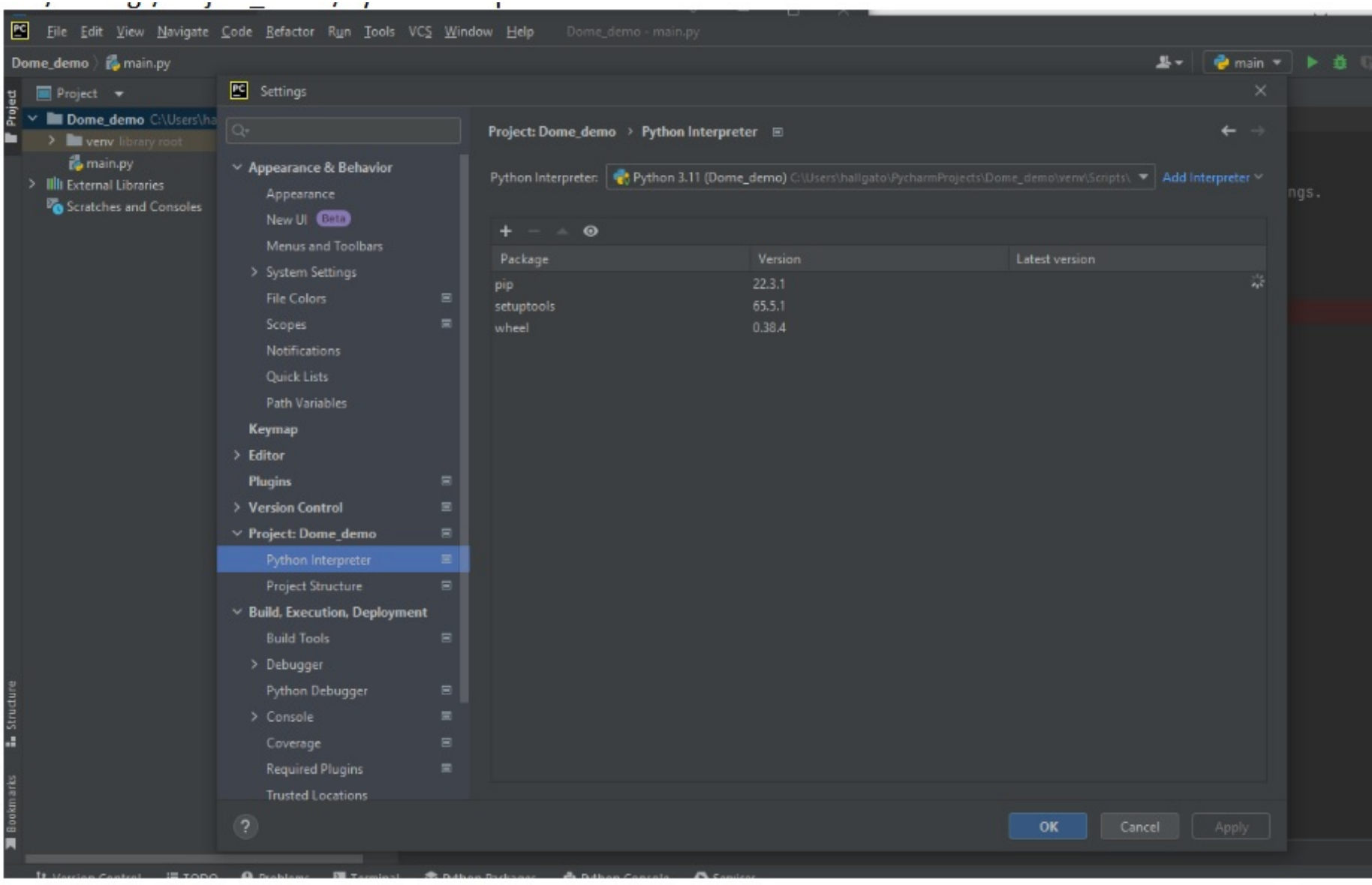
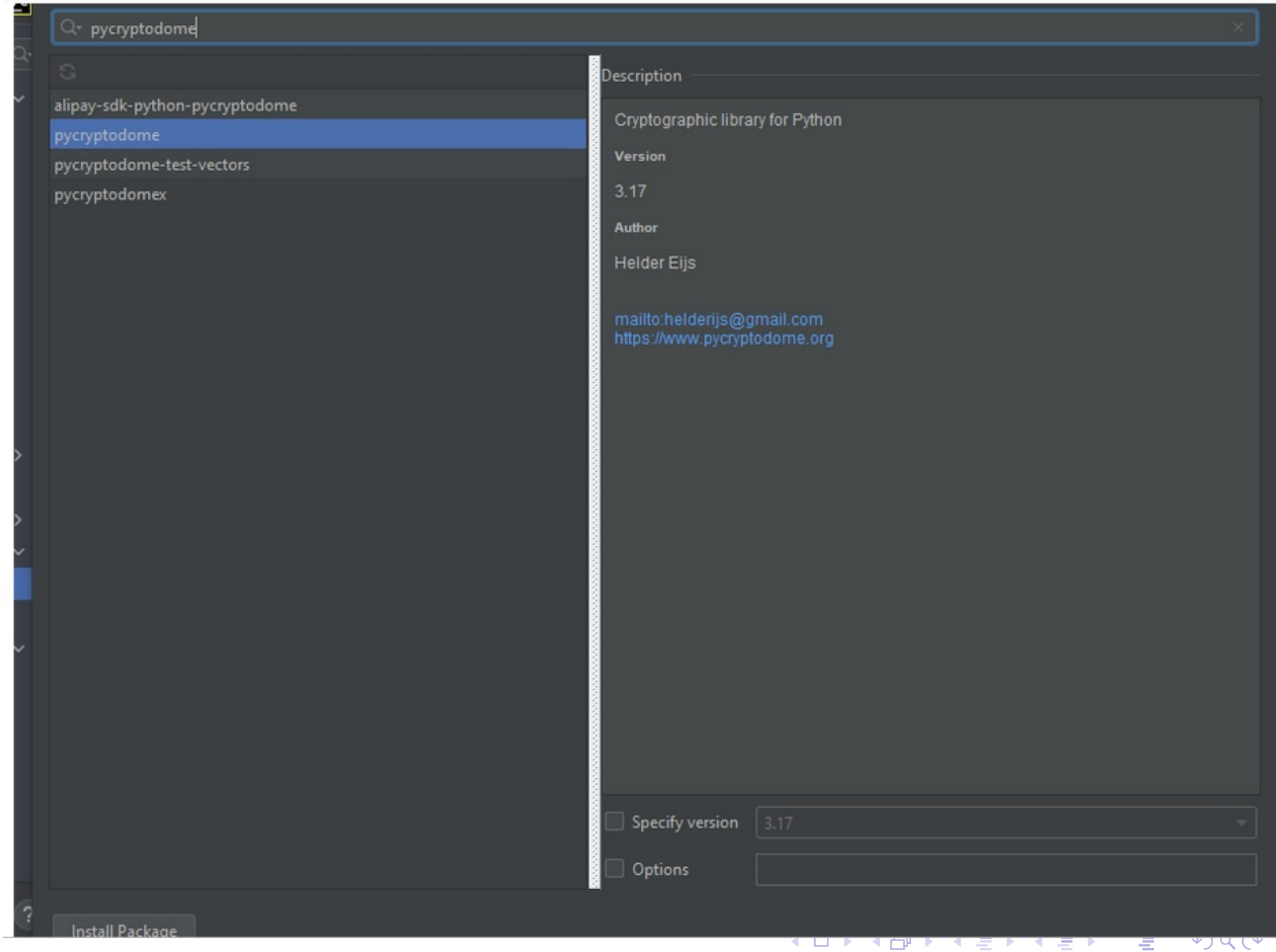
Kriptocsomagok használata

Szimmetrikus titkosítás: AES és Single and Triple DES (legacy)
Authenticated Encryption: GCM (AES only)
Aszimmetrikus titkosítás, aláírás: Asymmetric key generation: RSA, DSA; Asymmetric ciphers: PKCS1 (RSA)
RSAES-PKCS1-v1$_5$, $PKCS1(RSA)RSASSA - PKCS1 - v1_5$

A python pycrpytodome csomaghoz az első lépés annak beimportálása és telepítése:
File/Settings/Project$_n$ame/*Pythoninterpreter*


File Edit View Navigate Code Refactor Run Tools VCS Window Help
Dome_demo - main.py
Dome_demo main.py
main
Project
Dome_demo C:\Users\ha
venv library root
main.py
External Libraries
Scratches and Consoles
Settings
Appearance & Behavior
Appearance
New UI Beta
Menus and Toolbars
System Settings
File Colors
Scopes
Notifications
Quick Lists
Path Variables
Keymap
Editor
Plugins
Version Control
Project: Dome_demo
Python Interpreter
Project Structure
Build, Execution, Deployment
Build Tools
Debugger
Python Debugger
Console
Coverage
Required Plugins
Trusted Locations
Project: Dome_demo › Python Interpreter
Python Interpreter: Python 3.11 (Dome_demo) C:\Users\hallgato\PycharmProjects\Dome_demo\venv\Scripts\
Add Interpreter
Package
Version
Latest version
pip
22.3.1
setuptools
65.5.1
wheel
0.38.4
OK
Cancel
Apply
Structure
Bookmarks

plusz gomb: pycryptodome install



AES:
https://pycryptodome.readthedocs.io/en/latest/src/cipher/aes.html?highlight=AES#aes

```
from Crypto.Cipher import AES
>>>
>>> key = b'Sixteen byte key'
>>> cipher = AES.new(key, AES.MODE_EAX, nonce=nonce)
>>> plaintext = cipher.decrypt(ciphertext)
>>> try:
>>>     cipher.verify(tag)
>>>     print("The message is authentic:", plaintext)
>>> except ValueError:
>>>     print("Key incorrect or message corrupted")
```

AES GCM:
https://pycryptodome.readthedocs.io/en/latest/src/cipher/modern.html#gcm-mode

```
import json
>>> from base64 import b64encode
>>> from Crypto.Cipher import AES
>>> from Crypto.Random import get_random_bytes
>>>
>>> header = b"header"
>>> data = b"secret"
>>> key = get_random_bytes(16)
>>> cipher = AES.new(key, AES.MODE_GCM)
>>> cipher.update(header)
>>> ciphertext, tag = cipher.encrypt_and_digest(data)
>>>
>>> json_k = [ 'nonce', 'header', 'ciphertext', 'tag' ]
>>> json_v = [ b64encode(x).decode('utf-8') for x in (cipher.nonce, header, ciphertext, tag) ]
>>> result = json.dumps(dict(zip(json_k, json_v)))
>>> print(result)
{"nonce": "DpOK8NIOuSOQlTq+BphKWw==", "header": "aGVhZGVy", "ciphertext": "CZVqyacc", "tag":
"B2tBgICbyw+Wji9KpLVa8w=="}
```

```
import json
>>> from base64 import b64decode
>>> from Crypto.Cipher import AES
>>> from Crypto.Util.Padding import unpad
>>>
>>> # We assume that the key was securely shared beforehand
>>> try:
>>>     b64 = json.loads(json_input)
>>>     json_k = [ 'nonce', 'header', 'ciphertext', 'tag' ]
>>>     jv = {k:b64decode(b64[k]) for k in json_k}
>>>
>>>     cipher = AES.new(key, AES.MODE_GCM, nonce=jv['nonce'])
>>>     cipher.update(jv['header'])
>>>     plaintext = cipher.decrypt_and_verify(jv['ciphertext'], jv['tag'])
>>>     print("The message was: " + plaintext.decode('utf-8'))
>>> except (ValueError, KeyError):
>>>     print("Incorrect decryption")
```

3DES:
https://pycryptodome.readthedocs.io/en/latest/src/cipher/des3.html?highlight=des

```
from Crypto.Cipher import DES3
>>> from Crypto.Random import get_random_bytes
>>>
>>> # Avoid Option 3
>>> while True:
>>>     try:
>>>         key = DES3.adjust_key_parity(get_random_bytes(24))
>>>         break
>>>     except ValueError:
>>>         pass
>>>
>>> cipher = DES3.new(key, DES3.MODE_CFB)
>>> plaintext = b'We are no longer the knights who say ni!'
>>> msg = cipher.iv + cipher.encrypt(plaintext)
```

Asymmetric key generation:RSA:
https://pycryptodome.readthedocs.io/en/latest/src/public_key/rsa.html?highlight=rsa

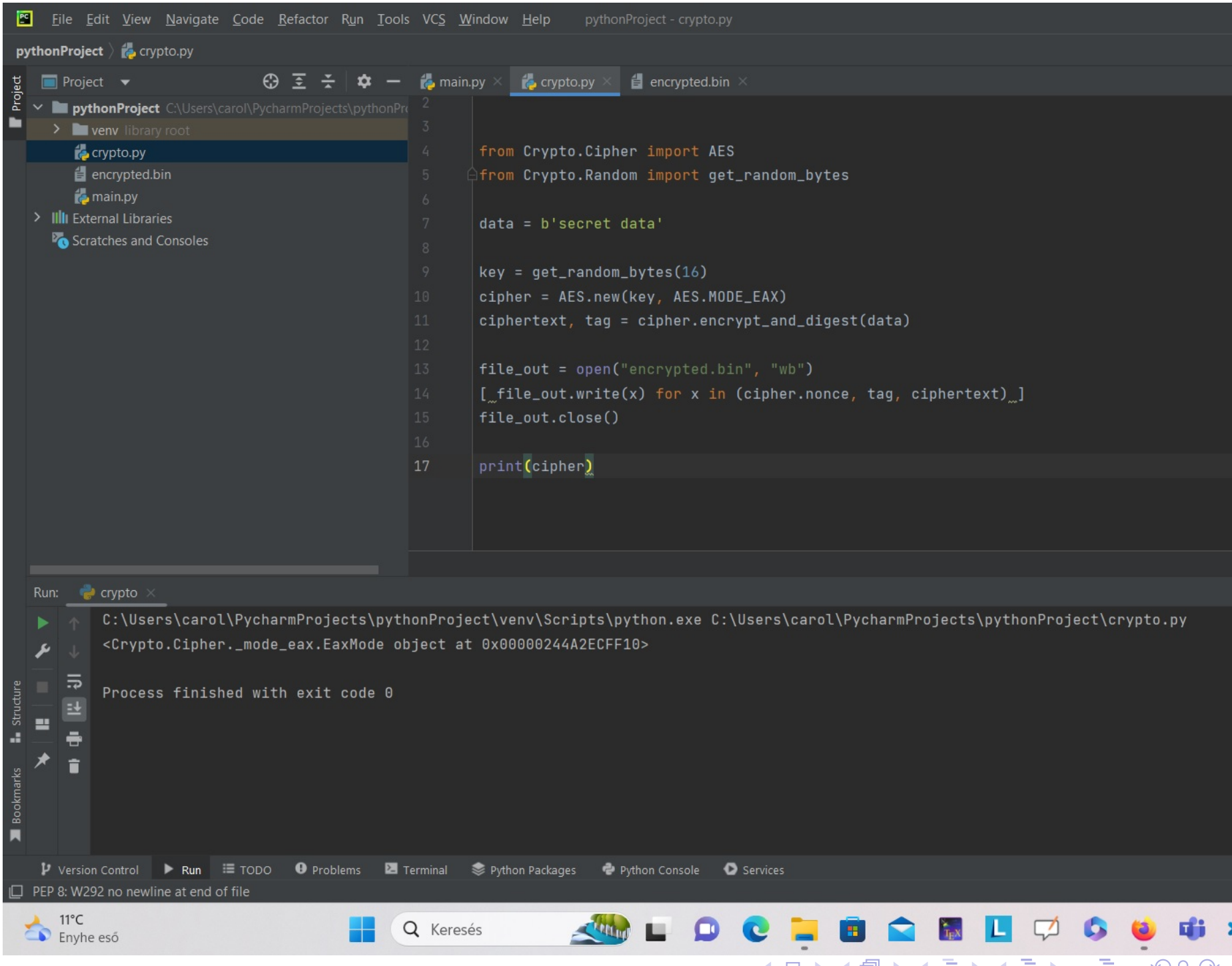
```
from Crypto.PublicKey import RSA
>>>
>>> key = RSA.generate(2048)
>>> f = open('mykey.pem','wb')
>>> f.write(key.export_key('PEM'))
>>> f.close()
```

1024bit javasolt (kisebb bitnél hibaüzenetet dob ki)

DSA

https://pycryptodome.readthedocs.io/en/latest/src/public_key/dsa.html?highlight=dsa

```
from Crypto.PublicKey import DSA
>>> from Crypto.Signature import DSS
>>> from Crypto.Hash import SHA256
>>>
>>> # Create a new DSA key
>>> key = DSA.generate(2048)
>>> f = open("public_key.pem", "w")
>>> f.write(key.publickey().export_key())
>>> f.close()
>>>
>>> # Sign a message
>>> message = b"Hello"
>>> hash_obj = SHA256.new(message)
>>> signer = DSS.new(key, 'fips-186-3')
>>> signature = signer.sign(hash_obj)
>>>
>>> # Load the public key
>>> f = open("public_key.pem", "r")
>>> hash_obj = SHA256.new(message)
>>> pub_key = DSA.import_key(f.read())
>>> verifier = DSS.new(pub_key, 'fips-186-3')
>>>
>>> # Verify the authenticity of the message
>>> try:
>>>     verifier.verify(hash_obj, signature)
>>>     print "The message is authentic."
>>> except ValueError:
>>>     print "The message is not authentic."
```


File Edit View Navigate Code Refactor Run Tools VCS Window Help
pythonProject - crypto.py
pythonProject
crypto.py
Project
main.py
crypto.py
encrypted.bin
pythonProject C:\Users\carol\PycharmProjects\pythonPr
venv library root
crypto.py
encrypted.bin
main.py
External Libraries
Scratches and Consoles
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
data = b'secret data'
key = get_random_bytes(16)
cipher = AES.new(key, AES.MODE_EAX)
ciphertext, tag = cipher.encrypt_and_digest(data)
file_out = open("encrypted.bin", "wb")
[file_out.write(x) for x in (cipher.nonce, tag, ciphertext)]
file_out.close()
print(cipher)
Run:
crypto
C:\Users\carol\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\carol\PycharmProjects\pythonProject\crypto.py
<Crypto.Cipher._mode_eax.EaxMode object at 0x00000244A2ECFF10>
Process finished with exit code 0
Structure
Bookmarks
Version Control
Run
TODO
Problems
Terminal
Python Packages
Python Console
Services
PEP 8: W292 no newline at end of file
11°C
Enyhe eső
Keresés