

GitHub

Na potrzeby zajęć swoje własne repozytorium na Githubie. Rezpozytorium będziesz wykorzystywać, aby wrzucać do niego rozwiązania zadań z przedmiotu. Możesz nazwać repozytorium dowolnie, ale najlepiej wykorzystaj nazwę:
`imie-nazwisko-bsk-umcs`.

Wirtualne środowisko pracy

Wirtualne środowisko pracy w Python, znane również jako `virtualenv` lub `venv`, jest sposobem za zapewnienie działania naszego programu niezależnie od maszyny na której jest uruchamiany ORAZ sposobem na uruchomienie innych programów na NASZEJ maszynie, tak aby instalowane z nią zależności nie zakłóciły pracy innych programów. Wirtualne środowisko pracy, jest swojego rodzaju odizolowanym katalogiem, zawierającym instalację języka programowania Python oraz zainstalowane biblioteki na potrzeby programu, który będziemy w nim uruchamiać.

Instalacja

```
sudo apt update
sudo apt install python3-pip
python3 -m pip install --upgrade pip

sudo apt install python3-venv
python3 -m pip install --user virtualenv
```

Tworzenie środowiska

```
python3 -m venv venv
source ./venv/bin/activate
```

Instalowanie pakietów

```
pip list
pip install black
black test.py
```

Dezaktywacja (koniec pracy) wirtualnego środowiska

```
deactivate
```

Dobre praktyki

W zadaniach najczęściej będziemy wykorzystywać język programowania Python. Aby zachować dobre praktyki programistyczne, będziemy używać **black'a** oraz **pylamy**. Jeśli to możliwe, możesz wykorzystać inny język programowania (jest to zależne od zadania). Pamiętaj jednak również o zachowaniu dobrych praktyk.

Narzędzia

hash-identifier

`hash-identifier` to narzędzie służące do identyfikacji różnych typów hashy kryptograficznych. Hash funkcjonuje jako skrót wiadomości (ang. *message digest*), który jest stosowany w kryptografii w celu sprawdzenia integralności danych lub w zabezpieczeniach haseł. Aby użyć `hash-identifier`, wystarczy uruchomić narzędzie z poziomu terminala, a następnie wprowadzić hash, który chcemy zidentyfikować. Program automatycznie przeanalizuje format i poda możliwe algorytmy, z jakimi może być związany podany hash. Narzędzie jest szczególnie przydatne podczas analizy danych pozyskanych z audytów bezpieczeństwa, forensyki cyfrowej, czy przy łamaniu haseł. `hash-identifier` to proste, ale użyteczne narzędzie, które przyspiesza proces identyfikacji algorytmów hashujących. Dzięki jego intuicyjnej obsłudze, jest to wartościowe narzędzie w arsenale specjalistów zajmujących się bezpieczeństwem informacji.

hashid

`hashid` HashID to niewielkie, ale bardzo przydatne narzędzie wiersza poleceń służące do automatycznej identyfikacji typu (algorytmu) dowolnego ciągu znaków zakodowanego w formie hash (skrótu). HashID analizuje wejściowy ciąg (np. 5f4dcc3b5aa765d61d8327deb882cf99) i zgaduje, jakim algorytmem został on wytworzony (MD5, SHA-1, SHA-256, NTLM, MySQL, bcrypt, itp.).

OpenSSL

To wieloplatformowa, otwarta implementacja protokołów SSL (wersji 2 i 3) i TLS (wersji 1) oraz algorytmów kryptograficznych ogólnego przeznaczenia. Dostępna jest dla systemów uniksopodobnych (m.in. Linux, BSD, Solaris), OpenVMS i Microsoft Windows. OpenSSL zawiera biblioteki implementujące wspomniane standardy oraz mechanizmy kryptograficzne, a także zestaw narzędzi konsolowych (przede wszystkim do tworzenia kluczy oraz certyfikatów, zarządzania urzędem certyfikacji, szyfrowania, dekryptażu i obliczania podpisów cyfrowych). OpenSSL pozwala na używanie wszystkich zastosowań kryptografii. Poniżej kilka kluczowych cech OpenSSL:

- **Wszechstronność**: OpenSSL oferuje bogaty zestaw narzędzi i bibliotek do obsługi różnych protokołów kryptograficznych, w tym SSL/TLS, kryptografię klucza publicznego, szyfrowanie danych i wiele innych.
- **Bezpieczeństwo sieciowe**: OpenSSL dostarcza narzędzia do zarządzania certyfikatami, generowania kluczy, podpisywania cyfrowego i szyfrowania danych, co umożliwia tworzenie bezpiecznych aplikacji internetowych i usług.
- **Wsparcie dla różnych platform**: OpenSSL jest dostępny na wielu platformach, w tym na systemach Unix, Linux, macOS, Windows oraz innych, co czyni go popularnym narzędziem wśród programistów i administratorów systemów.
- **Otwarty kod źródłowy**: OpenSSL jest projektem open-source, co oznacza, że jego kod jest dostępny publicznie i może być modyfikowany oraz rozwijany przez społeczność, co sprzyja ciągłemu ulepszaniu i dostosowywaniu narzędzia do różnych potrzeb i zastosowań.
- **Wsparcie dla wielu protokołów**: OpenSSL obsługuje wiele standardowych protokołów kryptograficznych, takich jak SSL/TLS, SSH, S/MIME, PKCS, co czyni go wszechstronnym narzędziem do implementacji bezpiecznych komunikacji w różnych aplikacjach i systemach.

Linki

- https://www.openssl.org/
- https://www.kali.org/tools/hash-identifier/
- https://pypi.org/project/hashID/
- https://pypi.org/project/black/
- https://pypi.org/project/pylama/
- https://www.toptal.com/developers/gitignore
- https://pypi.org/project/bcrypt/
- https://httpd.apache.org/docs/current/programs/htpasswd.html

- https://hashcat.net/wiki/doku.php?id=example_hashes
- https://en.wikipedia.org/wiki/Hash_function
- https://www.baeldung.com/cs/hashing
- https://cryptobook.nakov.com/cryptographic-hash-functions
- https://argon2.online/
- https://manpages.ubuntu.com/manpages/questing/man1/scrypt.1.html
- https://informatykzakladowy.pl/jak-serwer-sprawdza-haslo
- https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html
- https://www.cyber.mil.pl/funkcje-skrotu/

Teoria

Funkcje haszujące to deterministyczne algorytmy, które przekształcają dane wejściowe dowolnej długości na ciąg znaków o stałej długości – zwany skrótem (ang. *hash*). Ich podstawowe cechy to:

- **Jednostronność**: trudność odtworzenia danych wejściowych ze skrótu.
- **Stała długość skrótu**: niezależnie od długości danych wejściowych.
- **Unikalność**: minimalizacja kolizji – różne dane powinny dawać różne skróty.
- **Deterministyczność**: te same dane wejściowe dają ten sam wynik.
- **Szybkość**: szybkie przetwarzanie dużych danych.
- **Odporność na kolizje**: trudność znalezienia dwóch różnych danych z tym samym skrótem.

Zastosowanie funkcji haszujących

- **Zabezpieczanie haseł**: przechowywanie skrótów haseł zamiast samych haseł.
- **Weryfikacja integralności danych**: np. sumy kontrolne plików.
- **Kryptografia**: podpisy cyfrowe, generowanie kluczy.
- **Systemy autoryzacji i uwierzytelniania**.
- **Ochrona prywatności i anonimizacja danych**.

Klasyczne funkcje haszujące

- **MD5 (Message Digest 5)**:
  - 128-bitowy skrót.
  - Szybka, ale nieodporna na kolizje.
  - Złamana – niezalecana do zastosowań bezpieczeństwa.
- **SHA-1 (Secure Hash Algorithm 1)**:
  - 160-bitowy skrót.
  - Używana w przeszłości m.in. w certyfikatach SSL.
  - Również uważana za złamaną (od 2017 potwierdzone kolizje).
- **SHA-256**:
  - 256-bitowy skrót.
  - Obecnie uznawana za bezpieczną, ale szybka – nieoptymalna do przechowywania haseł.

Nowoczesne funkcje haszujące

- **bcrypt**:
  - Opiera się na algorytmie Blowfish.
  - Używa mechanizmu kosztu (`cost`), który określa czas haszowania.
  - Automatycznie dodaje sól (salt), dzięki czemu ten sam tekst hasłowy daje różne hasze.
  - Dobrze chroni przed atakami słownikowymi i tablicami tęczowymi.
- **scrypt**:
  - Wprowadza wymagania pamięciowe – utrudnia użycie specjalistycznego sprzętu (GPU/ASIC).
  - Umożliwia konfigurację pamięci, CPU oraz długości wyjścia.

- **Argon2** (zwycięzca PHC 2015):
  - Trzy warianty: `Argon2d`, `Argon2i`, `Argon2id`.
  - Pozwala ustawić parametry: pamięć, czas, liczbę wątków, sól.
  - Bardzo odporna na ataki brute-force i tęczowe tablice.
  - Obecnie rekomendowana funkcja do przechowywania haseł.

Porównanie cech: stare vs nowoczesne funkcje

Cechy funkcji haszujących	**Stare funkcje** (MD5, SHA-1, SHA-256)	**Nowoczesne funkcje** (bcrypt, scrypt, Argon2)
Jednostronność	Tak	Tak
Stała długość skrótu	Tak	Tak
Unikalność (mało kolizji)	Nie (MD5/SHA-1) / Częściowo (SHA-256)	Tak (odporne na kolizje)
Deterministyczność	Tak	Tak (ale z losową solą hasz daje inny wynik)
Szybkość	Bardzo szybkie (łatwiejsze do złamania)	Wolne (celowo, zwiększają koszt ataku)
Odporność na kolizje	Niska / średnia	Wysoka
Odporność na ataki GPU/rainbow tables	Nie	Tak
Konfigurowalność (koszt, pamięć, wątki)	Brak	Tak (duża elastyczność w doborze parametrów)

Podsumowanie

Klasyczne funkcje haszujące takie jak MD5 czy SHA-1 nie zapewniają już dziś wystarczającego poziomu bezpieczeństwa – głównie z powodu podatności na kolizje oraz braku mechanizmów obrony przed atakami brute-force. Dlatego też w kontekście przechowywania haseł zaleca się stosowanie nowoczesnych funkcji haszujących takich jak bcrypt, scrypt czy Argon2, które dzięki zastosowaniu soli, kosztu czasowego i wymagań pamięciowych skutecznie utrudniają łamanie haseł nawet przy użyciu nowoczesnego sprzętu.

Zadania

1.1 *Pod adresem* ***http://127.0.0.1:10001*** *działa prosty serwer HTTP udostępniający dwa endpointy:* ***http://127.0.0.1:10001/hash*** *oraz* ***http://127.0.0.1:10001/submit*** *.*

Twoim zadaniem jest obliczenie skrótu MD5 podanego przez serwer losowego słowa. Wynikowy hash powinien być w formacie szesnastkowym (hex). Aby otrzymać od serwera słowo do zahashowania oraz identyfikator sesji, wyślij zapytanie HTTP ***GET*** *do endpointa* ***http://127.0.0.1:10001/hash*** *. Otrzymasz w odpowiedzi unikalny identyfikator sesji oraz losowe słowo do zahashowania.*

Następnie powinieneś obliczyć skrót MD5 otrzymanego słowa, kodując wynik w formacie szesnastkowym (hex).

Po obliczeniu hash'a wyślij go do serwera metodą HTTP ***POST*** *na endpoint* ***http://127.0.0.1:10001/submit*** *, przekazując w formacie JSON zarówno identyfikator sesji, jak i obliczony skrót MD5 (hex). Serwer zweryfikuje poprawność przesłanego hasha i zwróci informację o sukcesie lub błędzie.*

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 10001:10001 mazurkatarzyna/hashing-md5-ex1:latest
```

(b) Do obliczenia skrótu MD5 użyj narzędzia `OpenSSL`.

(c) Aby wysłać odpowiedź do serwera, użyj narzędzia `cURL`. Pamiętaj o dołączeniu nagłówka HTTP `"Content-Type: application/json"`.

1.2 *Pod adresem* ***http://127.0.0.1:10002*** *działa prosty serwer HTTP udostępniający dwa endpointy:* ***http://127.0.0.1:10002/hash*** *oraz* ***http://127.0.0.1:10002/submit*** *.*

Twoim zadaniem jest obliczenie skrótu SHA-256 podanego przez serwer losowego słowa. Wynikowy hash powinien być w formacie szesnastkowym (hex). Aby otrzymać od serwera słowo do zahashowania oraz identyfikator sesji, wyślij zapytanie HTTP ***GET*** *do endpointa* ***http://127.0.0.1:10002/hash*** *. Otrzymasz w odpowiedzi unikalny identyfikator sesji oraz losowe słowo do zahashowania.*

Następnie powinieneś obliczyć skrót SHA-256 otrzymanego słowa, kodując wynik w formacie szesnastkowym (hex).

Po obliczeniu hash'a wyślij go do serwera metodą HTTP ***POST*** *na endpoint* ***http://127.0.0.1:10002/submit*** *, przekazując w formacie JSON zarówno identyfikator sesji, jak i obliczony skrót MD5 (hex). Serwer zweryfikuje poprawność przesłanego hasha i zwróci informację o sukcesie lub błędzie.*

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 10002:10002 mazurkatarzyna/hashing-sha256-ex1:latest
```

(b) Do obliczenia skrótu SHA-256 użyj narzędzia `OpenSSL`.

(c) Aby wysłać odpowiedź do serwera, użyj narzędzia `cURL`. Pamiętaj o dołączeniu nagłówka HTTP `"Content-Type: application/json"`.

1.3 *Pod adresem* `http://127.0.0.1:10003` *działa prosty serwer HTTP udostępniający dwa endpointy:* `http://127.0.0.1:10003/hash` *oraz* `http://127.0.0.1:10003/submit`.

Twoim zadaniem jest obliczenie skrótu SHA-512 podanego przez serwer losowego słowa. Wynikowy hash powinien być w formacie szesnastkowym (hex). Aby otrzymać od serwera słowo do zahashowania oraz identyfikator sesji, wyślij zapytanie HTTP `GET` *do endpointa* `http://127.0.0.1:10003/hash`. *Otrzymasz w odpowiedzi unikalny identyfikator sesji oraz losowe słowo do zahashowania.*

Następnie powinieneś obliczyć skrót SHA-512 otrzymanego słowa, kodując wynik w formacie szesnastkowym (hex).

Po obliczeniu hash'a wyślij go do serwera metodą HTTP `POST` *na endpoint* `http://127.0.0.1:10003/submit`, *przekazując w formacie JSON zarówno identyfikator sesji, jak i obliczony skrót SHA-512 (hex). Serwer zweryfikuje poprawność przesłanego hasha i zwróci informację o sukcesie lub błędzie.*

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 10003:10003 mazurkatarzyna/hashing-sha-512-ex1:latest
```

(b) Do obliczenia skrótu SHA-512 użyj narzędzia `OpenSSL`.

(c) Aby wysłać odpowiedź do serwera, użyj narzędzia `cURL`. Pamiętaj o dołączeniu nagłówka HTTP `"Content-Type: application/json"`.

1.4 *Pod adresem* `http://127.0.0.1:10004` *działa prosty serwer HTTP udostępniający dwa endpointy:* `http://127.0.0.1:10004/hash` *oraz* `http://127.0.0.1:10004/submit`.

Twoim zadaniem jest obliczenie skrótu ***Argon*** *podanego przez serwer losowego słowa. Wynikowy hash powinien być w formacie szesnastkowym (hex). Aby otrzymać od serwera słowo do zahashowania, parametry użyte do haszowania oraz identyfikator sesji, wyślij zapytanie HTTP* `GET` *do endpointa* `http://127.0.0.1:10004/hash`. *Otrzymasz w odpowiedzi unikalny identyfikator sesji, losowe słowo do zahashowania i niezbędne parametry.*

Następnie powinieneś obliczyć skrót ***Argon*** *otrzymanego słowa, kodując wynik w formacie szesnastkowym (hex).*

Po obliczeniu hash'a wyślij go do serwera metodą HTTP `POST` *na endpoint* `http://127.0.0.1:10004/submit`, *przekazując w formacie JSON zarówno identyfikator sesji, jak i obliczony skrót* ***Argon*** *(hex). Serwer zweryfikuje poprawność przesłanego hasha i zwróci informację o sukcesie lub błędzie.*

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 10004:10004 mazurkatarzyna/hashing-argon-ex1:latest
```

(b) Do obliczenia skrótu Argon narzędzia `OpenSSL` dostępnego jako kontener Dockerowy. W tym celu uruchom kontener, i wejdź do jego środka. Haszowanie za pomocą algorytmu Argon dostępne jest pod nazwą `kdf`.

```
docker run -it mazurkatarzyna/openssl-332-ubuntu:latest
openssl kdf -help
```

(c) Aby wysłać odpowiedź do serwera, użyj narzędzia `cURL`. Pamiętaj o dołączeniu nagłówka HTTP `"Content-Type: application/json"`.

1.5 *Pod adresem* `http://127.0.0.1:10005` *działa prosty serwer HTTP udostępniający dwa endpointy:* `http://127.0.0.1:10005/hash` *oraz* `http://127.0.0.1:10005/submit`.

Twoim zadaniem jest obliczenie skrótu ***bcrypt*** *podanego przez serwer losowego słowa. Wynikowy hash powinien być w formacie szesnastkowym (hex). Aby otrzymać od serwera słowo do zahashowania oraz identyfikator sesji, wyślij zapytanie HTTP* `GET` *do endpointa* `http://127.0.0.1:10005/hash`. *Otrzymasz w odpowiedzi unikalny identyfikator sesji i losowe słowo do zahashowania.*

Następnie powinieneś obliczyć skrót ***bcrypt*** *otrzymanego słowa, kodując wynik w formacie szesnastkowym (hex).*

Po obliczeniu hash'a wyślij go do serwera metodą HTTP `POST` *na endpoint* `http://127.0.0.1:10005/submit`, *przekazując w formacie JSON zarówno identyfikator sesji, jak i obliczony skrót* ***bcrypt*** *(hex). Serwer zweryfikuje poprawność przesłanego hasha i zwróci informację o sukcesie lub błędzie.*

(a) Uruchom serwer za pomocą poniższego polecenia:

```
docker run -p 10005:10005 mazurkatarzyna/hashing-bcrypt-ex1:latest
```

(b) Do obliczenia skrótu **bcrypt** użyj narzędzia `htpasswd` dostępnego jako kontener Dockerowy.

```
docker run mazurkatarzyna/htpasswd:latest --help
```

(c) Aby wysłać odpowiedź do serwera, użyj narzędzia `cURL`. Pamiętaj o dołączeniu nagłówka HTTP `"Content-Type: application/json"`.

1.6 Napisz skrypt w języku Python, w którym wygenerujesz hash **MD5** dowolnego ciągu znaków podawanego jako argument wywołania skryptu. Sprawdź poprawność wygenerowanego hasha porównując go z wynikiem otrzymanym przy pomocy `md5sum` lub `openssl`. Wykorzystaj bibliotekę `hashlib`. Możesz wykorzystać poniższy szkielet kodu:

```
import hashlib

def md5_hash_string(input_string: str) -> str:
    """
    Returns the MD5 hash of a given string.
    """
    pass

if __name__ == "__main__":
    text = "hello world"
    print(f"MD5 hash of '{text}': {md5_hash_string(text)}")
```

1.7 Napisz skrypt w języku Python, w którym wygenerujesz hash **SHA-1** dowolnego ciągu znaków podawanego jako argument wywołania skryptu. Sprawdź poprawność wygenerowanego hasha porównując go z wynikiem otrzymanym przy pomocy `sha1sum` lub `openssl`. Wykorzystaj bibliotekę hashlib. Możesz wykorzystać poniższy szkielet kodu:

```
import hashlib

def sha1_hash_string(input_string: str) -> str:
    """
    Returns the SHA-1 hash of a given string.
    """
    pass

if __name__ == "__main__":
    text = "hello world"
    print(f"SHA-1 hash of '{text}': {sha1_hash_string(text)}")
```

1.8 Napisz skrypt w języku Python, w którym wygenerujesz hash **SHA-256** dowolnego ciągu znaków podawanego jako argument wywołania skryptu. Sprawdź poprawność wygenerowanego hasha porównując go z wynikiem otrzymanym przy pomocy `openssl`. Wykorzystaj bibliotekę hashlib. Możesz wykorzystać poniższy szkielet kodu:

```
import hashlib

def sha256_hash_string(input_string: str) -> str:
    """
    Returns the SHA-256 hash of a given string.
    """
    pass

if __name__ == "__main__":
    text = "hello world"
    print(f"SHA-256 hash of '{text}': {sha256_hash_string(text)}")
```

1.9 Napisz skrypt w języku Python, w którym wygenerujesz hash **bcrypt** dowolnego ciągu znaków podawanego jako argument wywołania skryptu. Sprawdź poprawność wygenerowanego hasha porównując go z wynikiem otrzymanym przy pomocy `htpasswd`. Do napisania programu potrzebna jest biblioteka bcrypt zainstalowana w kontenerze Dockerowym. Uruchom kontener, i wejdź do jego środka - biblioteka jest już zainstalowana w kontenerze. Aby napisać program, wykorzystaj `nano`.

(a) Uruchom kontener za pomocą poniższego polecenia:

```
docker run -it mazurkatarzyna/hashing-bcrypt-ex2:latest
nano template.py
```

(b) Do weryfikacji skrótu **bcrypt** użyj narzędzia `htpasswd` dostępnego jako kontener Dockerowy.

```
docker run mazurkatarzyna/htpasswd:latest --help
```

Możesz wykorzystać poniższy szkielet kodu, który dostępny jest również wewnątrz kontenera jako plik `template.py`:

```
import bcrypt

def bcrypt_hash_password(password: str) -> str:
    """
    Hash a password using bcrypt.
    """
    pass

def bcrypt_verify_password(password: str, hashed: str) -> bool:
    """
    Verify a password against a bcrypt hash.
    """
    pass

# Example usage
if __name__ == "__main__":
    pwd = "my_secure_password"
```

1.10 Mając dany początkowy ciąg znaków `helloworld`, który następnie został zahashowany, określ, jaka funkcja skrótu została wykorzystana do utworzenia hasha: `6adfb183a4a2c94a2f92dab5ade762a47889a5a1`. Możesz wykorzystać narzędzie `hash-identifier`:

```
docker run -it mazurkatarzyna/hash-identifier:latest
```

Czy `hash-identifier` odnalazł nazwę hasha? Jeśli nie, zaproponuj inny sposób na rozwiązanie zadania.

1.11 Mając dany początkowy ciąg znaków `helloworld`, który następnie został zahashowany, określ, jaka funkcja skrótu została wykorzystana do utworzenia hasha: `$2y$10$xbyAv5a46CQYPay5UISCNeFWpVdx2qvhCBEOZ/YtfxoVXhOGrVKQa`. Możesz wykorzystać narzędzie `hash-identifier`:

```
docker run -it mazurkatarzyna/hash-identifier:latest
```

Czy `hash-identifier` odnalazł nazwę hasha? Jeśli nie, zaproponuj inny sposób na rozwiązanie zadania.

1.12 Mając dany początkowy ciąg znaków `helloworld`, który następnie został zahashowany, określ, jaka funkcja skrótu została wykorzystana do utworzenia hasha: `6adfb183a4a2c94a2f92dab5ade762a47889a5a1`. Możesz wykorzystać narzędzie `hashid`:

```
docker run mazurkatarzyna/hashid:latest yourhash
```

Czy `hashid` odnalazł nazwę hasha? Jeśli nie, zaproponuj inny sposób na rozwiązanie zadania.

1.13 Mając dany początkowy ciąg znaków `helloworld`, który następnie został zahashowany, określ, jaka funkcja skrótu została wykorzystana do utworzenia hasha: `$2y$10$xbyAv5a46CQYPay5UISCNeFWpVdx2qvhCBEOZ/YtfxoVXhOGrVKQa`. Możesz wykorzystać narzędzie `hashid`:

```
docker run mazurkatarzyna/hashid:latest yourhash
```

Czy `hashid` odnalazł nazwę hasha? Jeśli nie, zaproponuj inny sposób na rozwiązanie zadania.

1.14 Mając dany początkowy ciąg znaków `R3iSrSNmgU9SFHxVekUD`, który następnie został zahashowany, określ, jaka funkcja skrótu została wykorzystana do utworzenia hasha: `48cab4b54bef42fddaa6353c68a20b369f40026e`. Aby rozwiązać zadanie, napisz skrypt w języku Python. Podpowiedź: algorytm jest dostępny w bibliotece `hashlib`.