UPB  
Zadanie 2 - Kryptografia

Riešitelia: Emília Čurillová, Filip Harvančík, Lenka Puškášová

**Úloha1** (Filip 25%, Lenka 75%)

Pre tento projekt bude použití knižnica cryptography z nasledujúcich dôvodov

* Je aktívne vyvíjaná a údržovaná a reaguje na bezpečnostné zraniteľnosti.
* Je široko používaná v Python ekosystéme a odporúčaná komunitou.
* Dokumentácia otvorene uvádza zoznam známych zraniteľností a používa externé databázy (napr. OSV) na ich sledovanie.

Inštalácia: **pip install cryptography**

Zdroje:

<https://cryptography.io/en/latest/security>

**Úloha2** (Lenka)

Pre Generovanie asymetrického páru kľúčov bude použitý Rivest-Shamir-Adleman (RSA) algoritmus z dôvodu jeho rozšíreného použitia, robustnosti a bezpečnosti

Pre účely zašifrovania kľúčov bude vytvorený nový stĺpec do tabuľky – **password**  
Použitá knižnica: werkzeug.security, ktorá je súčasťou široko používanej Flask knižnice

**Postup vytvorenia páru kľúčov:**

1. Vytvorenie private key
   1. bude použitá funkcia **rsa.generate\_private\_key(**public\_exponent, key\_size)
      1. **public\_exponent** = [65537](https://www.daemonology.net/blog/2009-06-11-cryptographic-right-answers.html) – nie je to nevyhnutne ale tato hodnota je odporúčaná v dokumentácii z legacy dôvodov
      2. **key\_size** = 2048 – odporúčaná minimálna hodnota pre veľkosť kľúča
2. Odvodenie public key z private key
   1. **private\_key.public\_key()**
3. Serializácia kľúčov – kľúče sú uložené ako RSAPrivateKey a RSAPublicKey objekty. Treba ich serializovat, aby mali bajtový format a mohli byť uložené
   1. **private\_key.private\_bytes**( encoding, formate, encryption\_algorithm)
      1. encoding – PEM (base64)
      2. formate – PKCS8 (štandardne používaný format pre privátne kľúče)
      3. encryption\_algorithm - serialization. NoEncryption()
   2. public\_key.public\_bytes(encoding, formate)
      1. encoding – PEM
      2. formate – SubjectPublicKeyInfo – najčastejšie používaný pre verejné kľúče
4. Vytvorenie užívateľa – do db sa bude pridávať nový užívateľ (alebo upravovať už existujúci) s uživateľským menom user, heslom password a s verejným heslom public\_key (využije sa metóda decode na prevod z bajtov na string typ)
5. V api volaní sa vykoná generovanie privátneho a public kľúča. Nastane serializácia a vytvorí sa nový užívateľ spolu s public key v db tabuľke. Následne metoda vracia odpoved v súbore s využitím Content-Disposition v hlavičke odpovedi

Zdroje:  
RSA algoritmus:   
https://www.ibm.com/think/topics/asymmetric-encryption  
Privátny kľúč: <https://cryptography.io/en/latest/hazmat/primitives/asymmetric/rsa/#cryptography.hazmat.primitives.asymmetric.rsa.generate_private_key>  
Serializácia kľúčov:  
<https://cryptography.io/en/latest/hazmat/primitives/asymmetric/serialization/#serialization-of-private-keys>  
Serialization encoding  
<https://cryptography.io/en/latest/hazmat/primitives/asymmetric/serialization/#serialization-encodings>  
Serialization formats  
<https://cryptography.io/en/latest/hazmat/primitives/asymmetric/serialization/#serialization-formats>  
Serialization Encryption Types  
<https://cryptography.io/en/latest/hazmat/primitives/asymmetric/serialization/#serialization-encryption-types>  
werkzeug.sercurity library  
<https://werkzeug.palletsprojects.com/en/latest/utils/#module-werkzeug.security>  
password hashing  
<https://dev.to/goke/securing-your-flask-application-hashing-passwords-tutorial-2f0p>  
sqlalchemy ORM  
<https://docs.sqlalchemy.org/en/20/tutorial/orm_data_manipulation.html>  
Content-disposition header  
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Content-Disposition>