

Zadanie 3


I-UPB

18. października 2021

Úvod

Riešenie je naprogramované v Pythone a API použité pri šifrovaní sú z [pycrypto](#).

Spustenie aplikácie

Pred spustením programu je potrebné doinštalovať knižnice z **requirements.txt**. Riešenie neobsahuje žiadne GUI len CLI. 
K aplikácii som priložil aj 2 súbory na otestovanie celého procesu.

Inštalácia

Inštalácia je nasledovná:

```
git clone https://github.com/behindbone/upb-03.git
```

(všetky súbory sú aj priložené k tomuto dokumentu)

```
pip install -r requirements.txt
```

Spustenie

```
python3 app.py [FILE]
```

Aplikácia šifruje aj dešifruje vstupný súbor. Vytvorí zašifrovaný súbor (*encrypted.aes*) a .pem súbory pre kľúče (*RSA_private_key.pem*, *RSA_public_key.pem*).

Šifrovanie

```
1 public_ = RSA.generate(3072)
2 private_ = public_.publickey()
3 pubKeyPEM = private_.exportKey()
4 privKeyPEM = public_.exportKey()
5
6 with open ("RSA_private_key.pem", "wb") as prv_file:
7     prv_file.write(privKeyPEM)
8 with open ("RSA_public_key.pem", "wb") as pub_file:
9     pub_file.write(pubKeyPEM)
```

Listing 1: Generovanie RSA keypairu

Následne sa vygeneruje k tvorbe kľúču aj **salt** (náhodných 16 bajtov). Na tvorbu kľúča som použil funkciu [PBKDF2](#).

```
1 # Encrypt using AES GCM
2 cipher = AES.new(key, AES.MODE_GCM)
3 cipher.update(public_key.encode("utf8"))
4 ciphertext, tag = cipher.encrypt_and_digest(input_data)
5
6 _message = public_key, ciphertext, tag, cipher.nonce, salt
7 with open("encrypted.aes", 'wb') as f:
8     for var in _message:
9         if isinstance(var, str):
10             f.write(bytearray(bytes(var, encoding='utf-8')))
11         else:
12             f.write(bytearray(bytes(var)))
```

Listing 2: Ukážka šifrovania

Na šifrovanie je zvolený mód AES GCM. K šifre je náhodne generovaná hodnota na autentifikáciu (*nonce*).

```
1 received_pubKey = received_msg[:624]
2 received_ciphertext = received_msg[624:len(received_msg)-3*16]
3 received_tag = received_msg[len(received_msg)-3*16:len(received_msg)-2*16]
4 received_nonce = received_msg[len(received_msg)-2*16:len(received_msg)-1*16]
5 received_kdf_salt = received_msg[len(received_msg)-1*16:len(received_msg)]
```

Listing 3: Čítanie RSA private key zo súboru

V ukážke kódu 3 parsujem načítané bajty zo súboru *RSA_private_key.pem*. Za integritu dát zodpovedá premenná *received_tag*. Na obrázku 1 je zobrazené koľko bajtov predstavuje každý z blokov zašifrovaného súboru.

624	N	16	16	16
RSA Public Key	File Content	Tag	Nonce	Salt

Obr. 1: Štruktúra zašifrovaného súboru podľa zadania

Dešifrovanie

Pri dešifrovaní v ukážke 4 najprv prečítam private key zo súboru, potom opäť použijem funkciu PBKDF2.

```
1 # Decrypt
2 with open("RSA_private_key.pem", 'r') as f:
3     RSA_private_key_file = f.read()
4 decryption_key = PBKDF2(RSA_private_key_file, received_kdf_salt)
5 cipher = AES.new(decryption_key, AES.MODE_GCM, received_nonce)
6 cipher.update(received_pubKey)
```

Listing 4: Čítanie RSA private key zo súboru

Validácia

Ako posledné kontrolujem aj integritu súboru nasledovne:

```
1 # Validate
2 try:
3     decrypted_data =
4         cipher.decrypt_and_verify(received_ciphertext, received_tag)
5     print ("\nZachovanie integrity. MAC validácia úspešná.")
6     with open('decrypted_' + filename, 'wb') as f:
7         data = f.write(decrypted_data)
8     print ("Dešifrovaný súbor uložený ako {}".
9           .format('decrypted_' + filename))
10 except ValueError as mac_mismatch:
11     print ("\nPorušenie integrity. MAC validácia neúspešná pri dešifrovaní.")
```

Listing 5: Čítanie RSA private key zo súboru

Zdroje

- <https://github.com/wolf43/AES-GCM-example>
- https://www.programcreek.com/python/example/88000/Crypto.Cipher.AES.MODE_GCM
- <https://nitratine.net/blog/post/python-gcm-encryption-tutorial/>