DATA ENCRYPTION

PGP & PYTHON

Encryption Programs

1. GnuPG - The GNU Privacy Guard
   - Programs & Frameworks for managing encryption & decryption

2. Python-gnupg - A python Wrapper for GnuPG
   - Leverages functions provided by GPG

PGP Explained

PGP

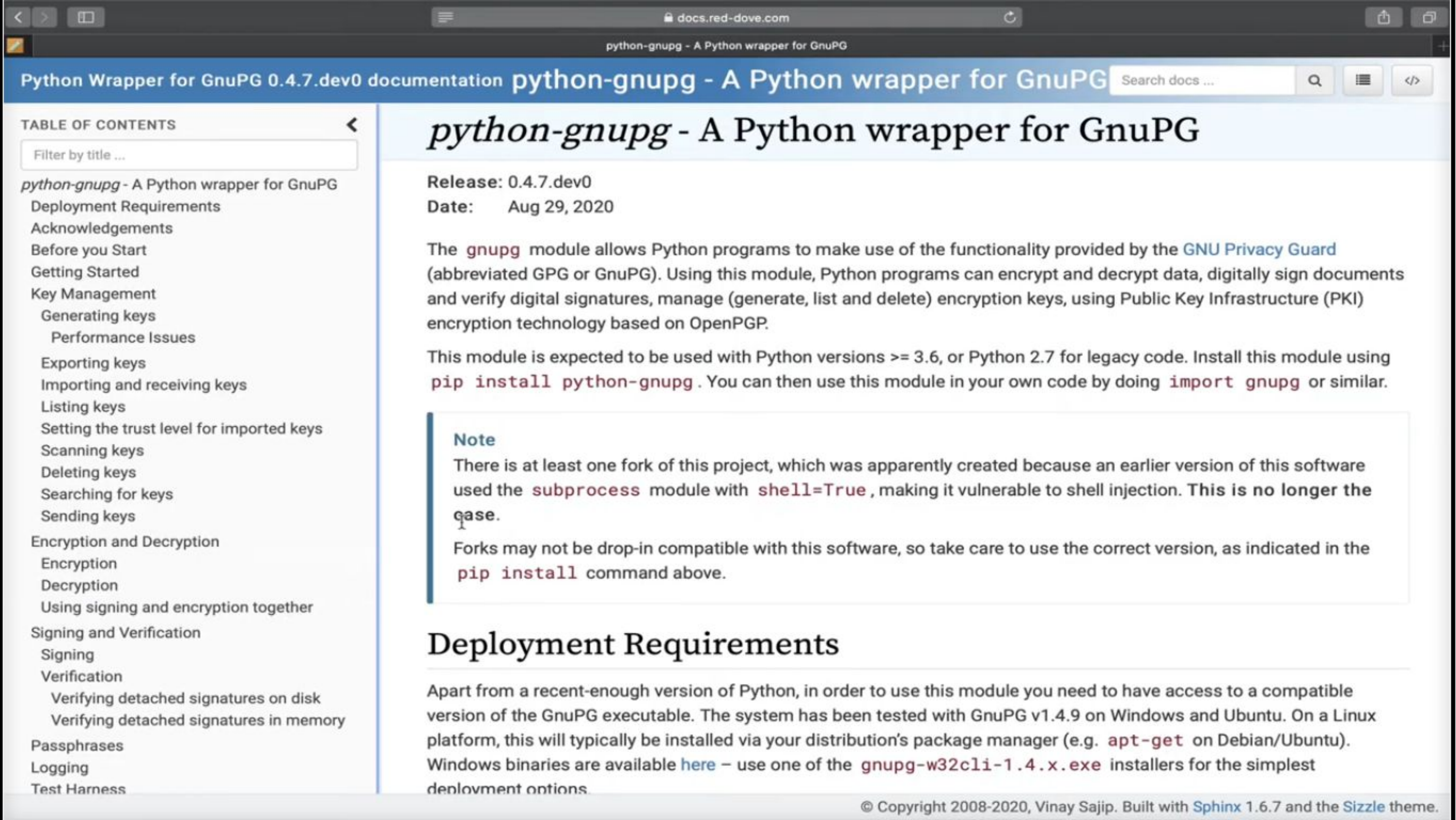
Developed in the early 90s. Currently owned by Symantec.

Open PGP

Defines standard formats for encrypted messages, signatures and certificates for exchanging public keys.

GNU PG

Free implementation of the OpenPGP standard that allows encryption and signing of data and communications.


docs.red-dove.com
python-gnupg - A Python wrapper for GnuPG
Python Wrapper for GnuPG 0.4.7.dev0 documentation
Search docs ...
TABLE OF CONTENTS
Filter by title ...

python-gnupg - A Python wrapper for GnuPG

Release: 0.4.7.dev0
Date: Aug 29, 2020

The `gnupg` module allows Python programs to make use of the functionality provided by the GNU Privacy Guard (abbreviated GPG or GnuPG). Using this module, Python programs can encrypt and decrypt data, digitally sign documents and verify digital signatures, manage (generate, list and delete) encryption keys, using Public Key Infrastructure (PKI) encryption technology based on OpenPGP.

This module is expected to be used with Python versions >= 3.6, or Python 2.7 for legacy code. Install this module using `pip install python-gnupg`. You can then use this module in your own code by doing `import gnupg` or similar.

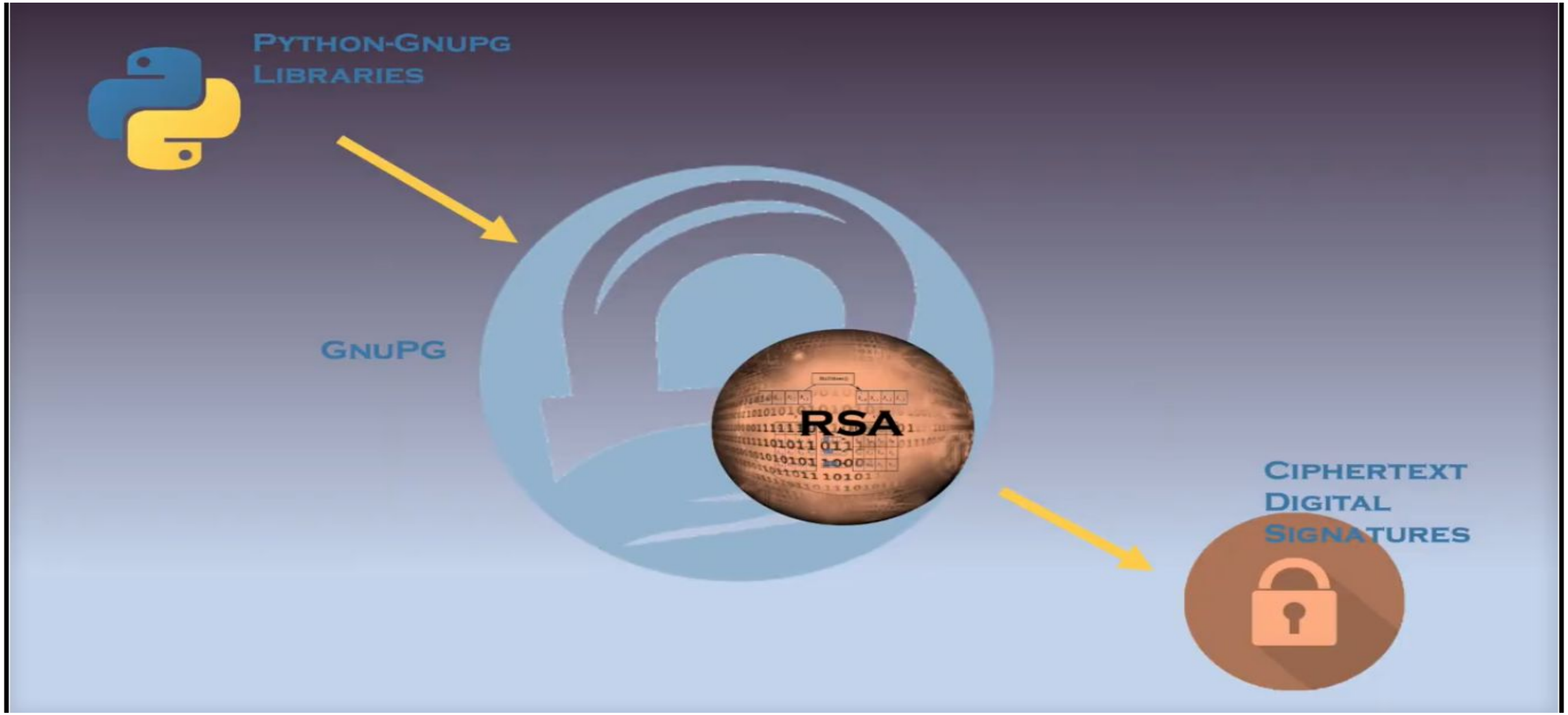
Note
There is at least one fork of this project, which was apparently created because an earlier version of this software used the `subprocess` module with `shell=True`, making it vulnerable to shell injection. **This is no longer the case.**

Forks may not be drop-in compatible with this software, so take care to use the correct version, as indicated in the `pip install` command above.

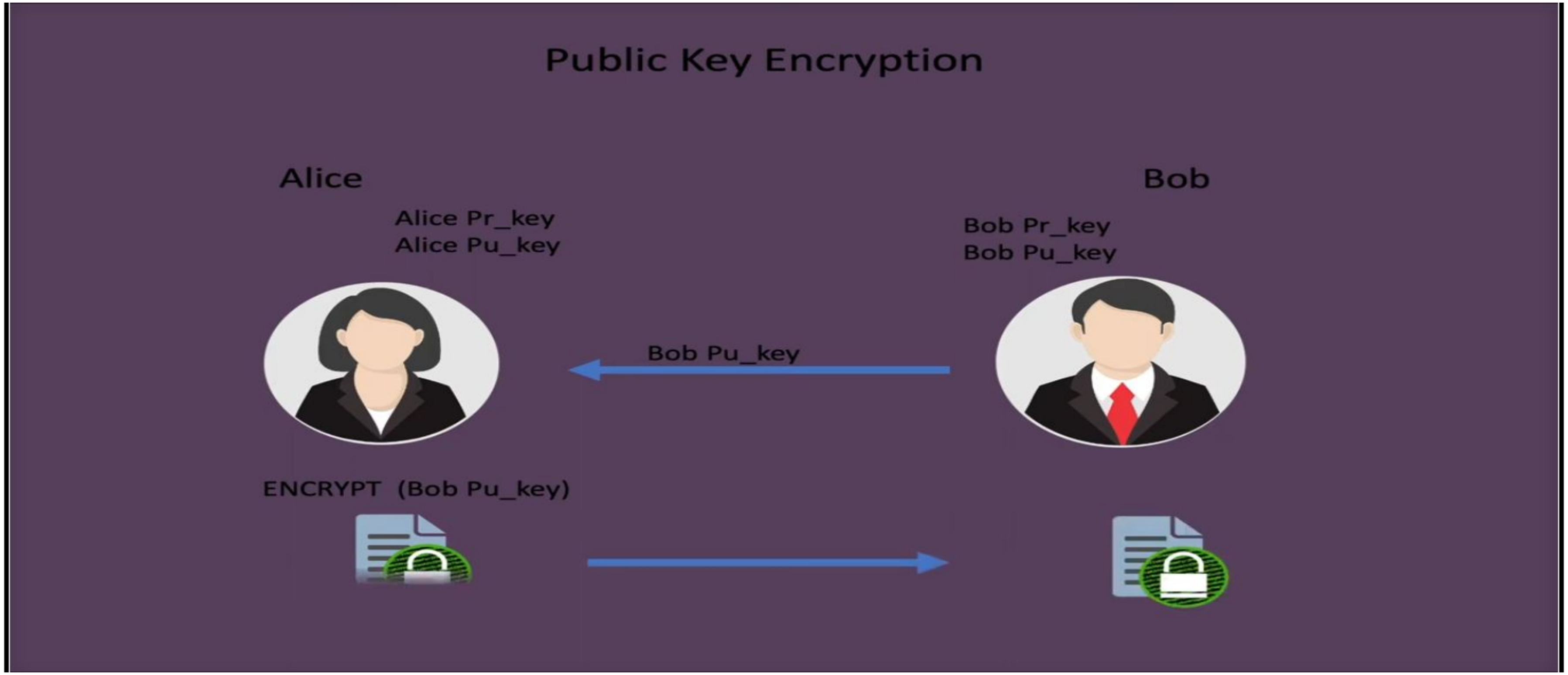
Deployment Requirements

Apart from a recent-enough version of Python, in order to use this module you need to have access to a compatible version of the GnuPG executable. The system has been tested with GnuPG v1.4.9 on Windows and Ubuntu. On a Linux platform, this will typically be installed via your distribution's package manager (e.g. `apt-get` on Debian/Ubuntu). Windows binaries are available here – use one of the `gnupg-w32cli-1.4.x.exe` installers for the simplest deployment options

PROGRAM STRUCTURE



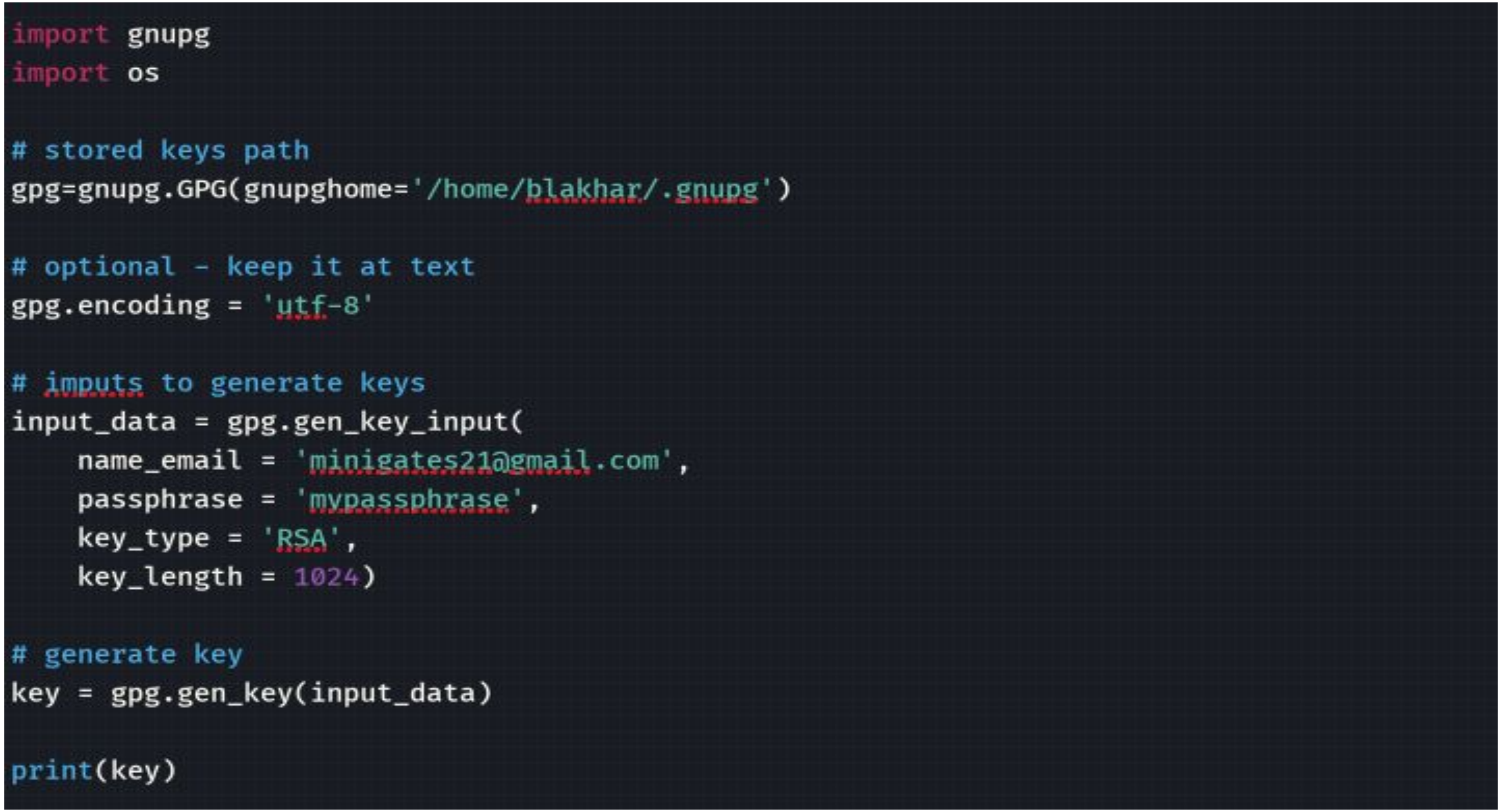
PUBLIC KEY ENCRYPTION



Encryption Steps

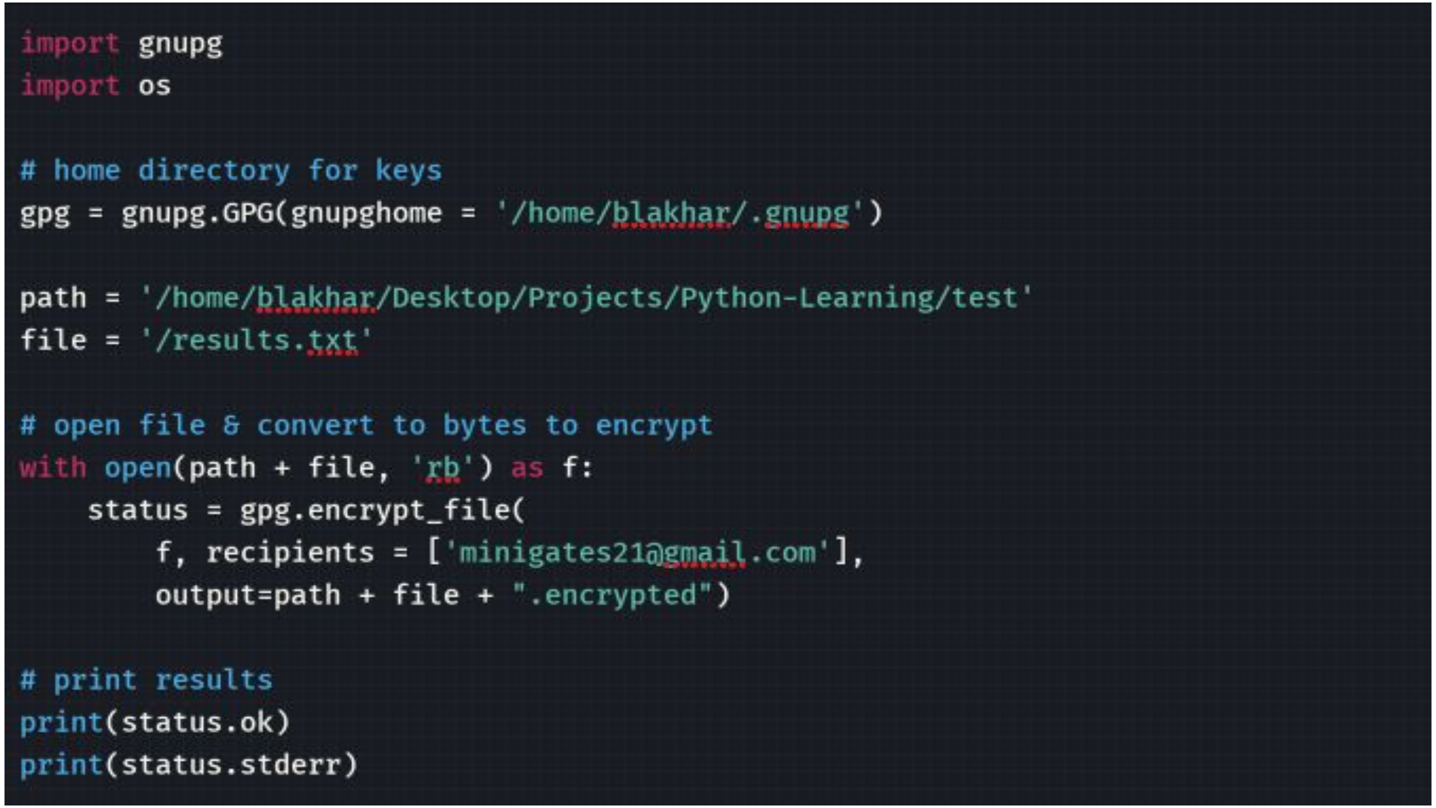
- Generate keys
- Encrypt with our public key
- Decrypt with private key
- Import keys from other users
- Sign a document
- Verify signature

Generate Keys



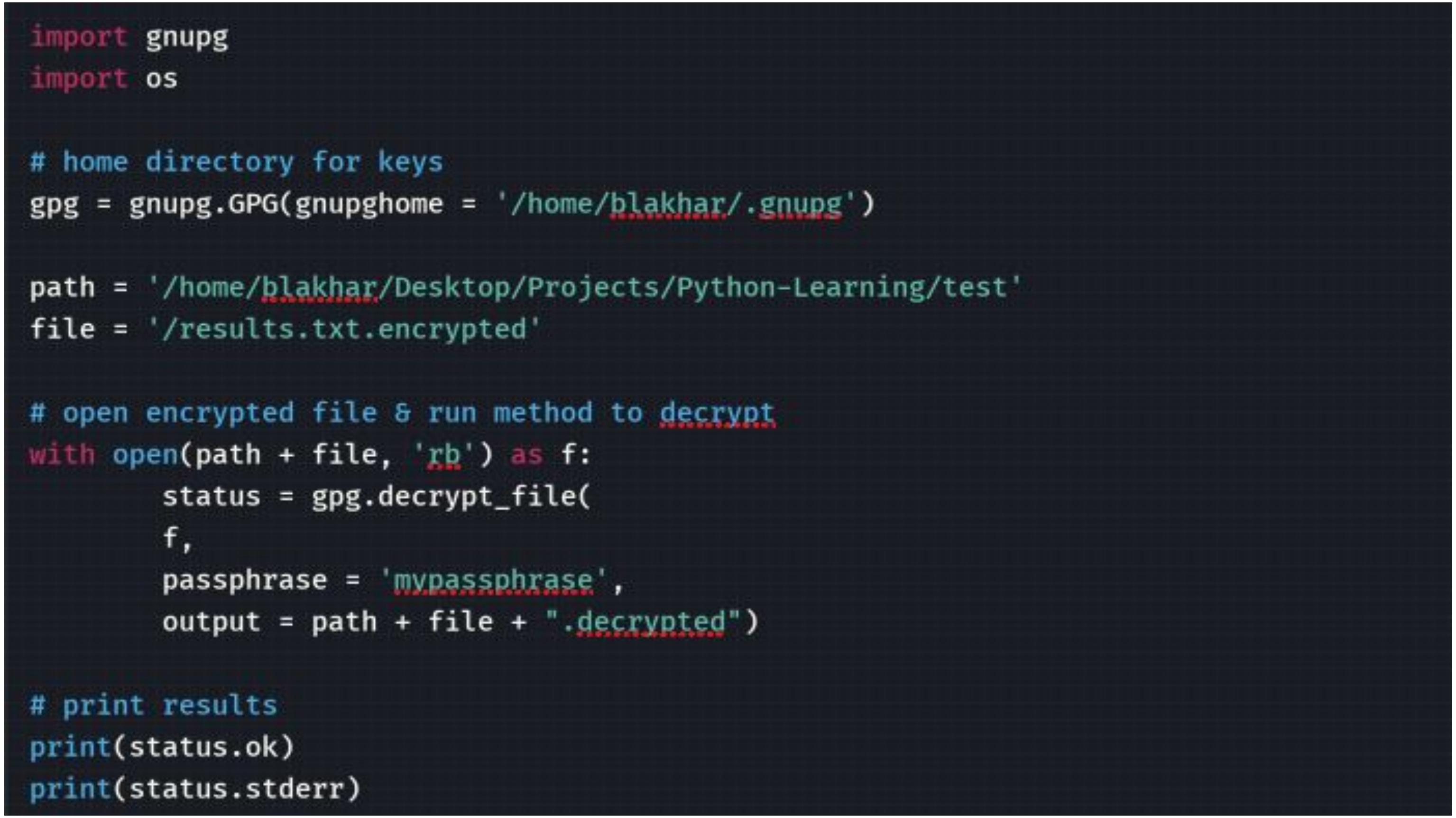
```python
import gnupg
import os

# stored keys path
gpg=gnupg.GPG(gnupghome='/home/blakhar/.gnupg')

# optional - keep it at text
gpg.encoding = 'utf-8'

# imputs to generate keys
input_data = gpg.gen_key_input(
    name_email = 'minigates21@gmail.com',
    passphrase = 'mypassphrase',
    key_type = 'RSA',
    key_length = 1024)

# generate key
key = gpg.gen_key(input_data)

print(key)
```

Encrypt File - Private Key



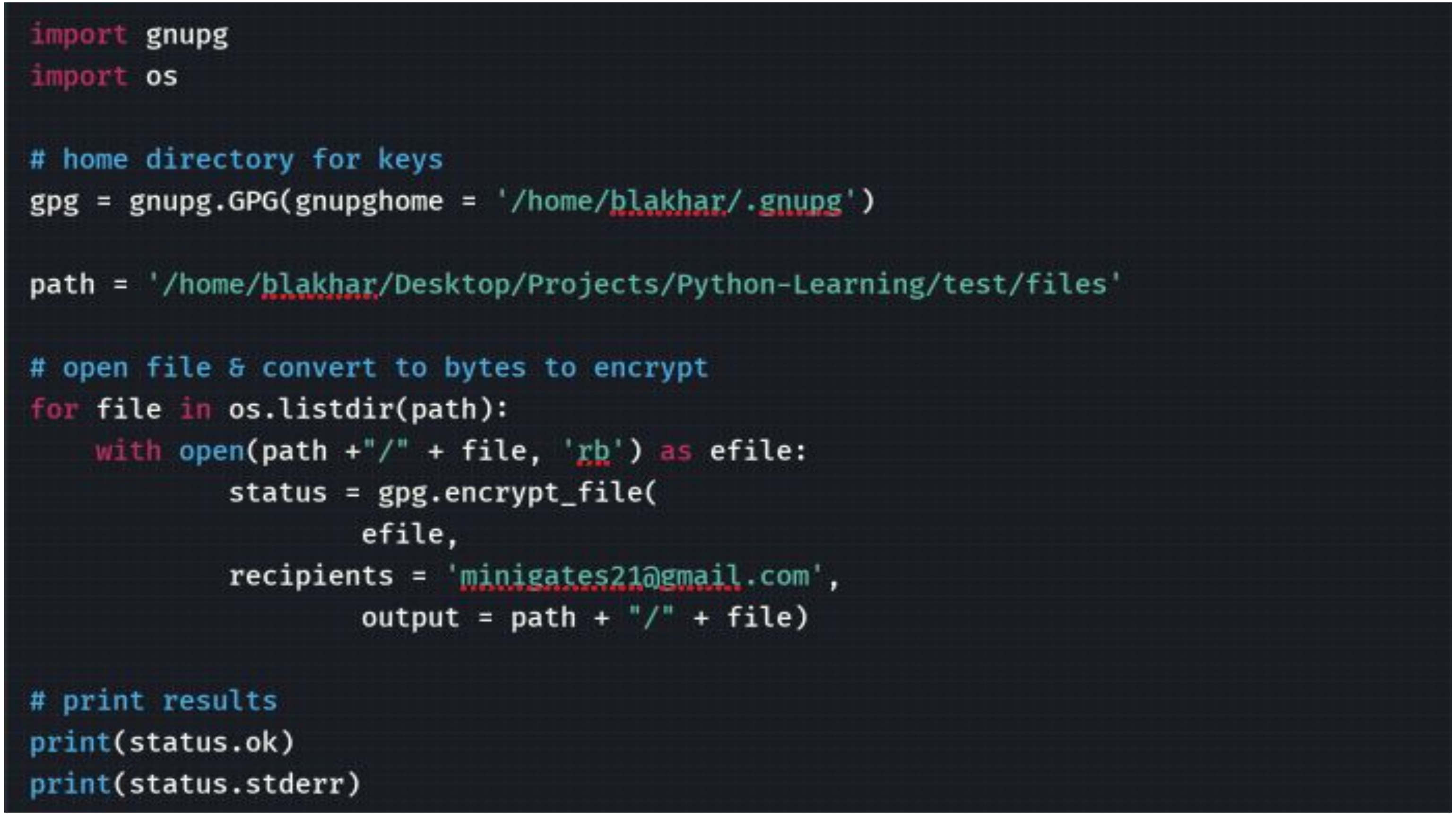
```python
import gnupg
import os

# home directory for keys
gpg = gnupg.GPG(gnupghome = '/home/blakhar/.gnupg')

path = '/home/blakhar/Desktop/Projects/Python-Learning/test'
file = '/results.txt'

# open file & convert to bytes to encrypt
with open(path + file, 'rb') as f:
    status = gpg.encrypt_file(
        f, recipients = ['minigates21@gmail.com'],
        output=path + file + ".encrypted")

# print results
print(status.ok)
print(status.stderr)
```

Decrypt File - Public Key



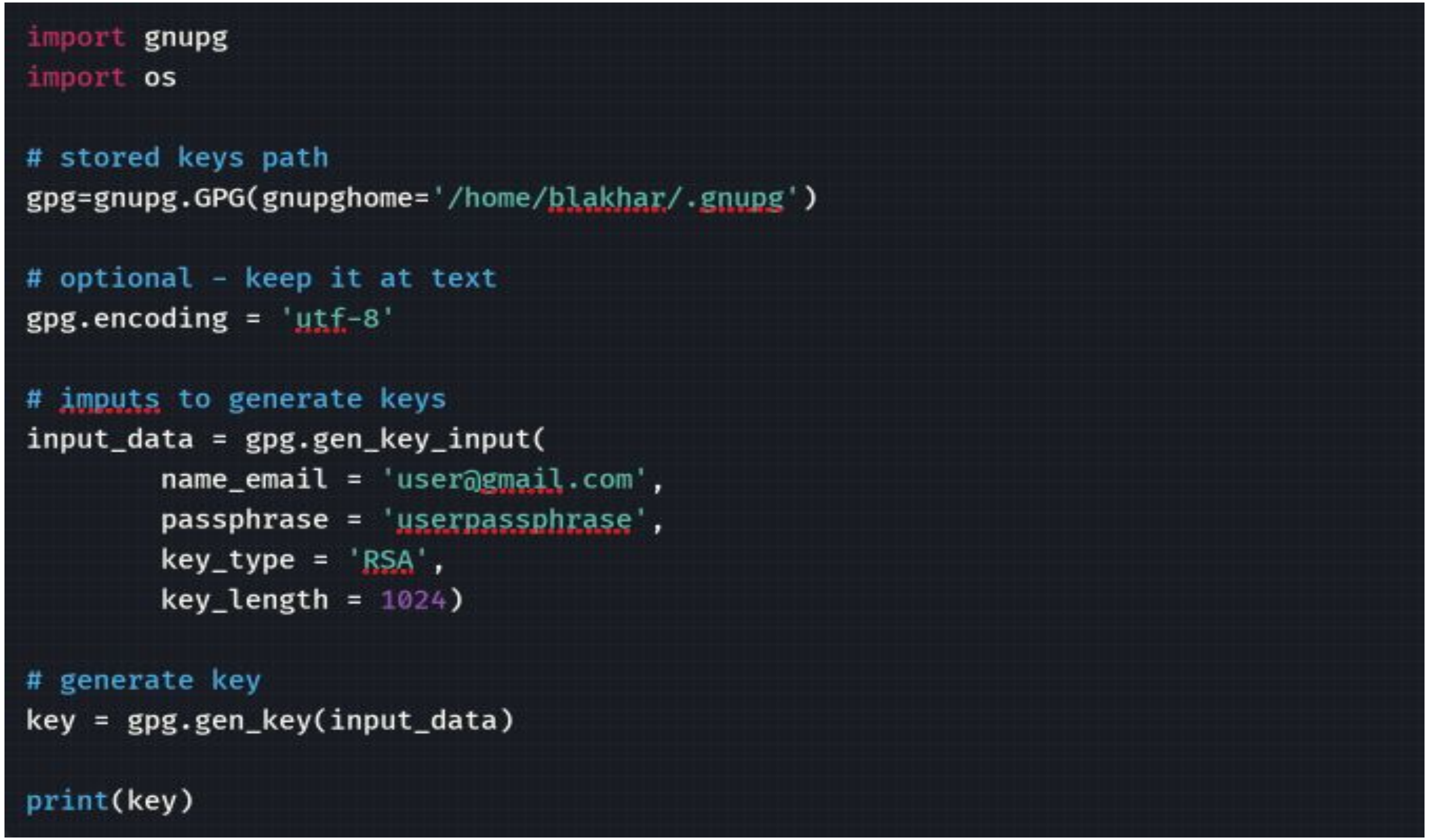
```python
import gnupg
import os

# home directory for keys
gpg = gnupg.GPG(gnupghome = '/home/blakhar/.gnupg')

path = '/home/blakhar/Desktop/Projects/Python-Learning/test'
file = '/results.txt.encrypted'

# open encrypted file & run method to decrypt
with open(path + file, 'rb') as f:
        status = gpg.decrypt_file(
        f,
        passphrase = 'mypassphrase',
        output = path + file + ".decrypted")

# print results
print(status.ok)
print(status.stderr)
```

Encrypt Multiple Files



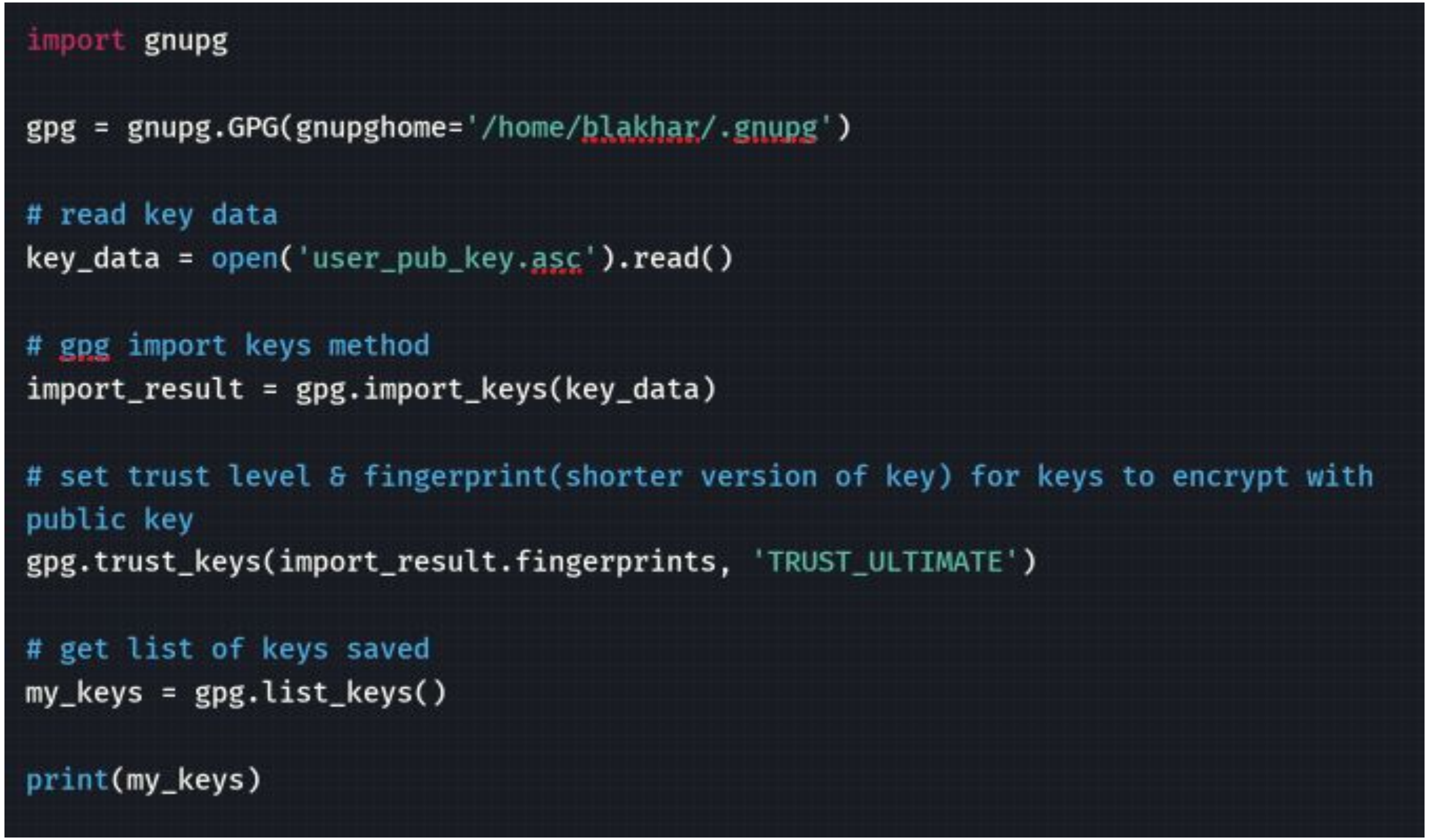
```
import gnupg
import os

# home directory for keys
gpg = gnupg.GPG(gnupghome = '/home/blakhar/.gnupg')

path = '/home/blakhar/Desktop/Projects/Python-Learning/test/files'

# open file & convert to bytes to encrypt
for file in os.listdir(path):
    with open(path +"/" + file, 'rb') as efile:
            status = gpg.encrypt_file(
                    efile,
            recipients = 'minigates21@gmail.com',
                    output = path + "/" + file)


# print results
print(status.ok)
print(status.stderr)
```

Encryption Keys & Verification



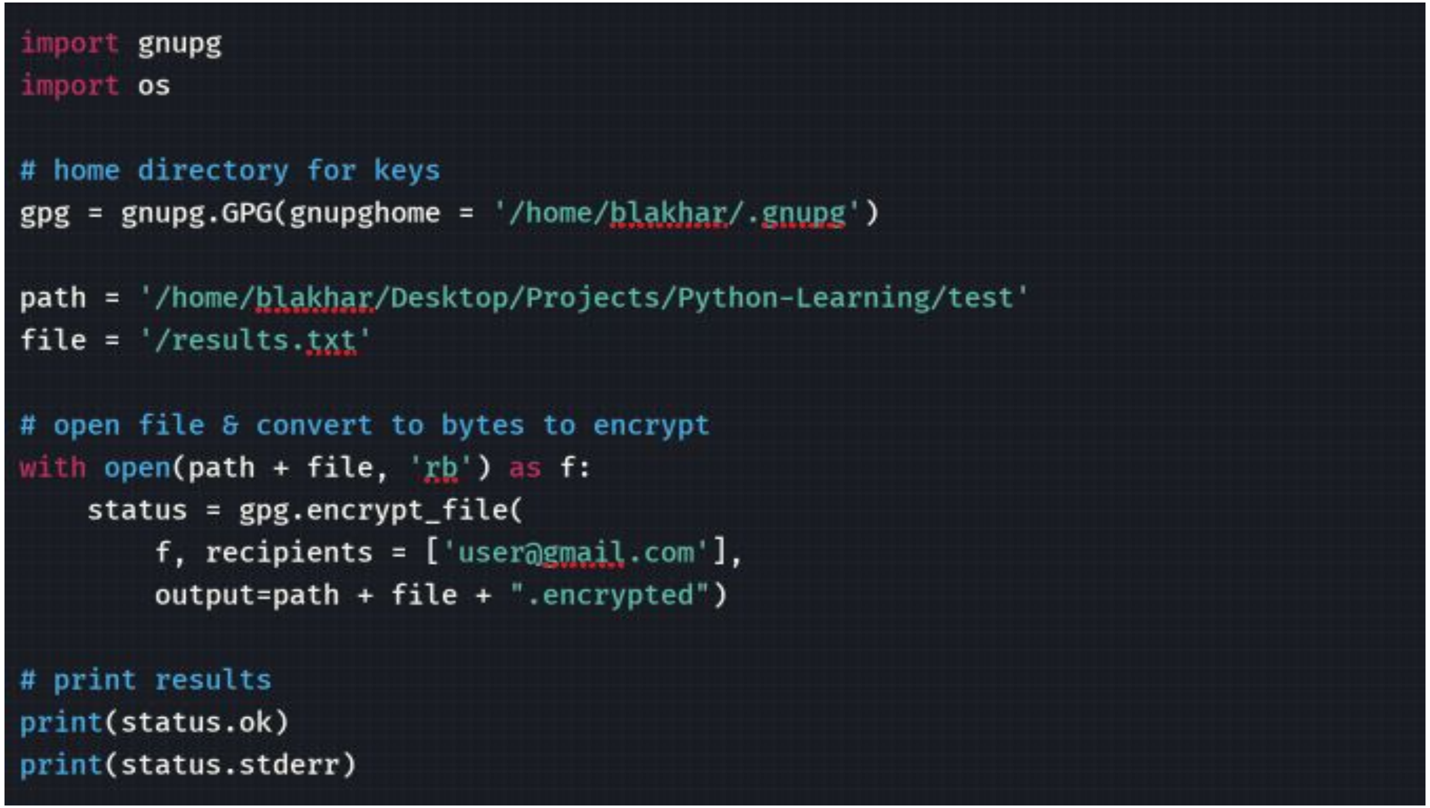
```python
import gnupg
import os

# stored keys path
gpg=gnupg.GPG(gnupghome='/home/blakhar/.gnupg')

# optional - keep it at text
gpg.encoding = 'utf-8'

# imputs to generate keys
input_data = gpg.gen_key_input(
        name_email = 'user@gmail.com',
        passphrase = 'userpassphrase',
        key_type = 'RSA',
        key_length = 1024)

# generate key
key = gpg.gen_key(input_data)

print(key)
```

Import Public Key To Keychain - Set Trust Level



```python
import gnupg

gpg = gnupg.GPG(gnupghome='/home/blakhar/.gnupg')

# read key data
key_data = open('user_pub_key.asc').read()

# gpg import keys method
import_result = gpg.import_keys(key_data)

# set trust level & fingerprint(shorter version of key) for keys to encrypt with
public key
gpg.trust_keys(import_result.fingerprints, 'TRUST_ULTIMATE')

# get list of keys saved
my_keys = gpg.list_keys()

print(my_keys)
```

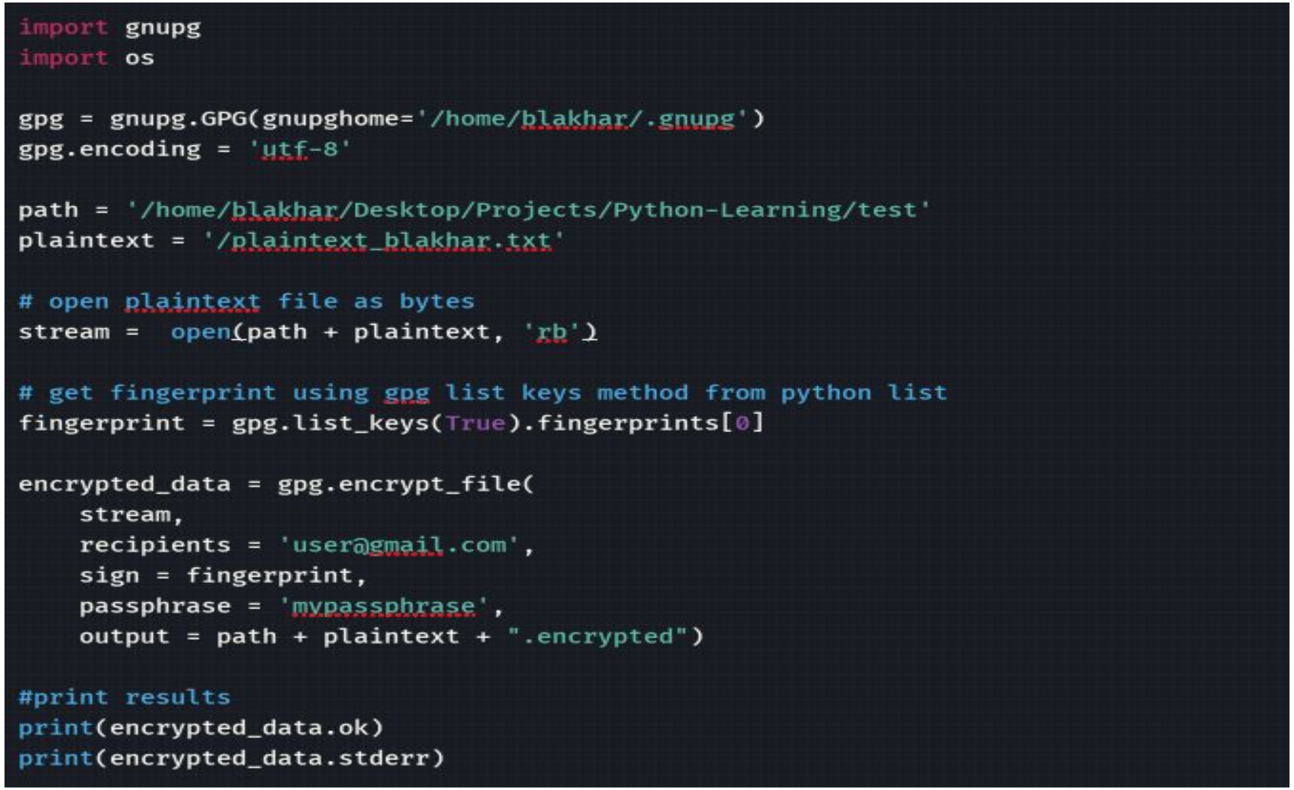
Encrypt File With User - Public Key



```python
import gnupg
import os

# home directory for keys
gpg = gnupg.GPG(gnupghome = '/home/blakhar/.gnupg')

path = '/home/blakhar/Desktop/Projects/Python-Learning/test'
file = '/results.txt'

# open file & convert to bytes to encrypt
with open(path + file, 'rb') as f:
    status = gpg.encrypt_file(
        f, recipients = ['user@gmail.com'],
        output=path + file + ".encrypted")

# print results
print(status.ok)
print(status.stderr)
```

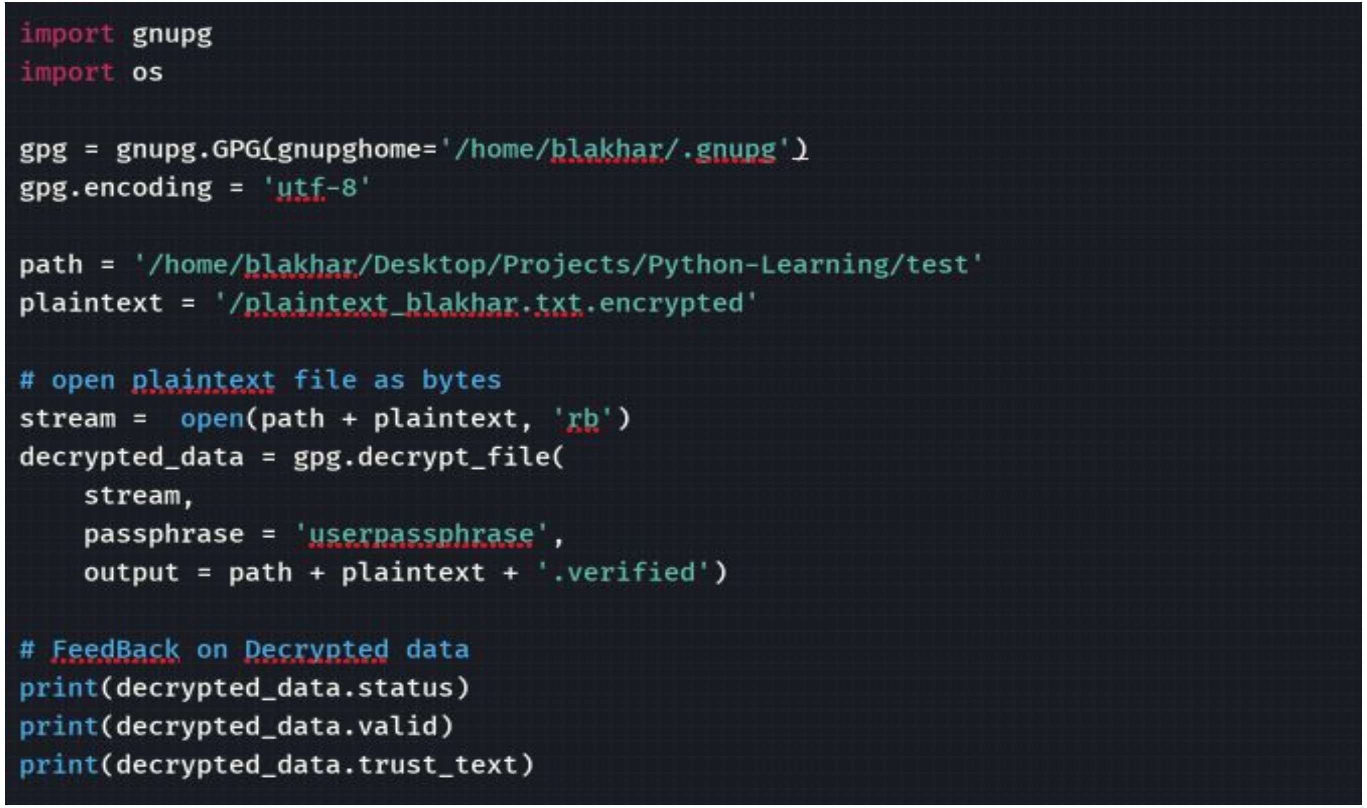
Signature Verification & Authentication

Encrypt with public key & sign with private key



```python
import gnupg
import os

gpg = gnupg.GPG(gnupghome='/home/blakhar/.gnupg')
gpg.encoding = 'utf-8'

path = '/home/blakhar/Desktop/Projects/Python-Learning/test'
plaintext = '/plaintext_blakhar.txt'

# open plaintext file as bytes
stream =  open(path + plaintext, 'rb')

# get fingerprint using gpg list keys method from python list
fingerprint = gpg.list_keys(True).fingerprints[0]

encrypted_data = gpg.encrypt_file(
    stream,
    recipients = 'user@gmail.com',
    sign = fingerprint,
    passphrase = 'mypassphrase',
    output = path + plaintext + ".encrypted")

#print results
print(encrypted_data.ok)
print(encrypted_data.stderr)
```

Decrypt & Verify File



```
import gnupg
import os

gpg = gnupg.GPG(gnupghome='/home/blakhar/.gnupg')
gpg.encoding = 'utf-8'

path = '/home/blakhar/Desktop/Projects/Python-Learning/test'
plaintext = '/plaintext_blakhar.txt.encrypted'

# open plaintext file as bytes
stream =  open(path + plaintext, 'rb')
decrypted_data = gpg.decrypt_file(
    stream,
    passphrase = 'userpassphrase',
    output = path + plaintext + '.verified')

# FeedBack on Decrypted data
print(decrypted_data.status)
print(decrypted_data.valid)
print(decrypted_data.trust_text)
```