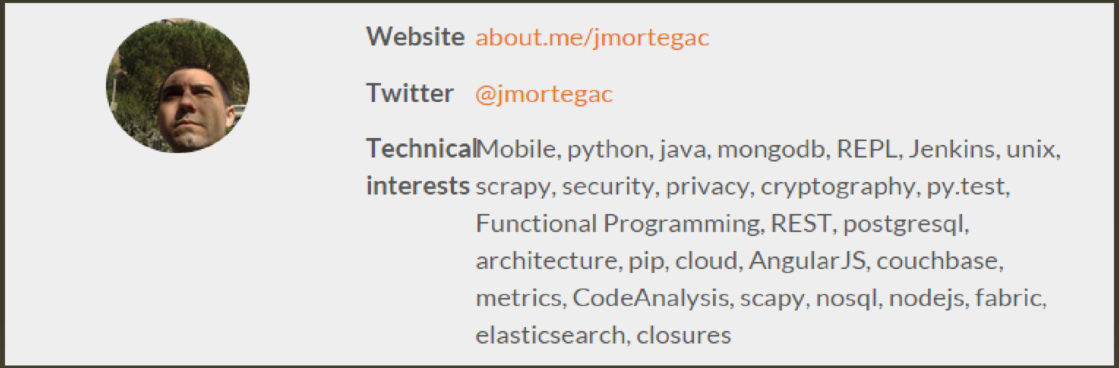
EUROPYTHON 2015 Bilbao, 20-26 July

Python Cryptography & Security

José Manuel Ortega | @jmortegac

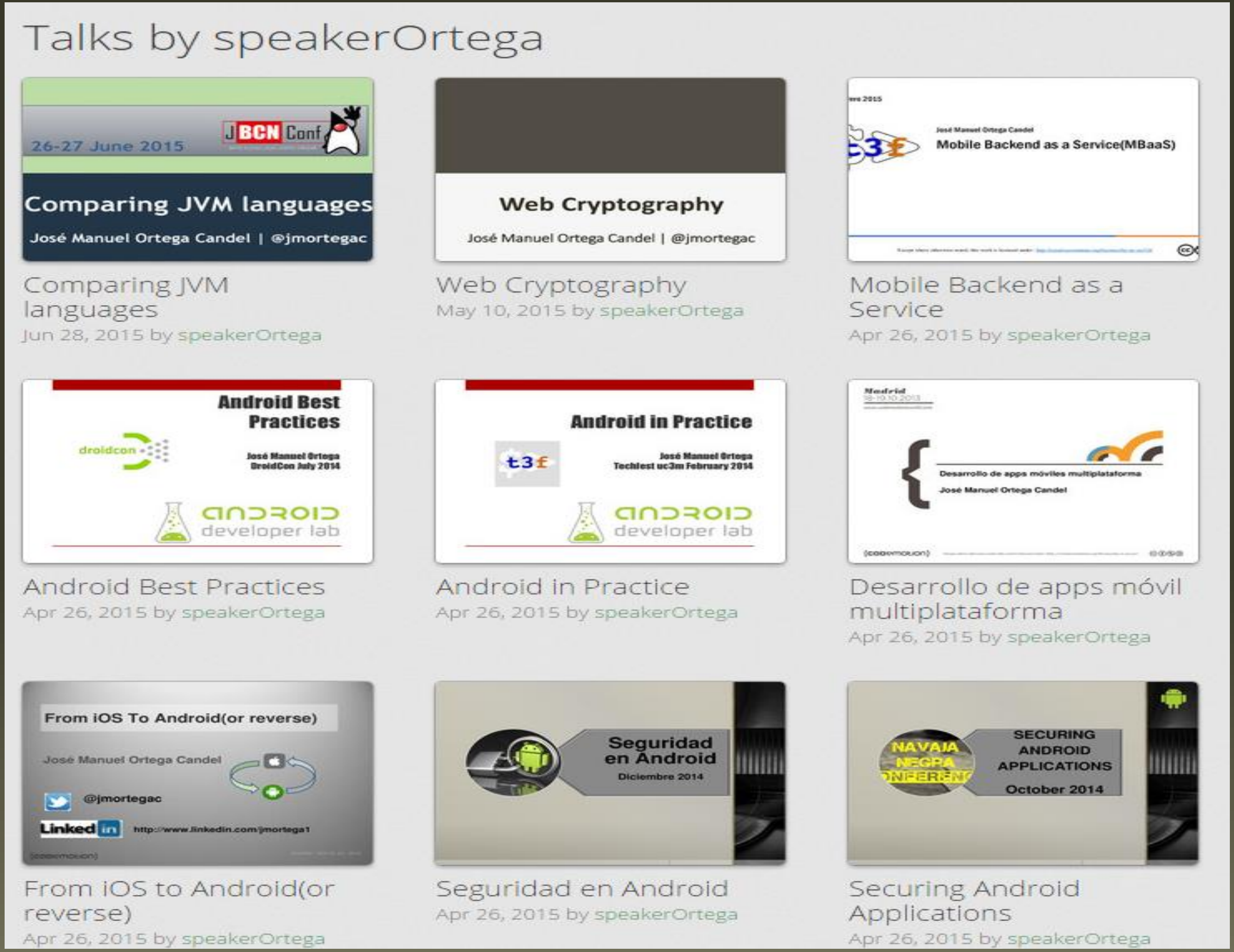
https://speakerdeck.com/jmortega



Website about.me/jmortegac

Twitter @jmortegac

Technical interests Mobile, python, java, mongodb, REPL, Jenkins, unix, scrapy, security, privacy, cryptography, py.test, Functional Programming, REST, postgresql, architecture, pip, cloud, AngularJS, couchbase, metrics, CodeAnalysis, scapy, nosql, nodejs, fabric, elasticsearch, closures



Talks by speakerOrtega

26-27 June 2015 JBCN Conf
Comparing JVM languages
José Manuel Ortega Candel | @jmortegac

Comparing JVM languages
Jun 28, 2015 by speakerOrtega

Web Cryptography
José Manuel Ortega Candel | @jmortegac

Web Cryptography
May 10, 2015 by speakerOrtega

José Manuel Ortega Candel
Mobile Backend as a Service(MBaaS)

Mobile Backend as a Service
Apr 26, 2015 by speakerOrtega

droidcon
Android Best Practices
José Manuel Ortega
DroidCon July 2014
android developer lab

Android Best Practices
Apr 26, 2015 by speakerOrtega

t3f
Android in Practice
José Manuel Ortega
Techfest uc3m February 2014
android developer lab

Android in Practice
Apr 26, 2015 by speakerOrtega

Madrid
Desarrollo de apps móviles multiplataforma
José Manuel Ortega Candel
{codemotion}

Desarrollo de apps móvil multiplataforma
Apr 26, 2015 by speakerOrtega

From iOS To Android(or reverse)
José Manuel Ortega Candel
@jmortegac
LinkedIn http://www.linkedin.com/jmortega1
{codemotion}

From iOS to Android(or reverse)
Apr 26, 2015 by speakerOrtega

Seguridad en Android
Diciembre 2014

Seguridad en Android
Apr 26, 2015 by speakerOrtega

NAVAJA NEGRA CONFERENCE
SECURING ANDROID APPLICATIONS
October 2014

Securing Android Applications
Apr 26, 2015 by speakerOrtega

Security Conferences

NAVAJA NEGRA CONFERENCE

Я был здесь и вернулся!

#nn4ed

- ALBACETE -

/Rooted CON

NcN No cON Name

SICKMINDS 2015

information tech conferences

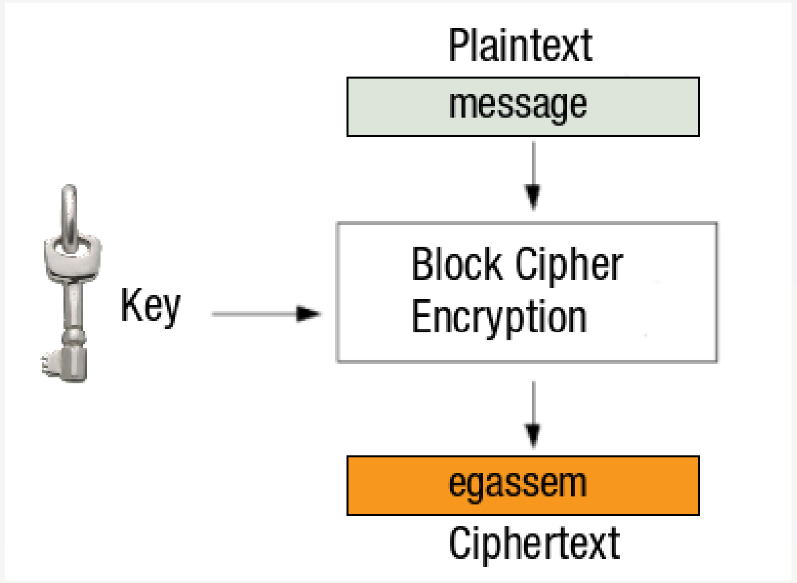
NOV 18-19-20-21, 2015

INDEX

Introduction to cryptography

- Key terms
- Caesar Chiper
- Hash functions(MD5,SHA)
- Symetric Encryption(AES)
- Asimetric Encription(RSA)
- PBKDF2-Key derivation function

Key terms



- **Key**: The piece of information that allows you to either encrypt or decrypt your data.
- **Plaintext**: The information that you want to keep hidden, in its unencrypted form. The plaintext can be any data at all: a picture, a spreadsheet, or even a whole hard disk
- **Ciphertext**: The information in encrypted form
- **Cipher**: The algorithm that converts plaintext to ciphertext and vice-versa

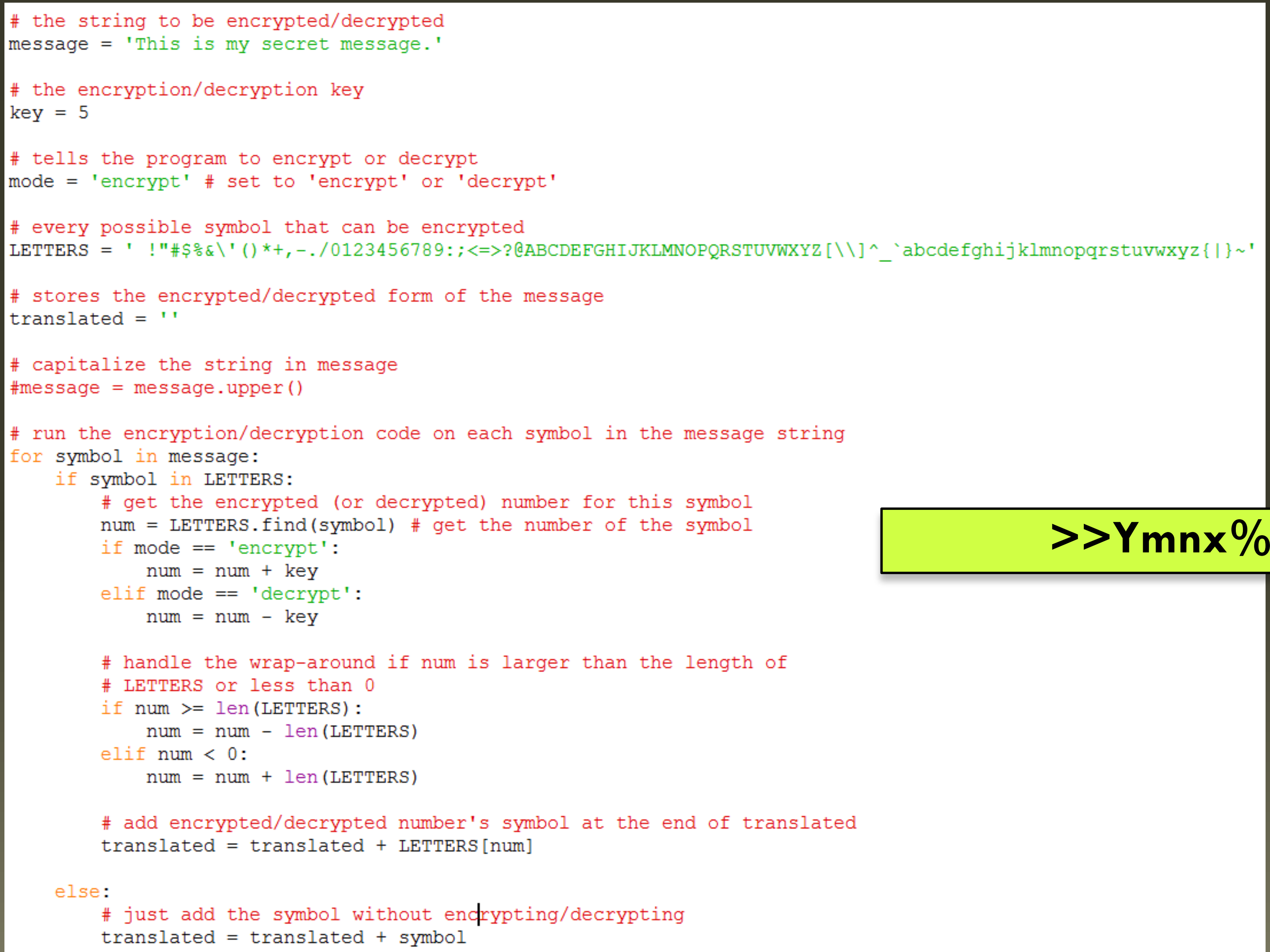
Key terms advanced

Salt – randomizes the hash of the key; prevents rainbow table attacks against the key

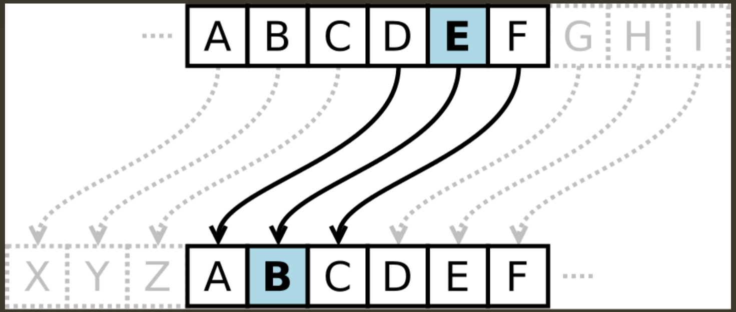
IV (initialization vector) – randomizes the encrypted message; prevents rainbow table attacks against the message

Derived Key – lengthens and strengthens the key via hashing; used instead of the original key; slows down brute-force attacks against the key

Caesar Chiper



```
# the string to be encrypted/decrypted
message = 'This is my secret message.'

# the encryption/decryption key
key = 5

# tells the program to encrypt or decrypt
mode = 'encrypt' # set to 'encrypt' or 'decrypt'

# every possible symbol that can be encrypted
LETTERS = ' !"#$%&\'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`abcdefghijklmnopqrstuvwxyz{|}~'

# stores the encrypted/decrypted form of the message
translated = ''

# capitalize the string in message
#message = message.upper()

# run the encryption/decryption code on each symbol in the message string
for symbol in message:
    if symbol in LETTERS:
        # get the encrypted (or decrypted) number for this symbol
        num = LETTERS.find(symbol) # get the number of the symbol
        if mode == 'encrypt':
            num = num + key
        elif mode == 'decrypt':
            num = num - key

        # handle the wrap-around if num is larger than the length of
        # LETTERS or less than 0
        if num >= len(LETTERS):
            num = num - len(LETTERS)
        elif num < 0:
            num = num + len(LETTERS)

        # add encrypted/decrypted number's symbol at the end of translated
        translated = translated + LETTERS[num]

    else:
        # just add the symbol without encrypting/decrypting
        translated = translated + symbol
```

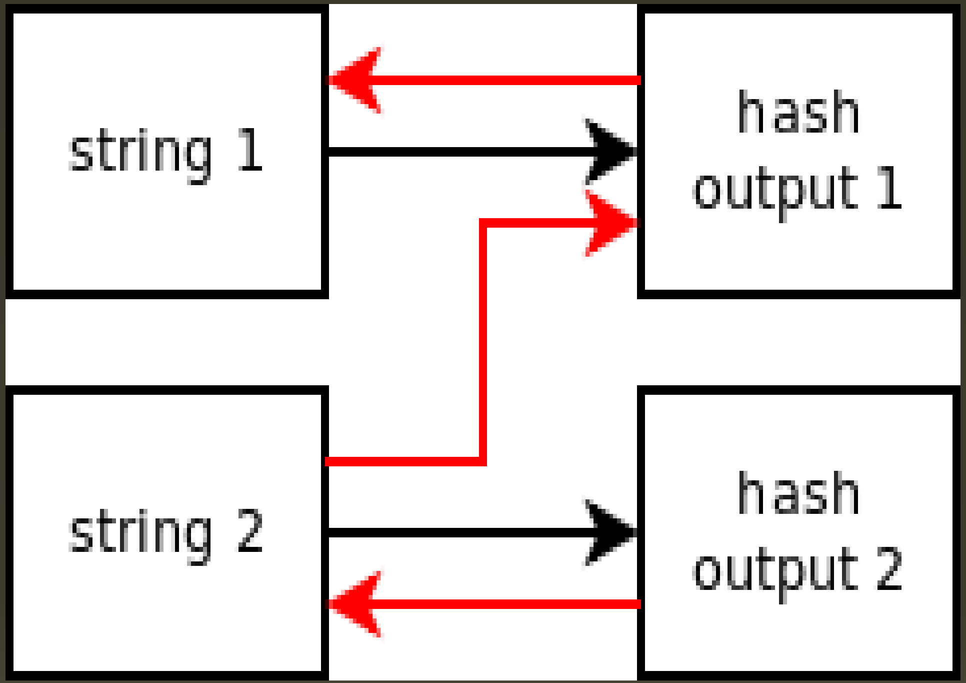


>>Ymnx%nx%r~%xjhwjy%rjxxflj3

Hash functions

- Calculate the checksum of some data
- File integrity checking
- Generate passwords
- Digital signatures and authentication
- MD5
- SHA-2(256 and 512 bits)
- SHA-3

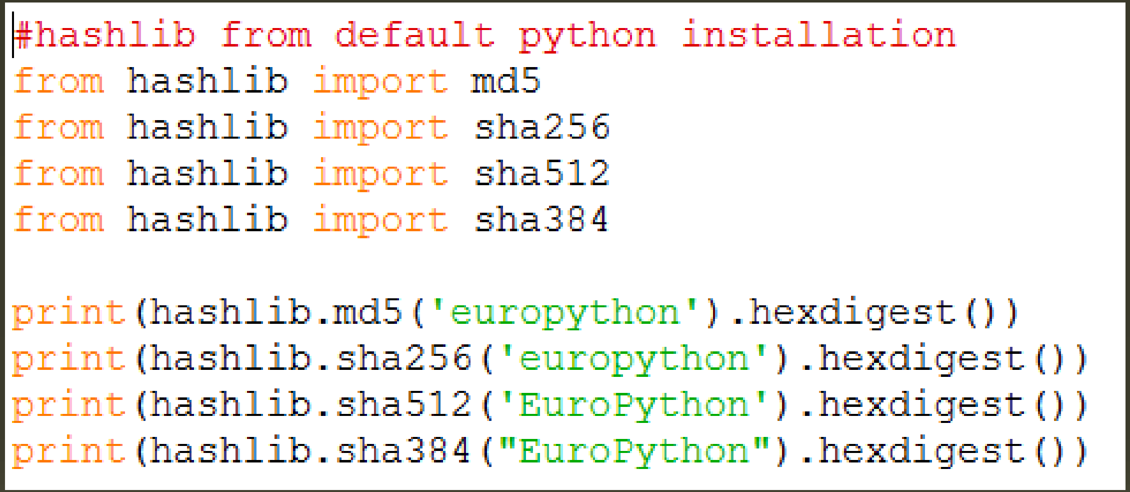
Hash functions



https://docs.python.org/2/library/hashlib.html

Hashlib functions

- One-way cryptographic hashing

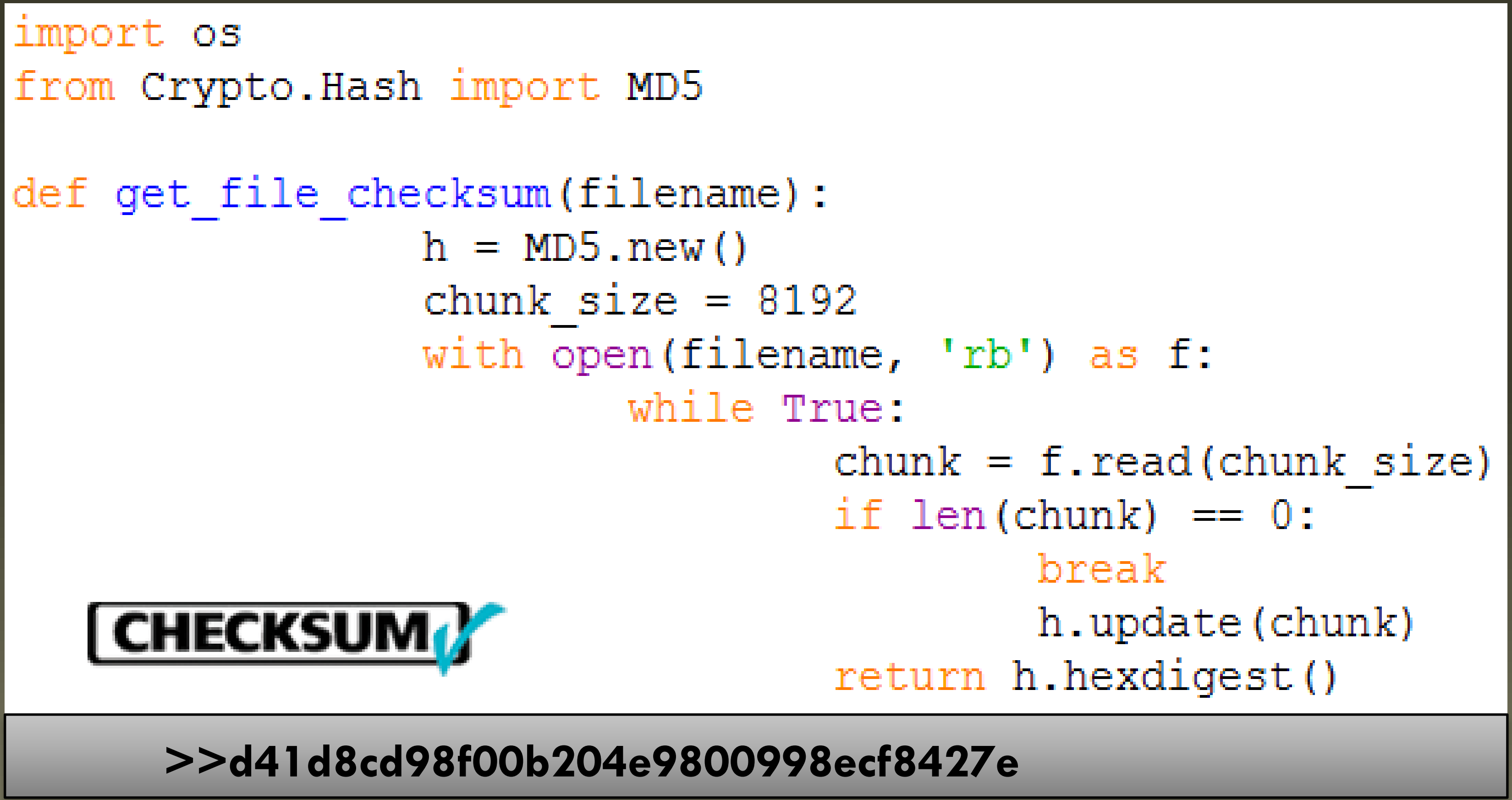


```
#hashlib from default python installation
from hashlib import md5
from hashlib import sha256
from hashlib import sha512
from hashlib import sha384

print(hashlib.md5('europython').hexdigest())
print(hashlib.sha256('europython').hexdigest())
print(hashlib.sha512('EuroPython').hexdigest())
print(hashlib.sha384("EuroPython").hexdigest())
```

```
>>03187564433616a654efef944871f1e4
>>bd576c4231b95dd439abd486be45e23d47a2cbb74b5348b3b113cef47463e15a
>>d47b290aa260af8871294e1ad6b473bd48b587593f8dea7b1b5d9271df12ee081
85a13217ae88e95d9bd425f3ada0593f1671004a2b32380039d3c88f685614c
>>8fadab23df7c580915deba5c6f0eb75bd32181f55c547a2b3999db055398095c33f
10b75c823a288e86636797f71b458
```

MD5 hash function

- Checking file integrity



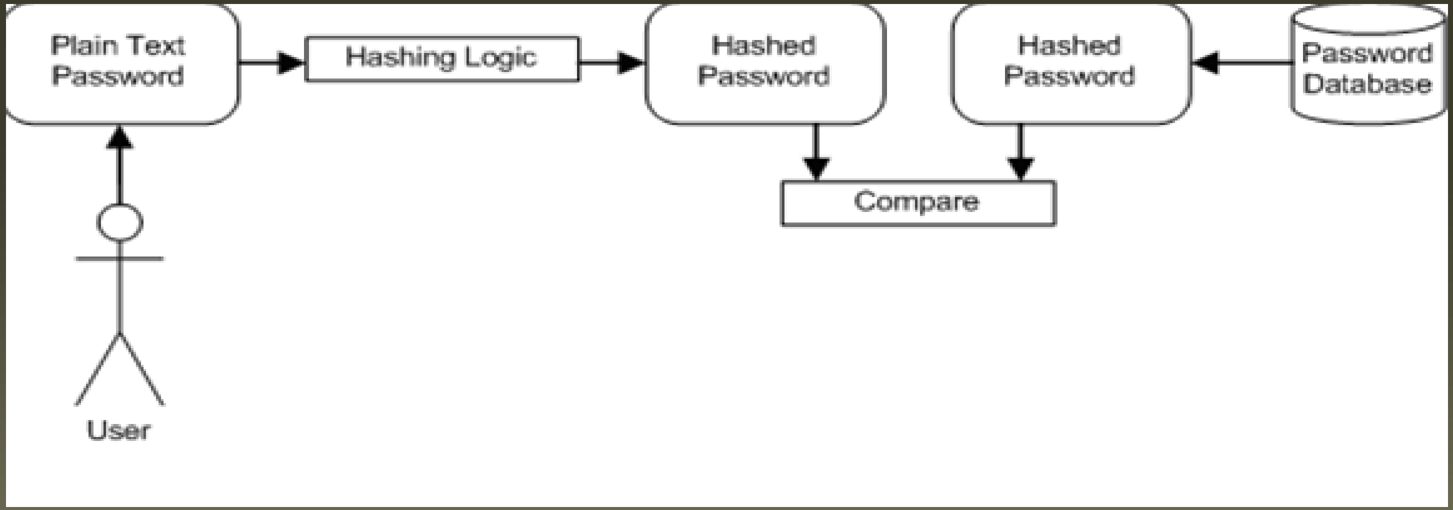
```python
import os
from Crypto.Hash import MD5

def get_file_checksum(filename):
                h = MD5.new()
                chunk_size = 8192
                with open(filename, 'rb') as f:
                        while True:
                                chunk = f.read(chunk_size)
                                if len(chunk) == 0:
                                        break
                                        h.update(chunk)
                                return h.hexdigest()
```

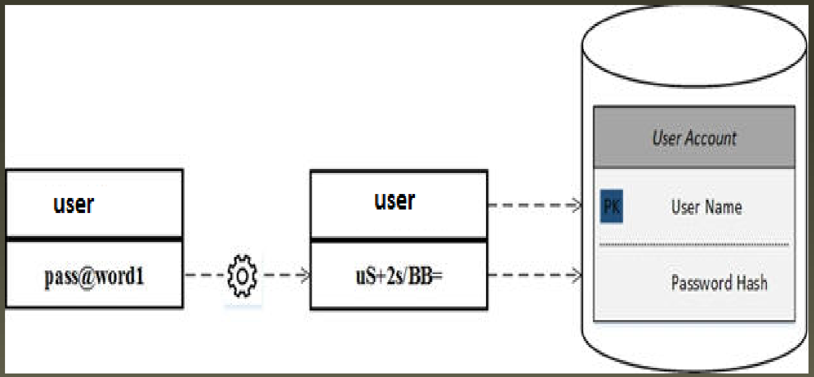
```
>>d41d8cd98f00b204e9800998ecf8427e
```

Hash passwords in DB

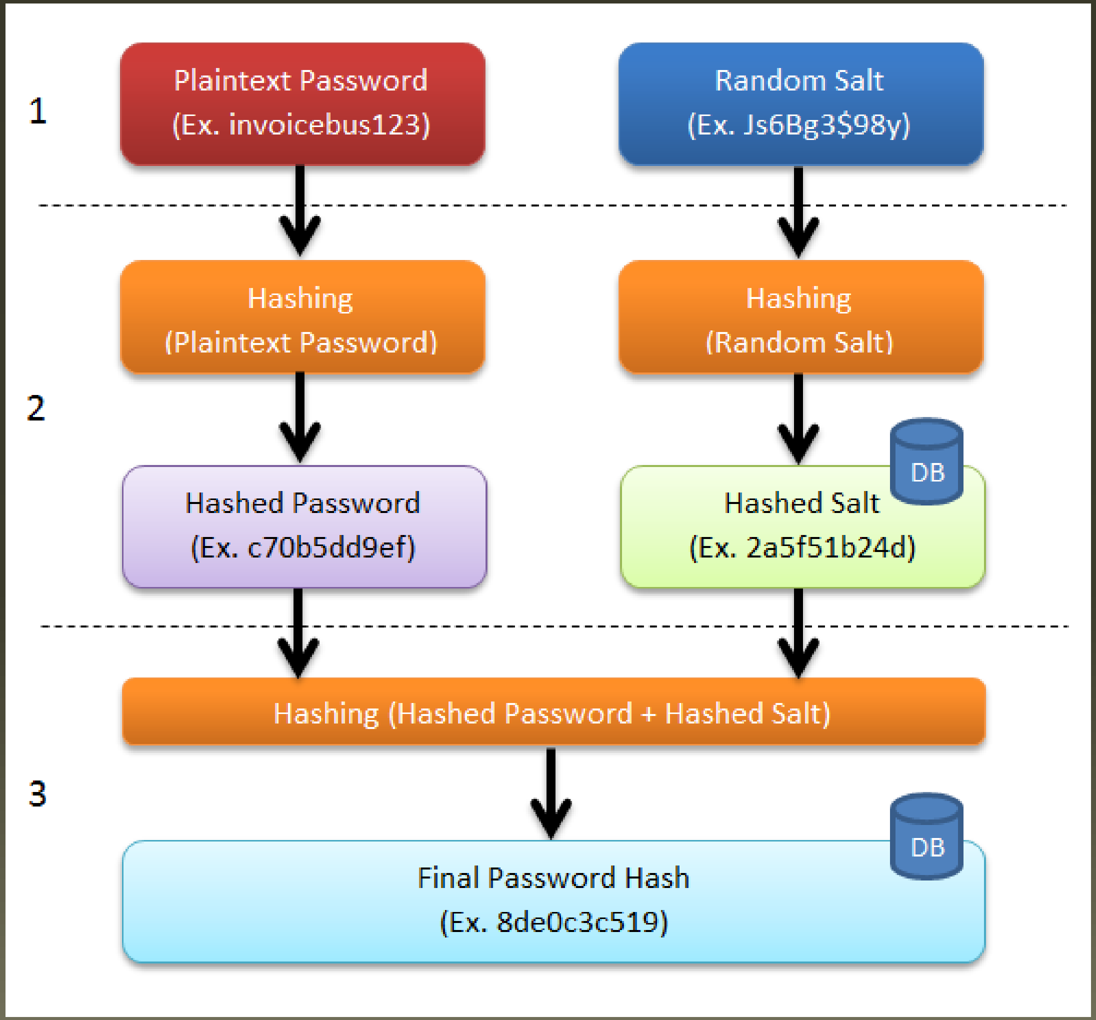
- Websites store hash of a password

```
hashlib.sha256('password').hexdigest()
>>'5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8'
```





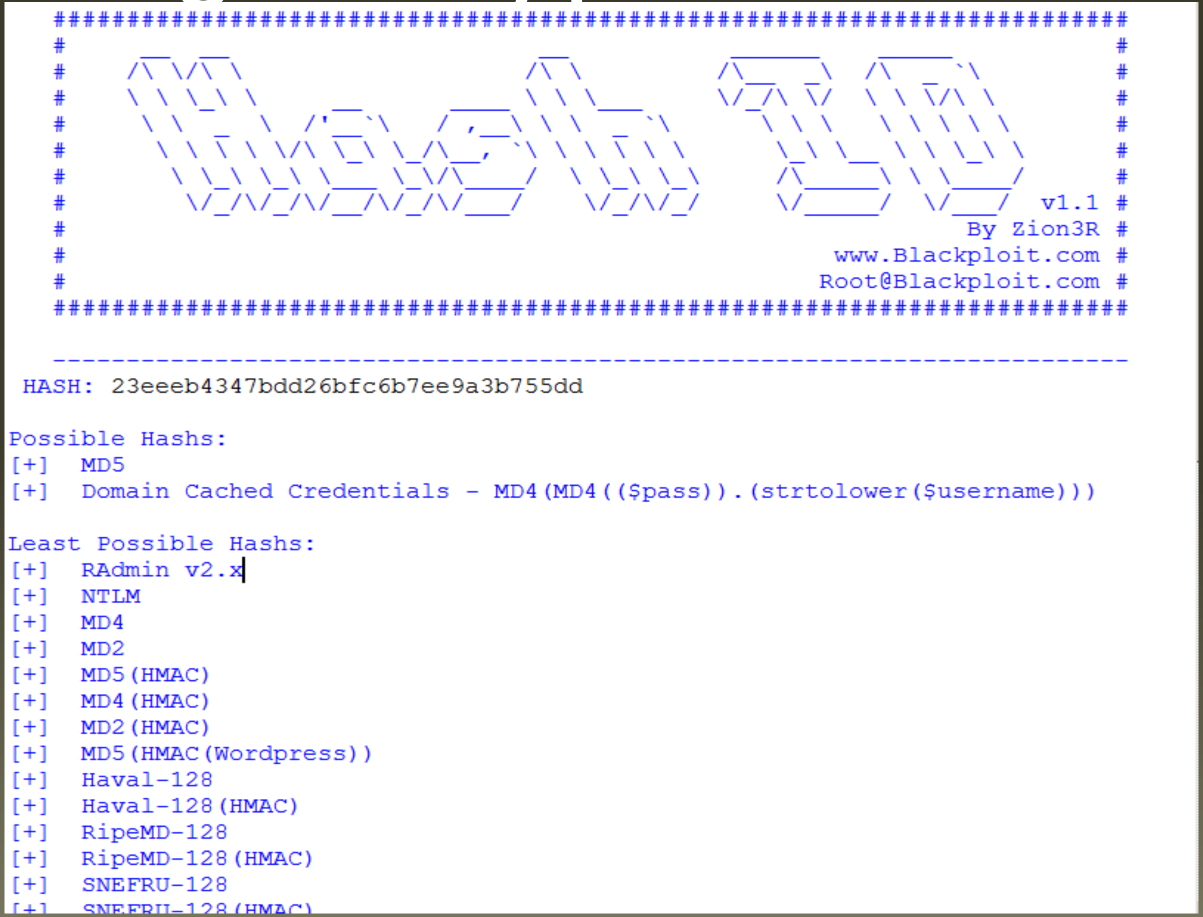
Hash passwords in DB



Hash identifier

https://code.google.com/p/hash-identifier

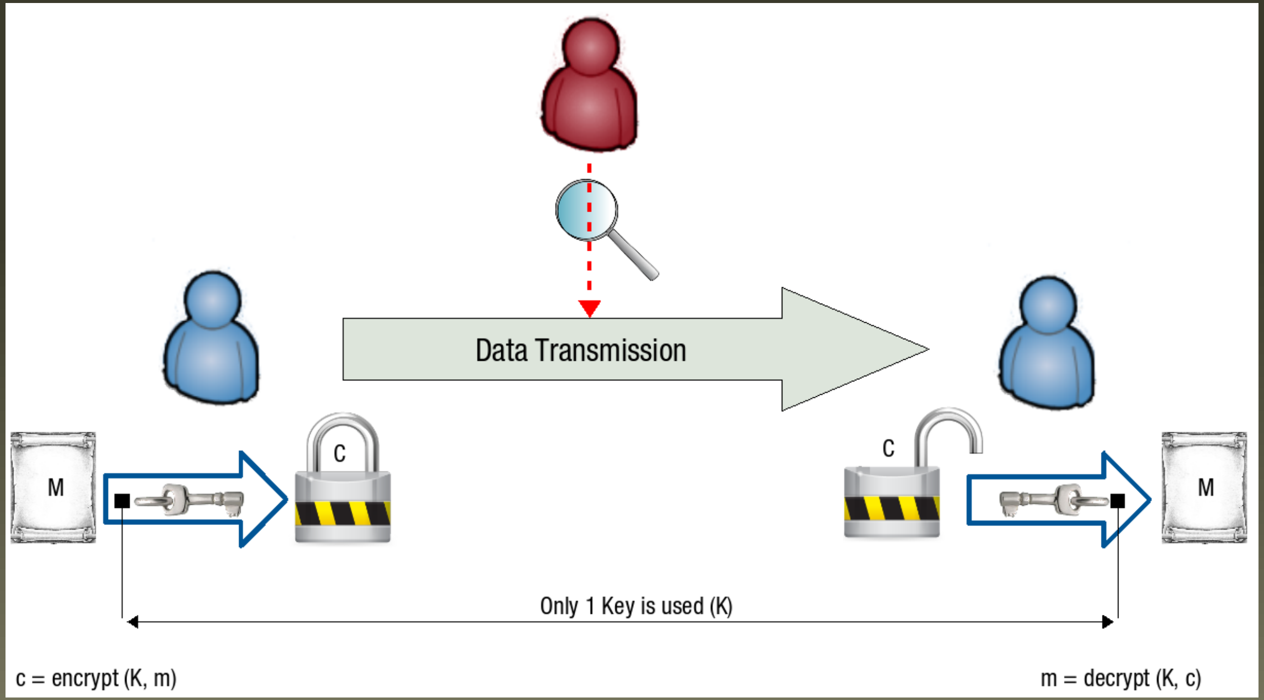
- For checking the type of Hash



```
   #########################################################################
   #     __  __                     __           ______    _____           #
   #    /\ \/\ \                   /\ \         /\__  _\  /\  _ `\         #
   #    \ \ \_\ \     __      ____ \ \ \___     \/_/\ \/  \ \ \/\ \        #
   #     \ \  _  \  /'__`\   / ,__\ \ \  _ `\      \ \ \   \ \ \ \ \       #
   #      \ \ \ \ \/\ \_\ \_/\__, `\ \ \ \ \ \      \_\ \__ \ \ \_\ \      #
   #       \ \_\ \_\ \___ \_\/\____/  \ \_\ \_\     /\_____\ \ \____/      #
   #        \/_/\/_/\/__/\/_/\/___/    \/_/\/_/     \/_____/  \/___/  v1.1 #
   #                                                             By Zion3R #
   #                                                    www.Blackploit.com #
   #                                                   Root@Blackploit.com #
   #########################################################################

   -------------------------------------------------------------------------
 HASH: 23eeeb4347bdd26bfc6b7ee9a3b755dd

Possible Hashs:
[+]  MD5
[+]  Domain Cached Credentials - MD4(MD4(($pass)).(strtolower($username)))

Least Possible Hashs:
[+]  RAdmin v2.x
[+]  NTLM
[+]  MD4
[+]  MD2
[+]  MD5(HMAC)
[+]  MD4(HMAC)
[+]  MD2(HMAC)
[+]  MD5(HMAC(Wordpress))
[+]  Haval-128
[+]  Haval-128(HMAC)
[+]  RipeMD-128
[+]  RipeMD-128(HMAC)
[+]  SNEFRU-128
[+]  SNEFRU-128(HMAC)
```

Symetric encryption

- AES
- Shared key for encrypt and decrypt

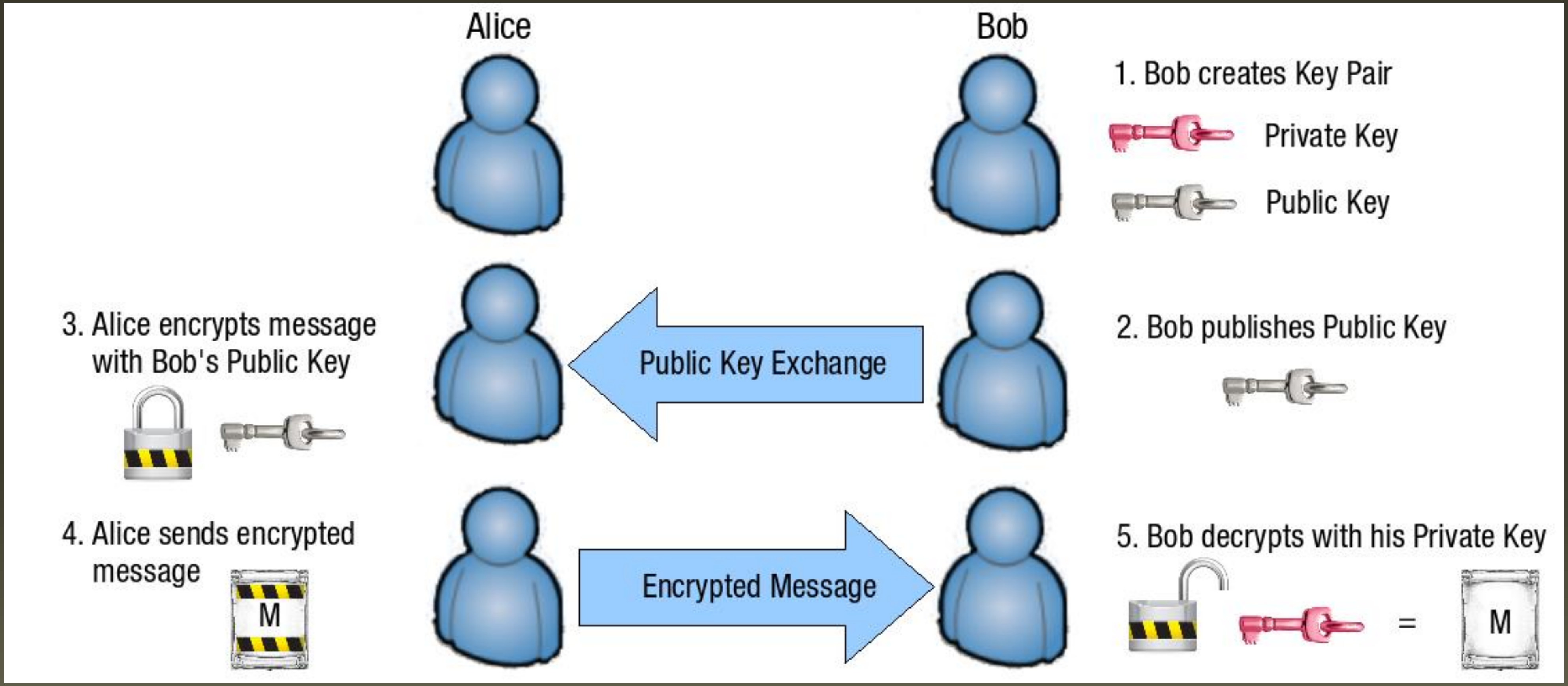


cipher	key size (bytes in ASCII)
AES-128	128 bits (16 bytes)
AES-192	192 bits (24 bytes)
AES-256	256 bits (32 bytes)

Asymetric encryption

- RSA
- 2 keys(public key and secret key)
- Public key(Pk) for encrypt
- Secret key(Sk) for decrypt
- Public key is derived from secret key

Asymetric encryption

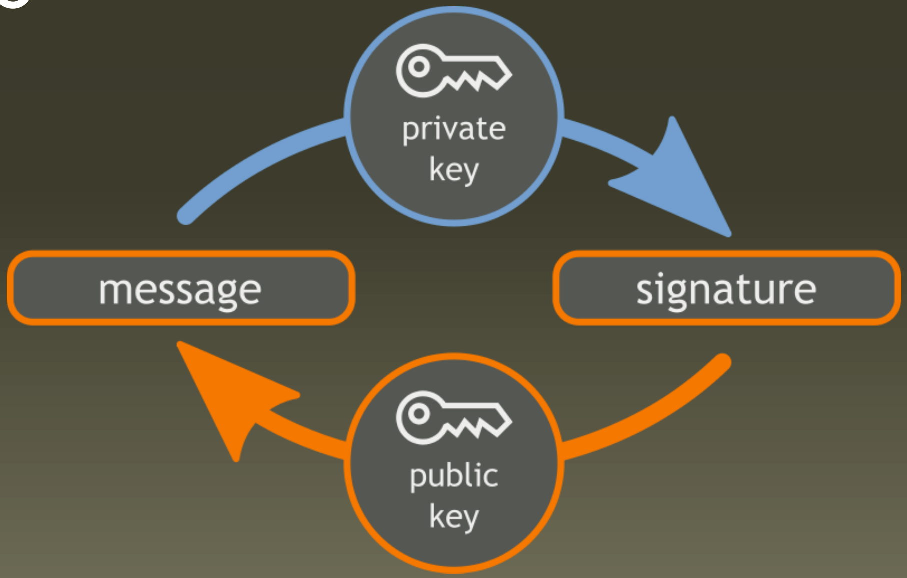

Alice
Bob
1. Bob creates Key Pair
Private Key
Public Key
3. Alice encrypts message with Bob's Public Key
Public Key Exchange
2. Bob publishes Public Key
4. Alice sends encrypted message
M
Encrypted Message
5. Bob decrypts with his Private Key
=
M

Encryption vs Signing

- Encryption→When encrypting, you use their public key to write message and they use their private key to read it.
- Signing→When signing, you use your private key to write message's signature, and they use your public key to check if it's really yours.

Digital signature

- Signing a message
- Only the owner of Pk/Sk pair should be able to sign the message



PyCrypto

https://pypi.python.org/pypi/pycrypto

- Supports Hash operations
- Block cipher AES,RSA
- Sign/verify documents

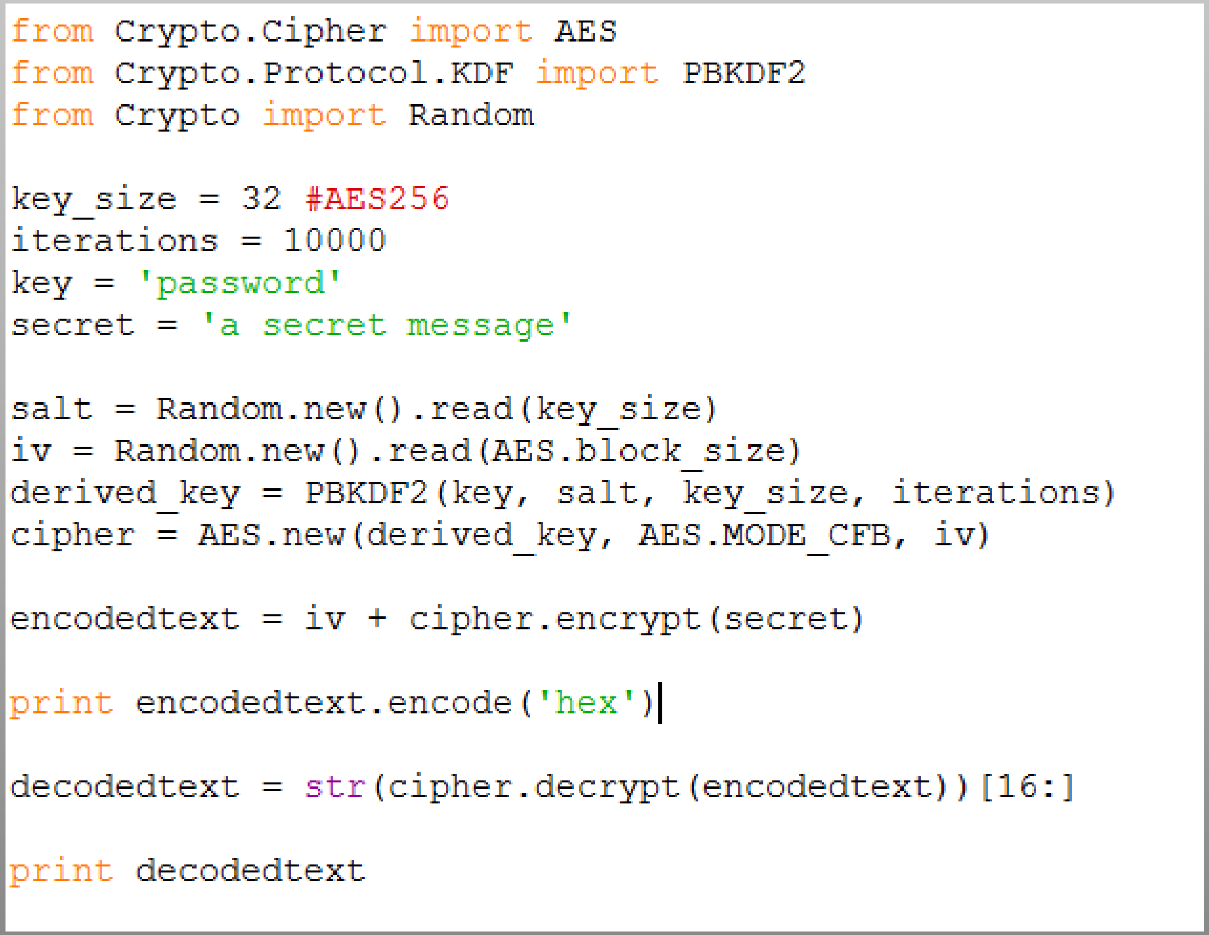
```
>> pip install pycrypto
```

PyCrypto Hash functions

```
from Crypto.Hash import SHA256
SHA256.new(‘password').hexdigest()
>>'5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8'
```

```
from Crypto.Hash import SHA512
SHA512.new(‘password').hexdigest()
>>'b109f3bbbc244eb82441917ed06d618b9008dd09b3befd1b5e07394c706a8bb98
0b1d7785e5976ec049b46df5f1326af5a2ea6d103fd07c95385ffab0cacbc86'
```

PyCrypto AES



```python
from Crypto.Cipher import AES
from Crypto.Protocol.KDF import PBKDF2
from Crypto import Random

key_size = 32 #AES256
iterations = 10000
key = 'password'
secret = 'a secret message'

salt = Random.new().read(key_size)
iv = Random.new().read(AES.block_size)
derived_key = PBKDF2(key, salt, key_size, iterations)
cipher = AES.new(derived_key, AES.MODE_CFB, iv)

encodedtext = iv + cipher.encrypt(secret)

print encodedtext.encode('hex')

decodedtext = str(cipher.decrypt(encodedtext))[16:]

print decodedtext
```

```
>>>
d1a2ea7f9661fae8b46b3904b0193ab81516653f73216dfeb5f51afde3d405b2
a secret message
```

PyCrypto PBKDF

Generating key from password

```
import Crypto.Random
from Crypto.Protocol.KDF import PBKDF2

password = 'europython'
iterations = 5000
key = ''
salt = Crypto.Random.new().read(32)

key = PBKDF2(password, salt, dkLen=32, count=iterations)

print 'Random salt (in hex):'
print salt.encode('hex')
print 'PBKDF2-derived key (in hex) of password after %d iterations: ' % iterations
print key.encode('hex')
```

A salt is a random sequence added to the password string before using the hash function. The salt is used in order to prevent dictionary attacks and rainbow tables attacks.

```
Random salt (in hex):
724138b9d987a04bf05d285db678824f9b7e2b1232229711c2e0e2e556a0c19a
PBKDF2-derived key (in hex) of password after 5000 iterations:
d725de7de88e27d16c9c4f224d4c871597357084119d1c949074962b48ce26900
```

PyCrypto RSA

Generate an RSA secret and public key pair

```
from Crypto.PublicKey import RSA


def generate_RSA(bits=1024):
#Generate an RSA keypair with an exponent of 65537 in PEM format
#param: bits The key length in bits
#Return secret key and public key
new_key = RSA.generate(bits, e=65537)
public_key = new_key.publickey().exportKey("PEM")
secret_key = new_key.exportKey("PEM")
return secret_key, public_key
```

PyCrypto RSA

Generate an RSA secret and public key pair

```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCYS9ITbjKu5i9i36FgzKg/HO3o
6CKGJ1c5E57qVlmYF6L1BcgH+eE+XiwJ6fWyShaVnZDuvUapWgQeOGZ60QBJ/vpu
DdwqsuGoTeJNqaRT9ButJa+o+0tchRKBcM6zKUXYWc7kdAlxEpO2OXZEqxD7bd1O
oxv7mEjqBpVXgNEVrwIDAQAB
-----END PUBLIC KEY-----
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQCYS9ITbjKu5i9i36FgzKg/HO3o6CKGJ1c5E57qVlmYF6L1BcgH
+eE+XiwJ6fWyShaVnZDuvUapWgQeOGZ60QBJ/vpuDdwqsuGoTeJNqaRT9ButJa+o
+0tchRKBcM6zKUXYWc7kdAlxEpO2OXZEqxD7bd1Ooxv7mEjqBpVXgNEVrwIDAQAB
AoGAc0qqzTTWP5tYciRTmeE02RqAbJoXULHFkRruaf5WsxHptk3blVakkr9d3V91
NbRqpnby+hjlvly701jlE8LW0QIccll9oWyV6kMSTEJMth9RlXpCbQY285pwg+bF
zyEhQJmjMj1hMDJLQ8dXLCeqXZ37etYGHTT2XQ+q5TOW4YkCQQC5WDQHBhYa/Mzt
UlXemLxv1ERaxt8zmXSX0bKjlkaYMv1SF3FskiN9Rm/zXvil3HuiySBq9g6/fPbN
T1+dtiZTAkEA0lpsRUqamlbii18aBBQGs/FbrUa71ahpoU7+8wXMxNYQBfVGvlzs
J+tKxSecMO196Hl4l5I14ASEs+4wKK5vtQJARe4gmzHRr1clntY87eKk3nCxZaq5
Vkek9Q86nlB1YEGE0K9lrTgqSb8EyEdh+3qH73CBWboC8H7ew7IZ+nBaXwJBAJEO
K8Vomcz+jvB/B0iyqqChmo+VzGecuCK1f9gEMt21o90H893H5E3u0mO8WdffnciX
I1KaT66lTx5o7SrQh1UCQGqP8B9bpzXjxMuLUJuL1DoRP4QBGHoXokdu8gKAlPzp
ZK8BKRSPRobwlNFlXWfXLAWIFwXIeqObII20U/oNwNE=
-----END RSA PRIVATE KEY-----
```

PyCrypto RSA

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP

def encrypt_RSA(public_key, message):
		key = (public_key, "r").read()
		rsakey = RSA.importKey(key)
		rsakey = PKCS1_OAEP.new(rsakey)
		encrypted = rsakey.encrypt(message)
		return encrypted.encode('base64')
```

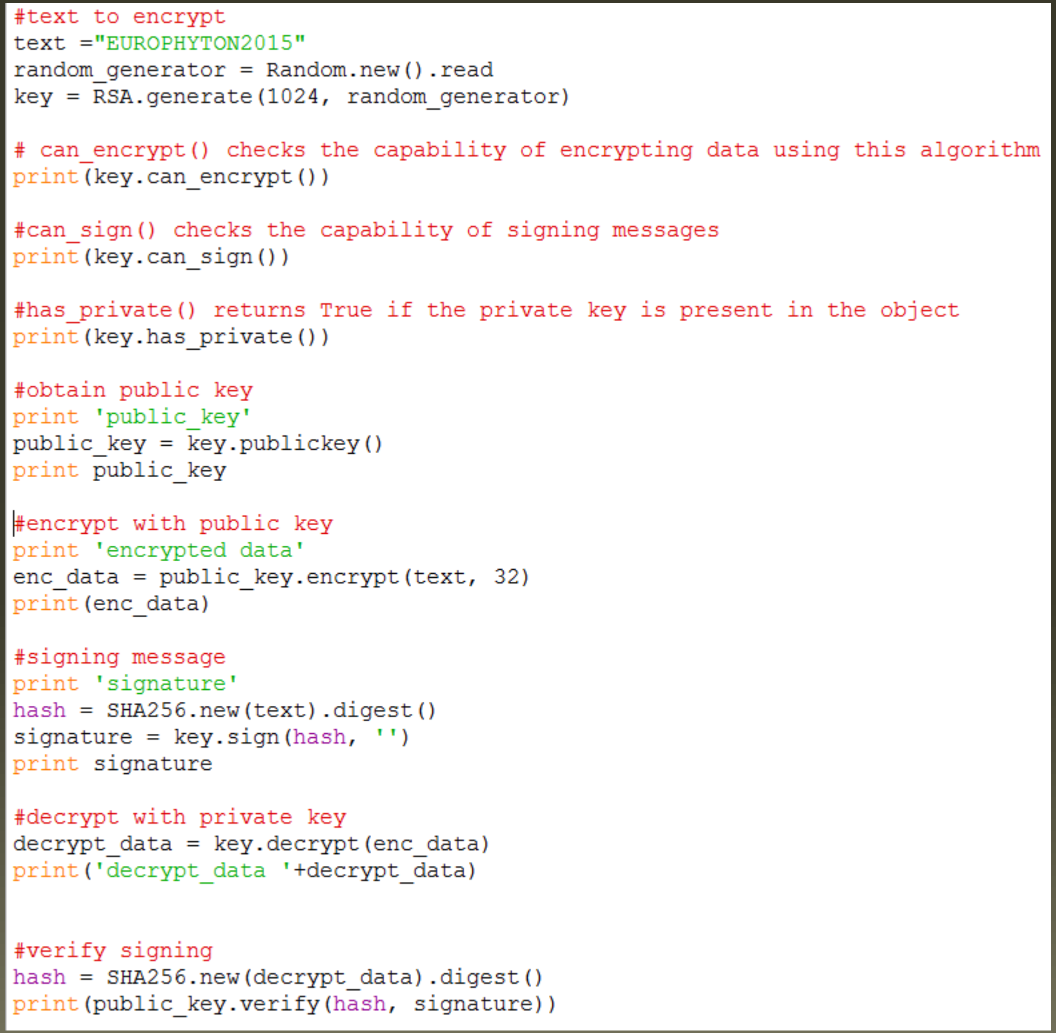
```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
from base64 import b64decode

def decrypt_RSA(secret_key, message):
		key = (secret_key, "r").read()
		rsakey = RSA.importKey(key)
		rsakey = PKCS1_OAEP.new(rsakey)
		decrypted =
rsakey.decrypt(b64decode(message))
		return decrypted
```

PyCrypto Sign/verify

```python
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
from Crypto.Hash import SHA256
from base64 import b64encode, b64decode

def sign_data(secret_key, data):
                key = (secret_key, "r").read()
                rsakey = RSA.importKey(key)
                signer = PKCS1_v1_5.new(rsakey)
                digest = SHA256.new()
                digest.update(b64decode(data))
                sign = signer.sign(digest)
                return b64encode(sign)
```

```python
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
from Crypto.Hash import SHA256
from base64 import b64decode

def verify_sign(public_key, signature, data):    ‘
#Verifies with a public key that the data was signed by their
#private key
pub_key = (public_key, "r").read()
rsakey = RSA.importKey(pub_key)
signer = PKCS1_v1_5.new(rsakey)
digest = SHA256.new()
digest.update(b64decode(data))
if signer.verify(digest, b64decode(signature)):
                return True
return False
```

PyCrypto RSA/Sign/verify



```python
#text to encrypt
text ="EUROPHYTON2015"
random_generator = Random.new().read
key = RSA.generate(1024, random_generator)

# can_encrypt() checks the capability of encrypting data using this algorithm
print(key.can_encrypt())

#can_sign() checks the capability of signing messages
print(key.can_sign())

#has_private() returns True if the private key is present in the object
print(key.has_private())

#obtain public key
print 'public_key'
public_key = key.publickey()
print public_key

#encrypt with public key
print 'encrypted data'
enc_data = public_key.encrypt(text, 32)
print(enc_data)

#signing message
print 'signature'
hash = SHA256.new(text).digest()
signature = key.sign(hash, '')
print signature

#decrypt with private key
decrypt_data = key.decrypt(enc_data)
print('decrypt_data '+decrypt_data)


#verify signing
hash = SHA256.new(decrypt_data).digest()
print(public_key.verify(hash, signature))
```

```
True
True
True
public_key
<_RSAobj @0x2b56648 n(1024),e>
encrypted data
('I\xe6\xff\\
M$\x12\xbb\x95\xee\x02\xcf\x82Im\tf+\x1f\xaeU\xbdv`
^\x94\xfa\xe6_\x8b\xed\x8d\xa3\xab\xfc
\xae\x17\x07=|\x18\xca\x18j\xc5\x1d\x01\xad`\xd6W
E\xfbU\xd1\x12\x0c-
\xb6\x9c\xc4\x07\xaa\x93<\xb5zw&\x98\xa2\xdc\x8e\
x9e-
\x06gQ\xcf\xfa\xc8r/\xd5\x98|\xd5\xcdg\xb2\xda\xcd:
d\xaf\xde\xe2\xcd\xcd\xf5{p`\x07\xbb~\x1b\xa4hHJ#c\
tE6\xfa\xc3\x87\x8d\xf2O8,\xe2W',)
signature
(445755122549853282247622461459180943435051515
5918916324891286777775175591376873419505852842
3900156177220742858645089371096255086061177099
8101038368420840785203067622854793789417670298
3088451295738677105320376959152029164761636442
8930467543317371804318093617486393498897888949
152557196686676342045445446511829L,)
decrypt_data EUROPHYTON2015
True
```

Best practices

- Avoid hashing methods like MD5 or SHA-1,use at least SHA-2 or SHA-3
- Key Stretching for strong passwords
- Preventing Brute-force or dictionary attacks

```
for i in xrange(iterations):
        m = hashlib.sha512()
        m.update(key + password + salt)
        key = m.digest()
```

Cryptography

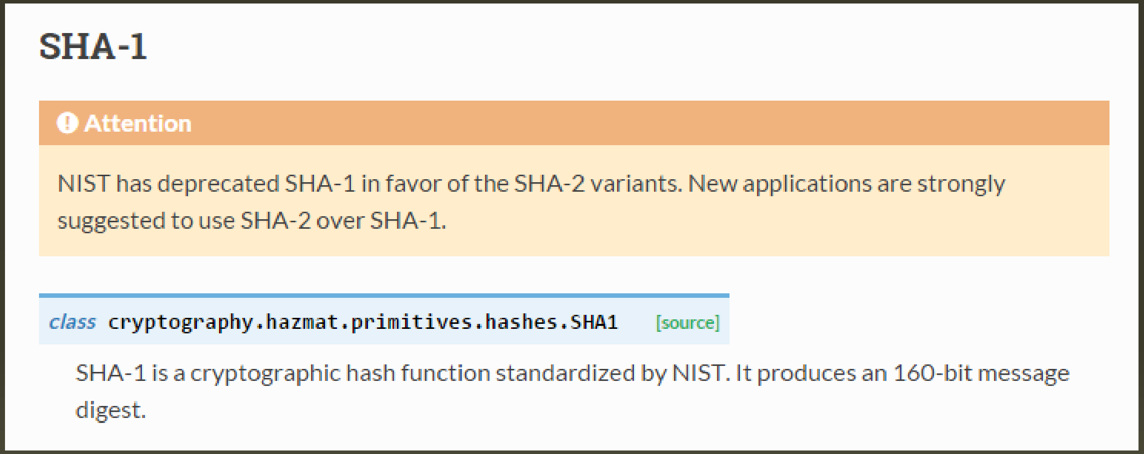
https://cryptography.io

```
$ pip install cryptography
```

- Support for Python 3
- Support for modern algorithms such as AESGCM and HKDF
- Improved debugability and testability
- Secure API design

Cryptography

https://cryptography.io



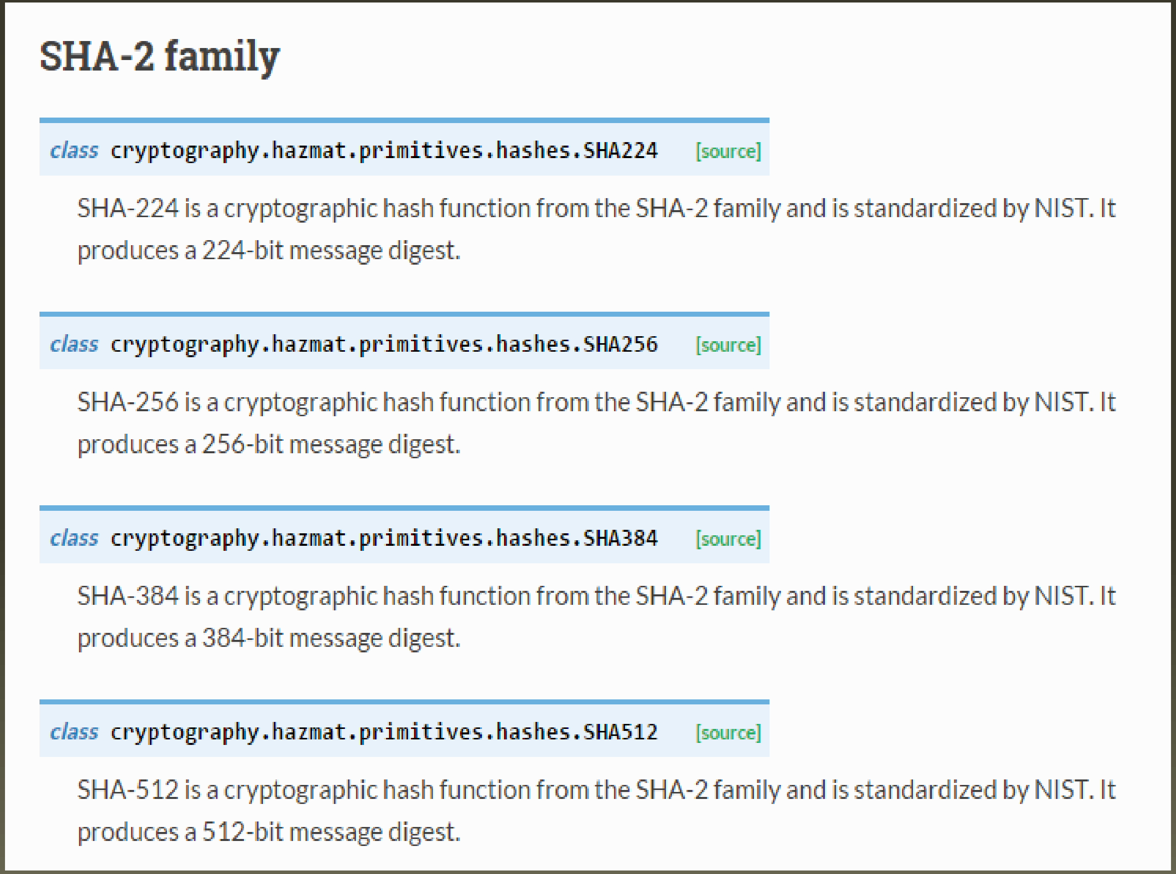
SHA-1

Attention

NIST has deprecated SHA-1 in favor of the SHA-2 variants. New applications are strongly suggested to use SHA-2 over SHA-1.

class `cryptography.hazmat.primitives.hashes.SHA1` [source]

SHA-1 is a cryptographic hash function standardized by NIST. It produces an 160-bit message digest.



SHA-2 family

class `cryptography.hazmat.primitives.hashes.SHA224` [source]

SHA-224 is a cryptographic hash function from the SHA-2 family and is standardized by NIST. It produces a 224-bit message digest.

class `cryptography.hazmat.primitives.hashes.SHA256` [source]

SHA-256 is a cryptographic hash function from the SHA-2 family and is standardized by NIST. It produces a 256-bit message digest.

class `cryptography.hazmat.primitives.hashes.SHA384` [source]

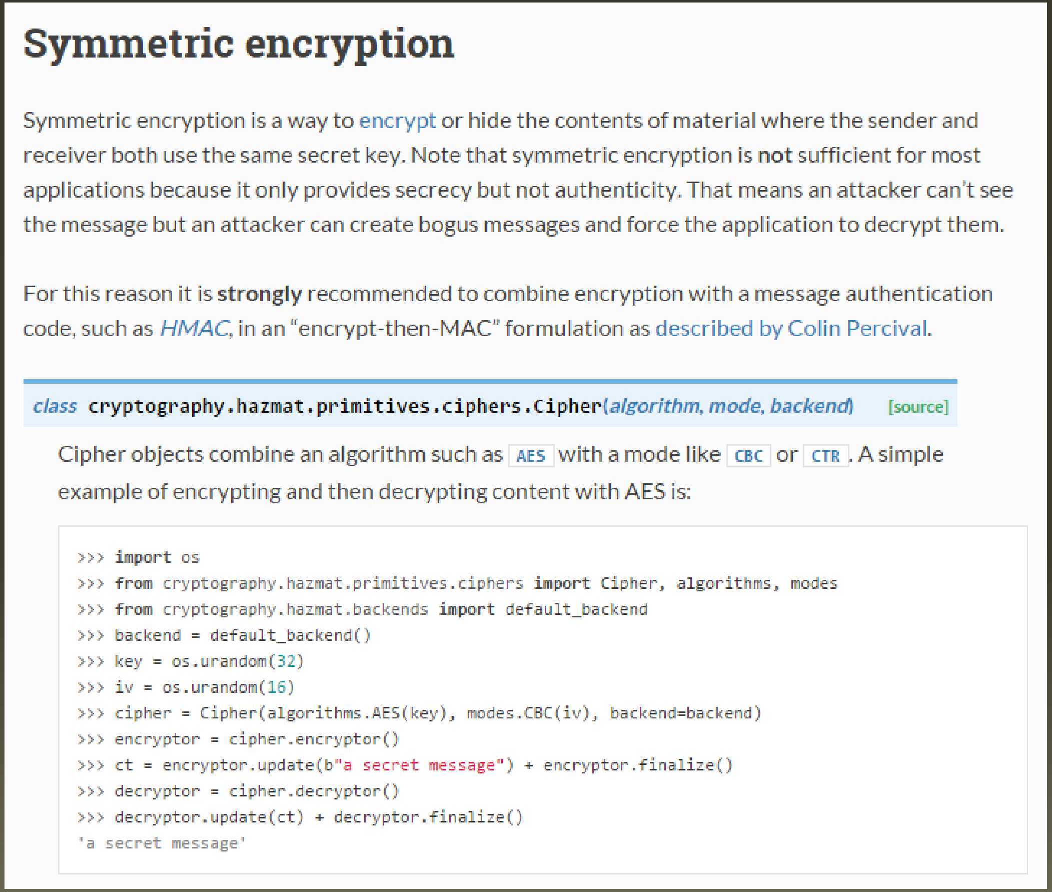
SHA-384 is a cryptographic hash function from the SHA-2 family and is standardized by NIST. It produces a 384-bit message digest.

class `cryptography.hazmat.primitives.hashes.SHA512` [source]

SHA-512 is a cryptographic hash function from the SHA-2 family and is standardized by NIST. It produces a 512-bit message digest.

Cryptography

https://cryptography.io



Symmetric encryption

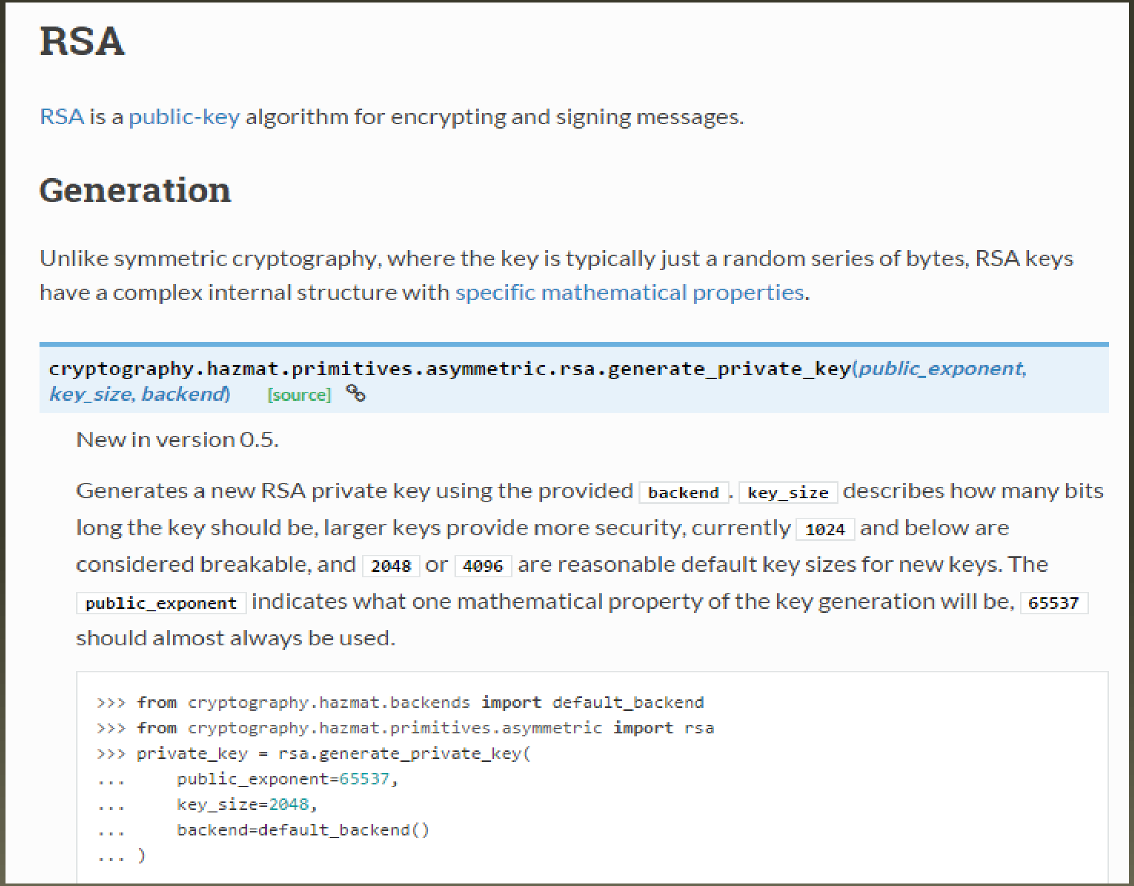
Symmetric encryption is a way to encrypt or hide the contents of material where the sender and receiver both use the same secret key. Note that symmetric encryption is **not** sufficient for most applications because it only provides secrecy but not authenticity. That means an attacker can't see the message but an attacker can create bogus messages and force the application to decrypt them.

For this reason it is **strongly** recommended to combine encryption with a message authentication code, such as *HMAC*, in an "encrypt-then-MAC" formulation as described by Colin Percival.

class `cryptography.hazmat.primitives.ciphers.Cipher`(*algorithm, mode, backend*) [source]

Cipher objects combine an algorithm such as `AES` with a mode like `CBC` or `CTR`. A simple example of encrypting and then decrypting content with AES is:

```
>>> import os
>>> from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
>>> from cryptography.hazmat.backends import default_backend
>>> backend = default_backend()
>>> key = os.urandom(32)
>>> iv = os.urandom(16)
>>> cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=backend)
>>> encryptor = cipher.encryptor()
>>> ct = encryptor.update(b"a secret message") + encryptor.finalize()
>>> decryptor = cipher.decryptor()
>>> decryptor.update(ct) + decryptor.finalize()
'a secret message'
```



RSA

RSA is a public-key algorithm for encrypting and signing messages.

Generation

Unlike symmetric cryptography, where the key is typically just a random series of bytes, RSA keys have a complex internal structure with specific mathematical properties.

`cryptography.hazmat.primitives.asymmetric.rsa.generate_private_key`(*public_exponent, key_size, backend*) [source]

New in version 0.5.

Generates a new RSA private key using the provided `backend`. `key_size` describes how many bits long the key should be, larger keys provide more security, currently `1024` and below are considered breakable, and `2048` or `4096` are reasonable default key sizes for new keys. The `public_exponent` indicates what one mathematical property of the key generation will be, `65537` should almost always be used.

```
>>> from cryptography.hazmat.backends import default_backend
>>> from cryptography.hazmat.primitives.asymmetric import rsa
>>> private_key = rsa.generate_private_key(
...     public_exponent=65537,
...     key_size=2048,
...     backend=default_backend()
... )
```

django Security

https://www.securedjango.com

- Are you vulnerable to the heartbleed bug?
- Are you enforcing SSL correctly?
- Did you set the proper flags for your cookies?
- Did you remember to disable weak ciphers?
- How are you managing your secret keys?
- Are you sure you authorise users correctly?

django Security

https://www.securedjango.com

- We can use frameworks for building API REST
  - Tastypie http://django-tastypie.readthedocs.org/en/latest
  - django-rest-framework http://django-rest-framework.org
  - dj-webmachine http://benoitc.github.com/dj-webmachine

- Django-secure package

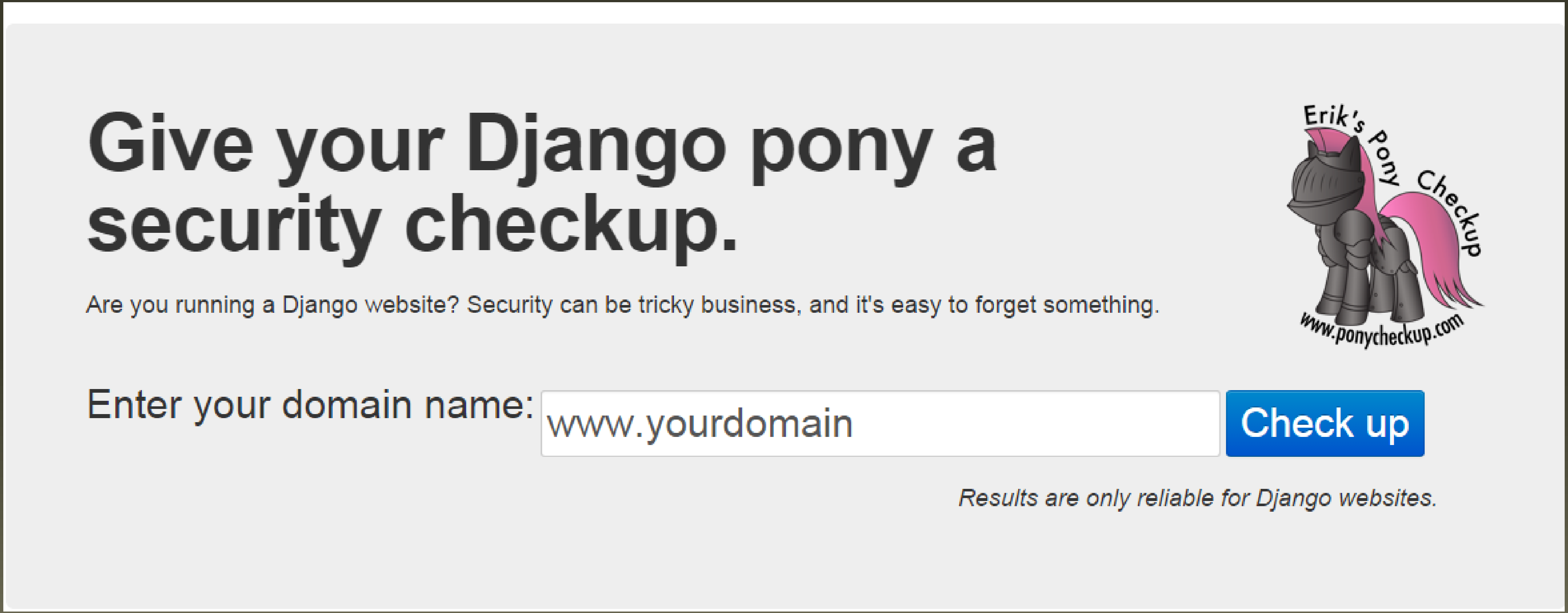
django Security

https://www.securedjango.com

- **What provide these frameworks?**
  - Cross-site scripting(XSS) Protection
  - Cross-site request forgery(CSRF) Protection
  - SQL Injection Protection
  - Clickjacking Protection
  - Supports SSL/HTTPS
  - Secure Password Storage with PBKDF2 algorithm and SHA256
  - Data Validation

django Security

https://www.ponycheckup.com/



Give your Django pony a security checkup.

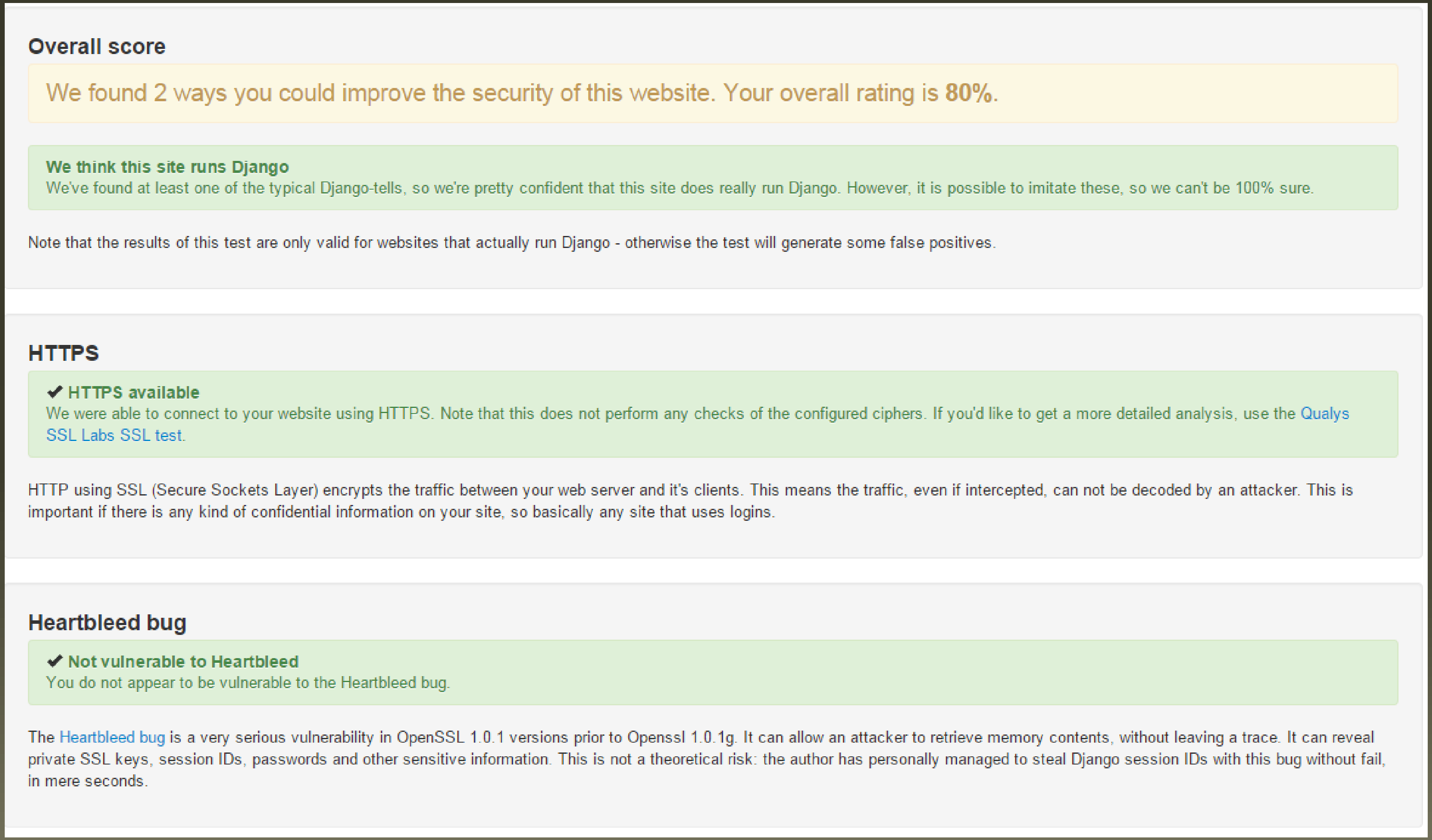
Are you running a Django website? Security can be tricky business, and it's easy to forget something.

Enter your domain name: www.yourdomain

Results are only reliable for Django websites.

django Security

https://www.ponycheckup.com/



Overall score

We found 2 ways you could improve the security of this website. Your overall rating is **80%**.

We think this site runs Django
We've found at least one of the typical Django-tells, so we're pretty confident that this site does really run Django. However, it is possible to imitate these, so we can't be 100% sure.

Note that the results of this test are only valid for websites that actually run Django - otherwise the test will generate some false positives.

HTTPS

✔ HTTPS available
We were able to connect to your website using HTTPS. Note that this does not perform any checks of the configured ciphers. If you'd like to get a more detailed analysis, use the Qualys SSL Labs SSL test.

HTTP using SSL (Secure Sockets Layer) encrypts the traffic between your web server and it's clients. This means the traffic, even if intercepted, can not be decoded by an attacker. This is important if there is any kind of confidential information on your site, so basically any site that uses logins.

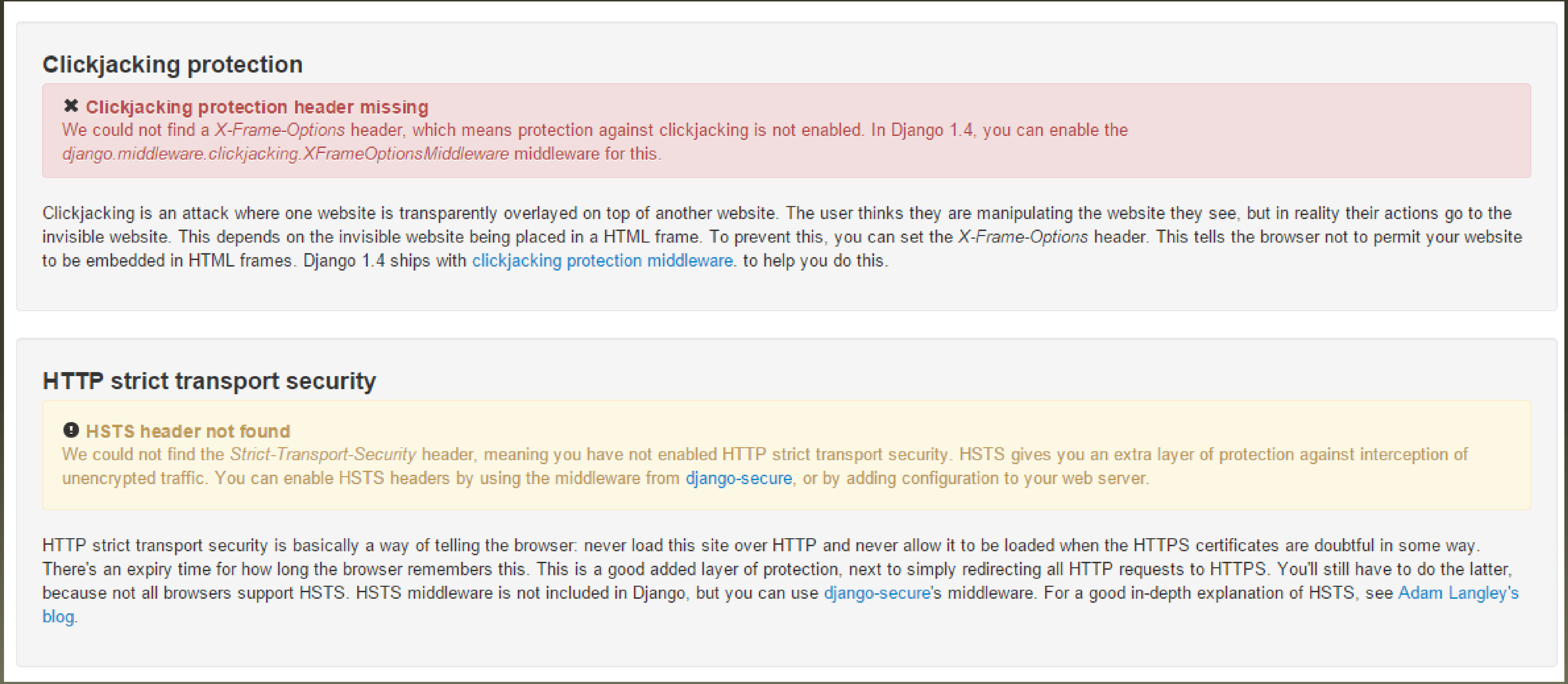
Heartbleed bug

✔ Not vulnerable to Heartbleed
You do not appear to be vulnerable to the Heartbleed bug.

The Heartbleed bug is a very serious vulnerability in OpenSSL 1.0.1 versions prior to Openssl 1.0.1g. It can allow an attacker to retrieve memory contents, without leaving a trace. It can reveal private SSL keys, session IDs, passwords and other sensitive information. This is not a theoretical risk: the author has personally managed to steal Django session IDs with this bug without fail, in mere seconds.

django Security

https://www.ponycheckup.com/



Clickjacking protection

✖ **Clickjacking protection header missing**
We could not find a *X-Frame-Options* header, which means protection against clickjacking is not enabled. In Django 1.4, you can enable the *django.middleware.clickjacking.XFrameOptionsMiddleware* middleware for this.

Clickjacking is an attack where one website is transparently overlayed on top of another website. The user thinks they are manipulating the website they see, but in reality their actions go to the invisible website. This depends on the invisible website being placed in a HTML frame. To prevent this, you can set the *X-Frame-Options* header. This tells the browser not to permit your website to be embedded in HTML frames. Django 1.4 ships with clickjacking protection middleware. to help you do this.

HTTP strict transport security

HSTS header not found
We could not find the *Strict-Transport-Security* header, meaning you have not enabled HTTP strict transport security. HSTS gives you an extra layer of protection against interception of unencrypted traffic. You can enable HSTS headers by using the middleware from django-secure, or by adding configuration to your web server.

HTTP strict transport security is basically a way of telling the browser: never load this site over HTTP and never allow it to be loaded when the HTTPS certificates are doubtful in some way. There's an expiry time for how long the browser remembers this. This is a good added layer of protection, next to simply redirecting all HTTP requests to HTTPS. You'll still have to do the latter, because not all browsers support HSTS. HSTS middleware is not included in Django, but you can use django-secure's middleware. For a good in-depth explanation of HSTS, see Adam Langley's blog.

Security best practices

https://www.

- **Always use HTTPS if you have anything non-public**
- **Proper SSL deployment**
  - Enable HTTPS with a proper server certificate
  - Enforce HTTPS on your entire domain
  - Configure redirects to enforce HTTPS usage
  - Set the secure flag on all cookies
  - **Django only send session cookies over HTTPS**

```
SESSION_COOKIE_SECURE = true
CSRF_COOKIE_SECURE_true
```

Security best practices

- Keep in secrets keys and credentials
- Put DEBUG=false in production in settings.py
- Use ALLOWED_HOSTS variable in production for setting a list of request allowed hosts names
- Limit access to admin with IP`s filter

```
ALLOWED_HOSTS =[*]
```

```
ALLOWED_HOSTS =['.yourdomain.com']
```

Password storage

- **PBKDF2 + SHA256 by default**

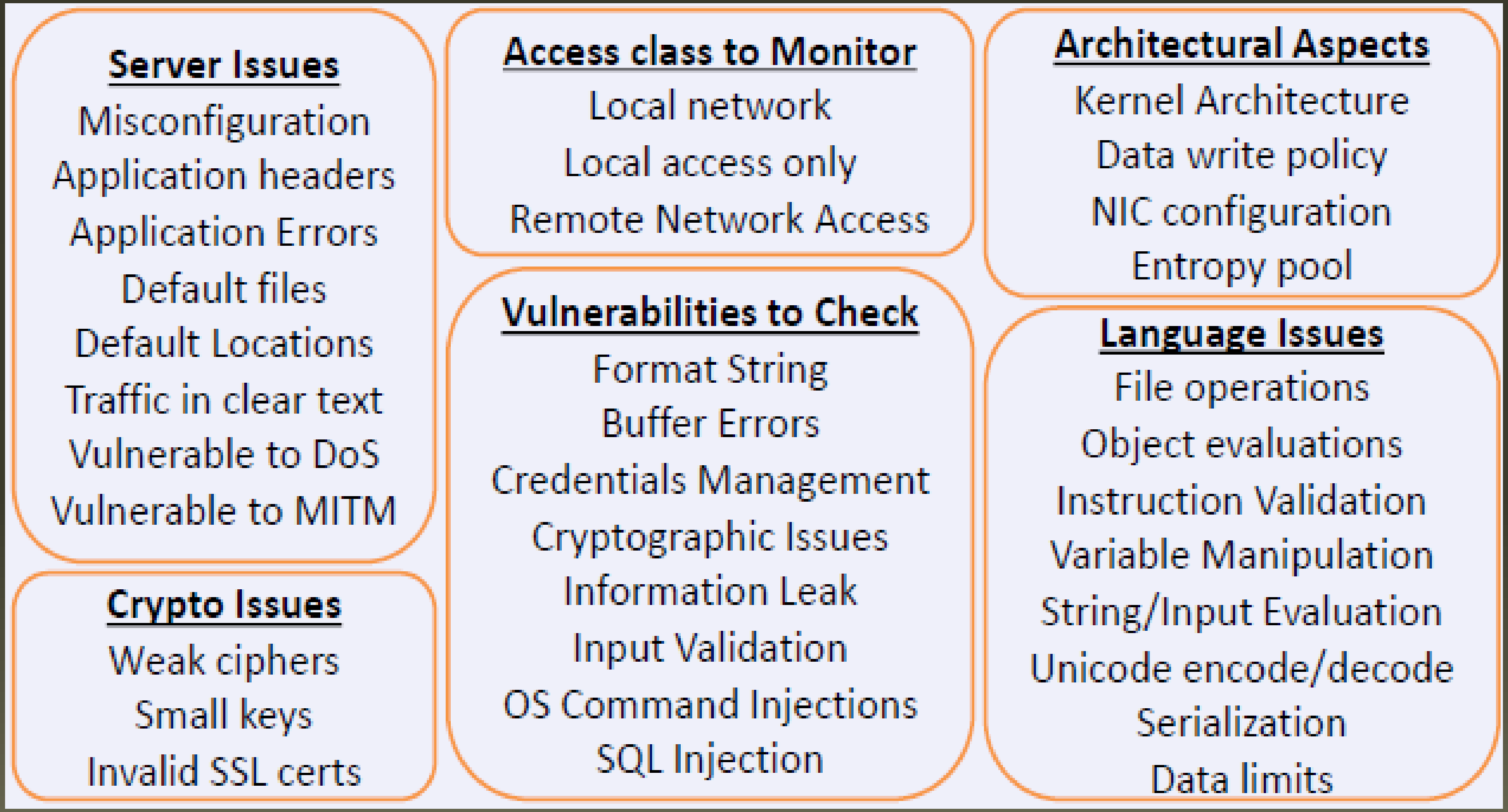
```
PASSWORD_HASHERS = (
'django.contrib.auth.hashers.PBKDF2PasswordHasher',
'django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher',
'django.contrib.auth.hashers.SHA1PasswordHasher',
'django.contrib.auth.hashers.MD5PasswordHasher',
'django.contrib.auth.hashers.CryptPasswordHasher')
```

https://github.com/django/django/blob/master/django/contrib/auth/hashers.py/

Password storage

```
class PBKDF2PasswordHasher(BasePasswordHasher):
    """
    Secure password hashing using the PBKDF2 algorithm (recommended)
    Configured to use PBKDF2 + HMAC + SHA256.
    The result is a 64 byte binary string.  Iterations may be changed
    safely but you must rename the algorithm if you change SHA256.
    """
    algorithm = "pbkdf2_sha256"
    iterations = 24000
    digest = hashlib.sha256

    def encode(self, password, salt, iterations=None):
        assert password is not None
        assert salt and '$' not in salt
        if not iterations:
            iterations = self.iterations
        hash = pbkdf2(password, salt, iterations, digest=self.digest)
        hash = base64.b64encode(hash).decode('ascii').strip()
        return "%s$%d$%s$%s" % (self.algorithm, iterations, salt, hash)
```

OWASP



Server Issues
Misconfiguration
Application headers
Application Errors
Default files
Default Locations
Traffic in clear text
Vulnerable to DoS
Vulnerable to MITM

Crypto Issues
Weak ciphers
Small keys
Invalid SSL certs

Access class to Monitor
Local network
Local access only
Remote Network Access

Vulnerabilities to Check
Format String
Buffer Errors
Credentials Management
Cryptographic Issues
Information Leak
Input Validation
OS Command Injections
SQL Injection

Architectural Aspects
Kernel Architecture
Data write policy
NIC configuration
Entropy pool

Language Issues
File operations
Object evaluations
Instruction Validation
Variable Manipulation
String/Input Evaluation
Unicode encode/decode
Serialization
Data limits

OWASP

- *SQL injection*

- **Cross site Scripting(XSS)**

SQL injection

- *Never* trust user-submitted data
- **Django generates properly-escaped parameters SQL**
- Using cursor method and **bind parameter** is the best option for avoid SQL INJECT

```
from django.db import connection

def select_user(request):
        user = request.GET['username']
        sql = "SELECT * FROM users WHERE username = %s"
        cursor = connection.cursor()
        cursor.execute(sql, [user])
```

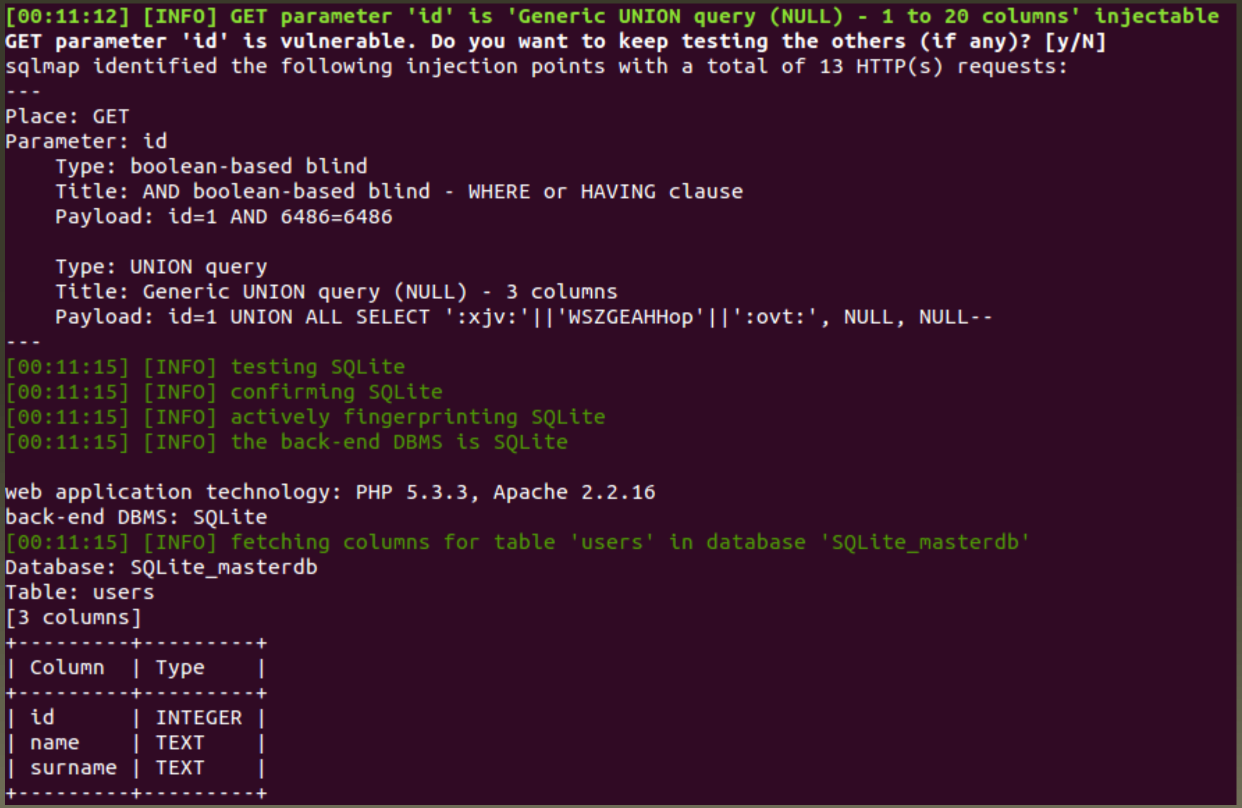
SQL injection

- *Django ORM –QuerySets -Models*
- Django automatically gives you a database-abstraction API that lets you create, retrieve, update and delete objects
- Write python classes and it will convert to SQL securely

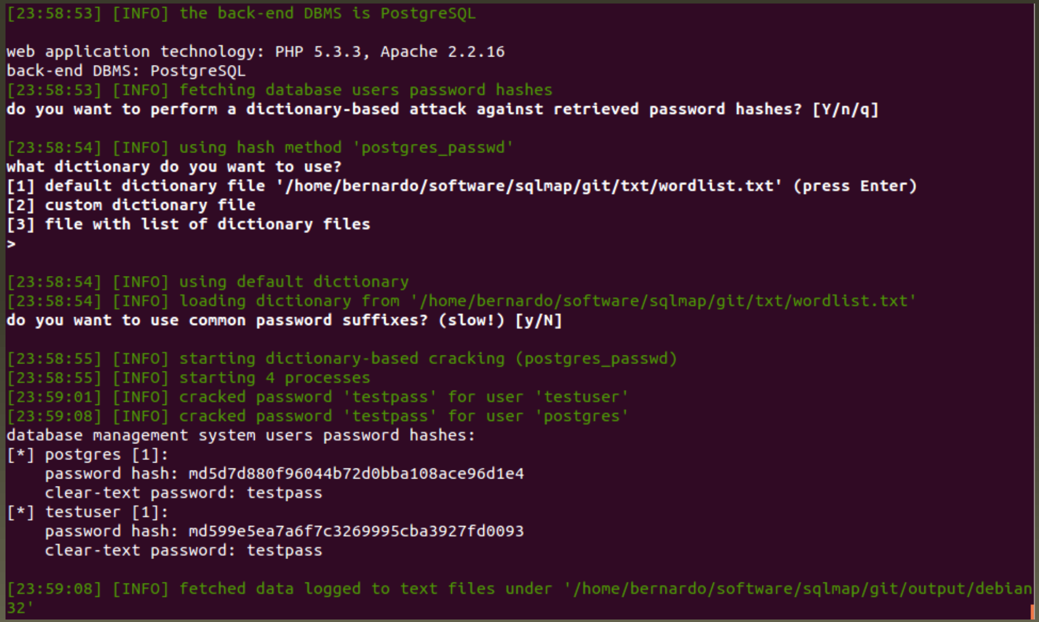
```
from django.db import models

class Blog(models.Model):
        name = models.CharField(max_length=100)
        description = models.TextField()

 >>b = Blog(name='My Bblog', description='django security')
 >>> b.save()
```

SQL injection

- *SQLMAP*



```
[00:11:12] [INFO] GET parameter 'id' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N]
sqlmap identified the following injection points with a total of 13 HTTP(s) requests:
---
Place: GET
Parameter: id
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1 AND 6486=6486

    Type: UNION query
    Title: Generic UNION query (NULL) - 3 columns
    Payload: id=1 UNION ALL SELECT ':xjv:'||'WSZGEAHHop'||':ovt:', NULL, NULL--
---
[00:11:15] [INFO] testing SQLite
[00:11:15] [INFO] confirming SQLite
[00:11:15] [INFO] actively fingerprinting SQLite
[00:11:15] [INFO] the back-end DBMS is SQLite

web application technology: PHP 5.3.3, Apache 2.2.16
back-end DBMS: SQLite
[00:11:15] [INFO] fetching columns for table 'users' in database 'SQLite_masterdb'
Database: SQLite_masterdb
Table: users
[3 columns]
+---------+---------+
| Column  | Type    |
+---------+---------+
| id      | INTEGER |
| name    | TEXT    |
| surname | TEXT    |
+---------+---------+
```



```
[23:58:53] [INFO] the back-end DBMS is PostgreSQL

web application technology: PHP 5.3.3, Apache 2.2.16
back-end DBMS: PostgreSQL
[23:58:53] [INFO] fetching database users password hashes
do you want to perform a dictionary-based attack against retrieved password hashes? [Y/n/q]

[23:58:54] [INFO] using hash method 'postgres_passwd'
what dictionary do you want to use?
[1] default dictionary file '/home/bernardo/software/sqlmap/git/txt/wordlist.txt' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
>

[23:58:54] [INFO] using default dictionary
[23:58:54] [INFO] loading dictionary from '/home/bernardo/software/sqlmap/git/txt/wordlist.txt'
do you want to use common password suffixes? (slow!) [y/N]

[23:58:55] [INFO] starting dictionary-based cracking (postgres_passwd)
[23:58:55] [INFO] starting 4 processes
[23:59:01] [INFO] cracked password 'testpass' for user 'testuser'
[23:59:08] [INFO] cracked password 'testpass' for user 'postgres'
database management system users password hashes:
[*] postgres [1]:
    password hash: md5d7d880f96044b72d0bba108ace96d1e4
    clear-text password: testpass
[*] testuser [1]:
    password hash: md599e5ea7a6f7c3269995cba3927fd0093
    clear-text password: testpass

[23:59:08] [INFO] fetched data logged to text files under '/home/bernardo/software/sqlmap/git/output/debian
32'
```

Cross site Scripting

- *Allows an attacker obtain session information*
- ***Used with in phising sites***
- Django’s **render template** system automatically escapes all variable values in HTML

```
from django.shortcuts import render

def render_page(request):
        user = request.GET['username']
        return render(request, ‘page.html', {‘user’: user})
```

Security best practices in forms

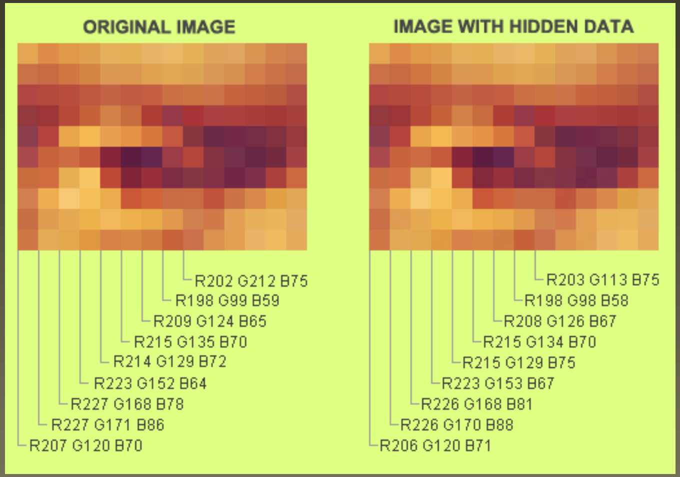
- Validate form data with Django Forms package
- Use POST method in HTML Forms
- Use Meta.Fields in ModelForms

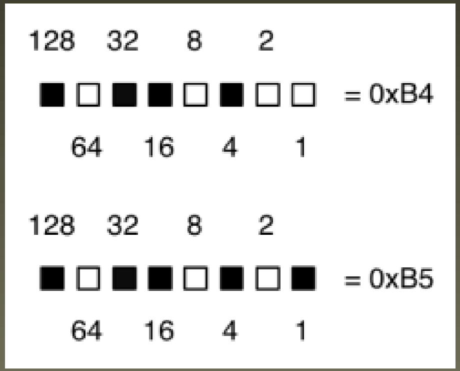
Steganography

- Hiding data(text/images) within images
- **Where is stored the data?**

Steganography

- In the pixels in RGB Components
- Altering the Least Significant Bit(LSB)
- Use one bit per pixel for storing data

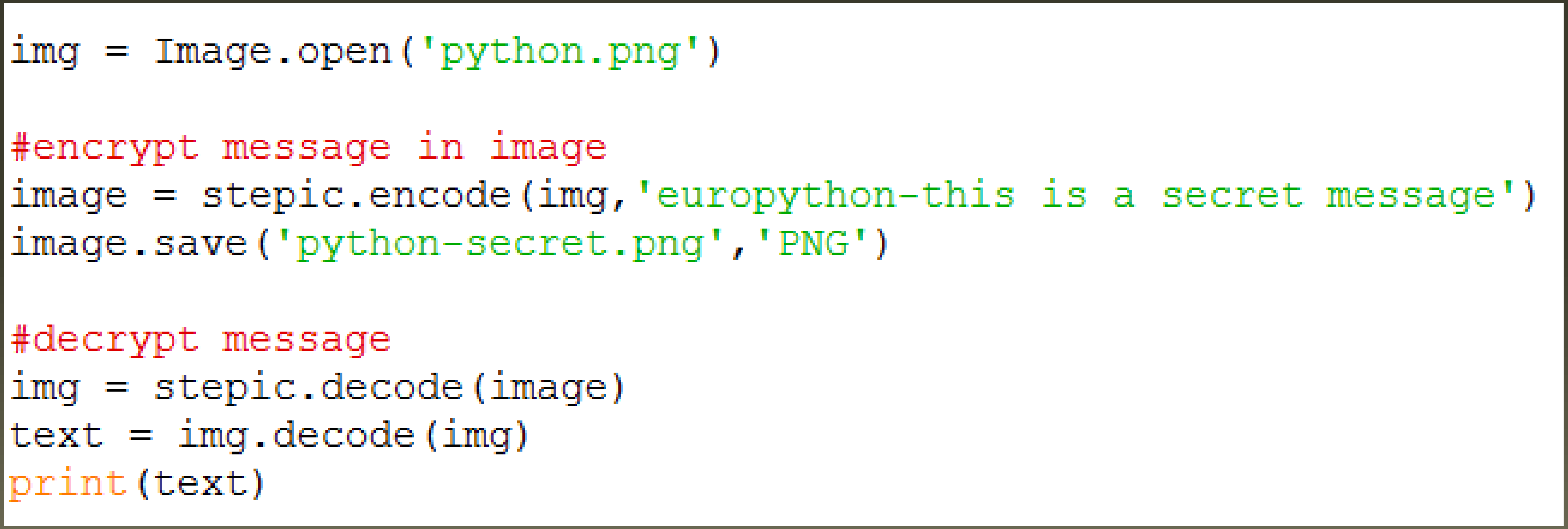




Libraries in python

- *Stepic*

http://domnit.org/stepic/doc



```
img = Image.open('python.png')

#encrypt message in image
image = stepic.encode(img,'europython-this is a secret message')
image.save('python-secret.png','PNG')

#decrypt message
img = stepic.decode(image)
text = img.decode(img)
print(text)
```

Libraries

- *Stegano*

https://code.google.com/p/stegano-cb

```
from stegano import slsb
secret = slsb.hide("python.png", "europython-this is a secret message")

secret.save("python-secret.png")
slsb.reveal("python-secret.png")
```

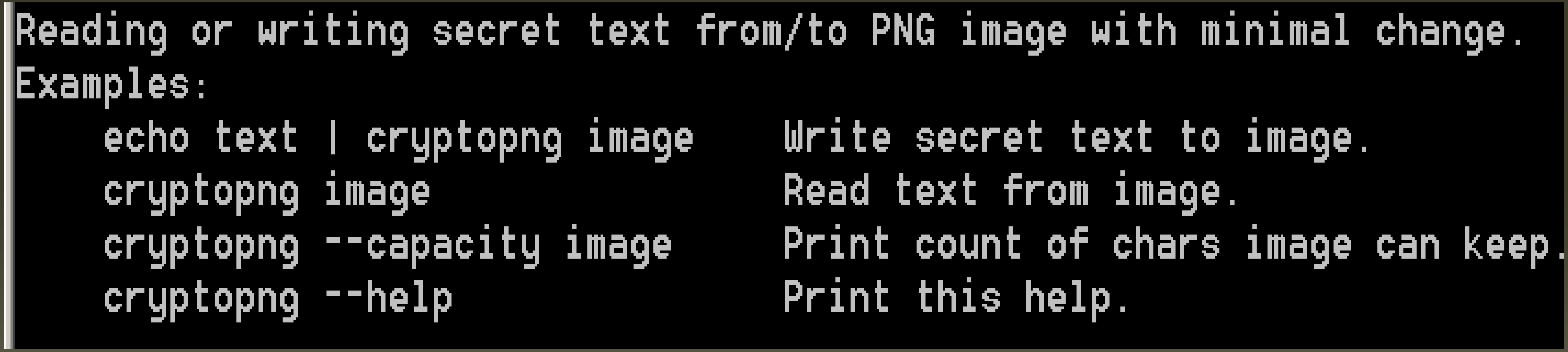
```
$ slsb.py --hide -i python.png -o python-secret.png –m "euro.."
```

```
$ slsb.py --hide -i python.png -o python-secret.png –f img.png
```

Tools

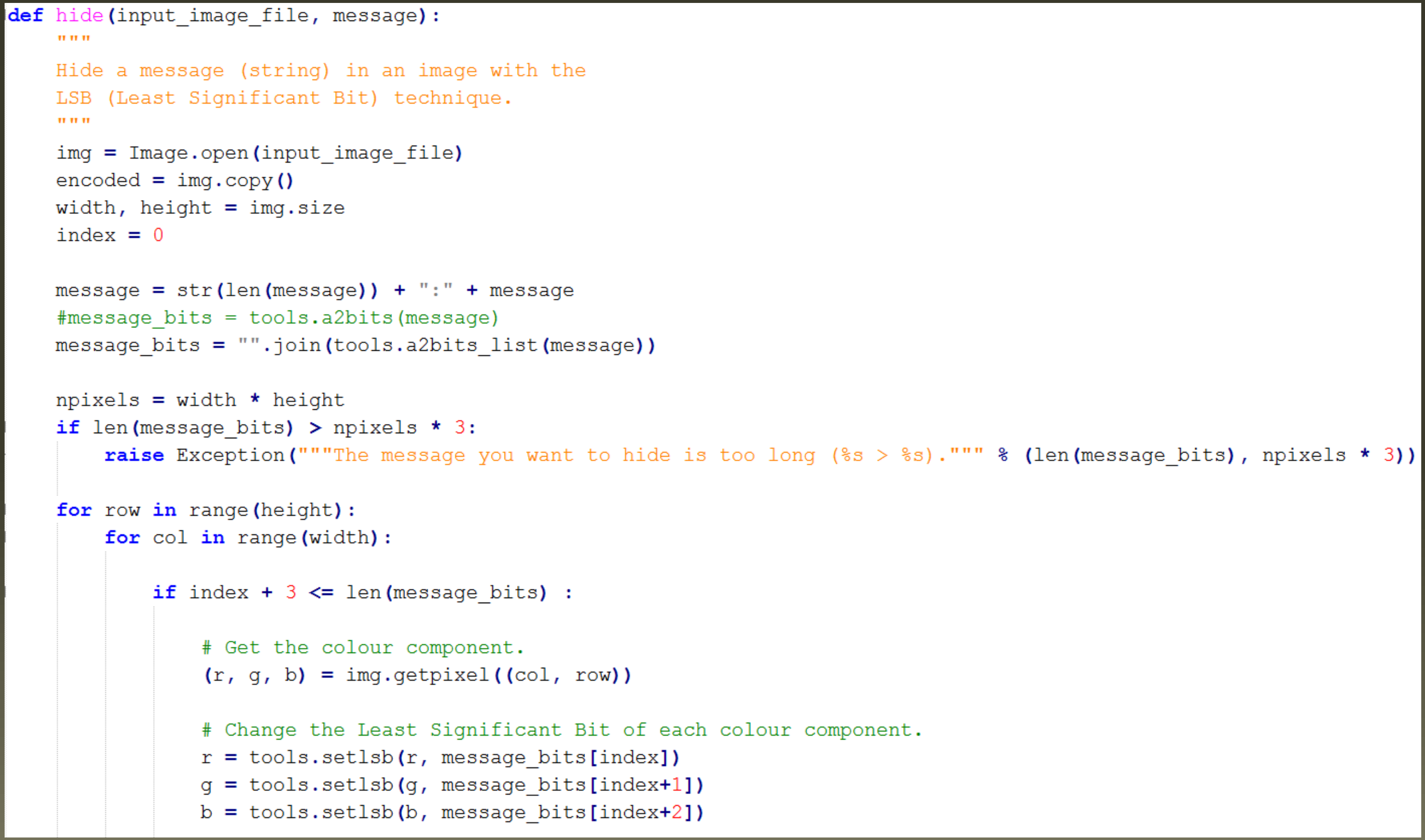
- ***Cryptopng***

https://pypi.python.org/pypi/cryptoPNG/0.1



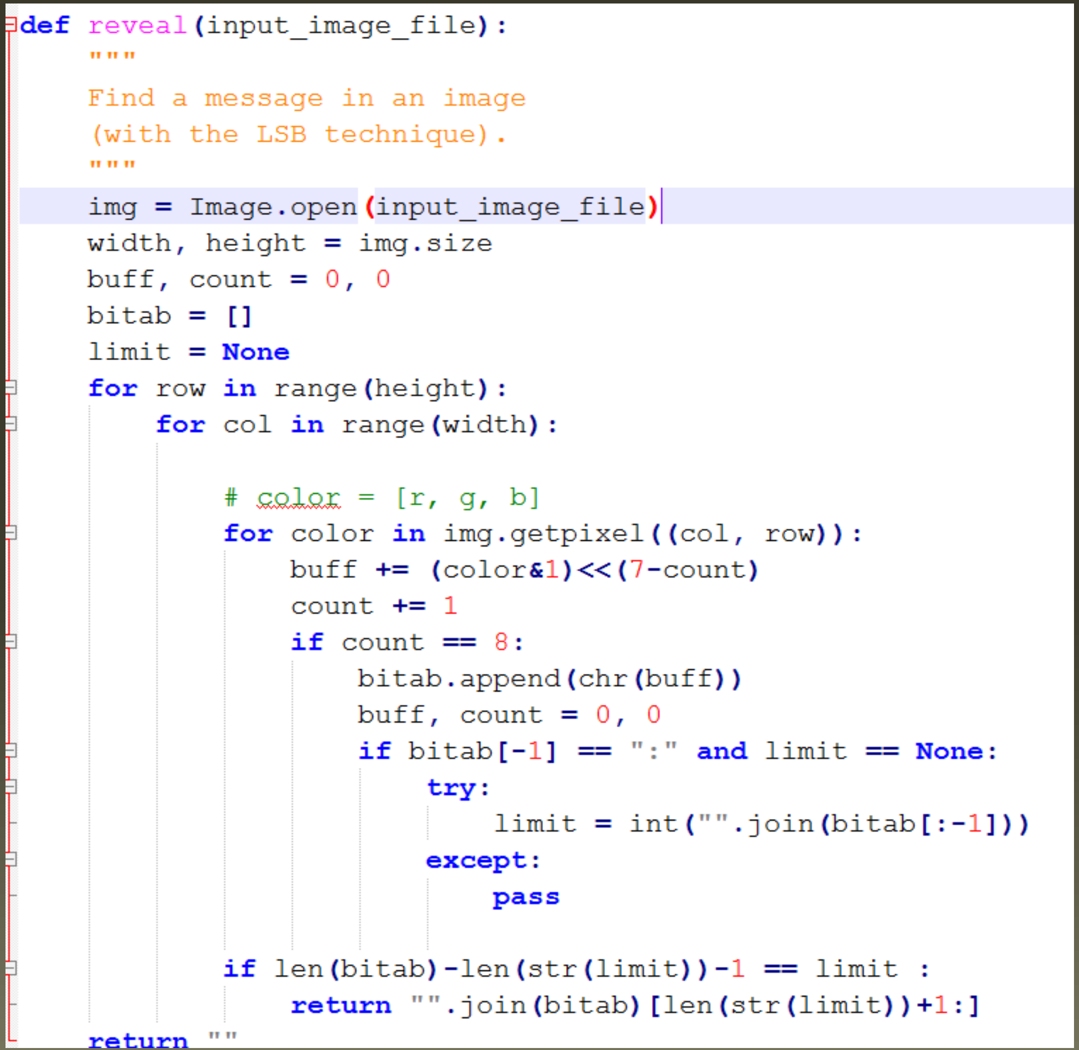
```
Reading or writing secret text from/to PNG image with minimal change.
Examples:
    echo text | cryptopng image    Write secret text to image.
    cryptopng image                Read text from image.
    cryptopng --capacity image     Print count of chars image can keep.
    cryptopng --help               Print this help.
```

Hide text in image(LSB)



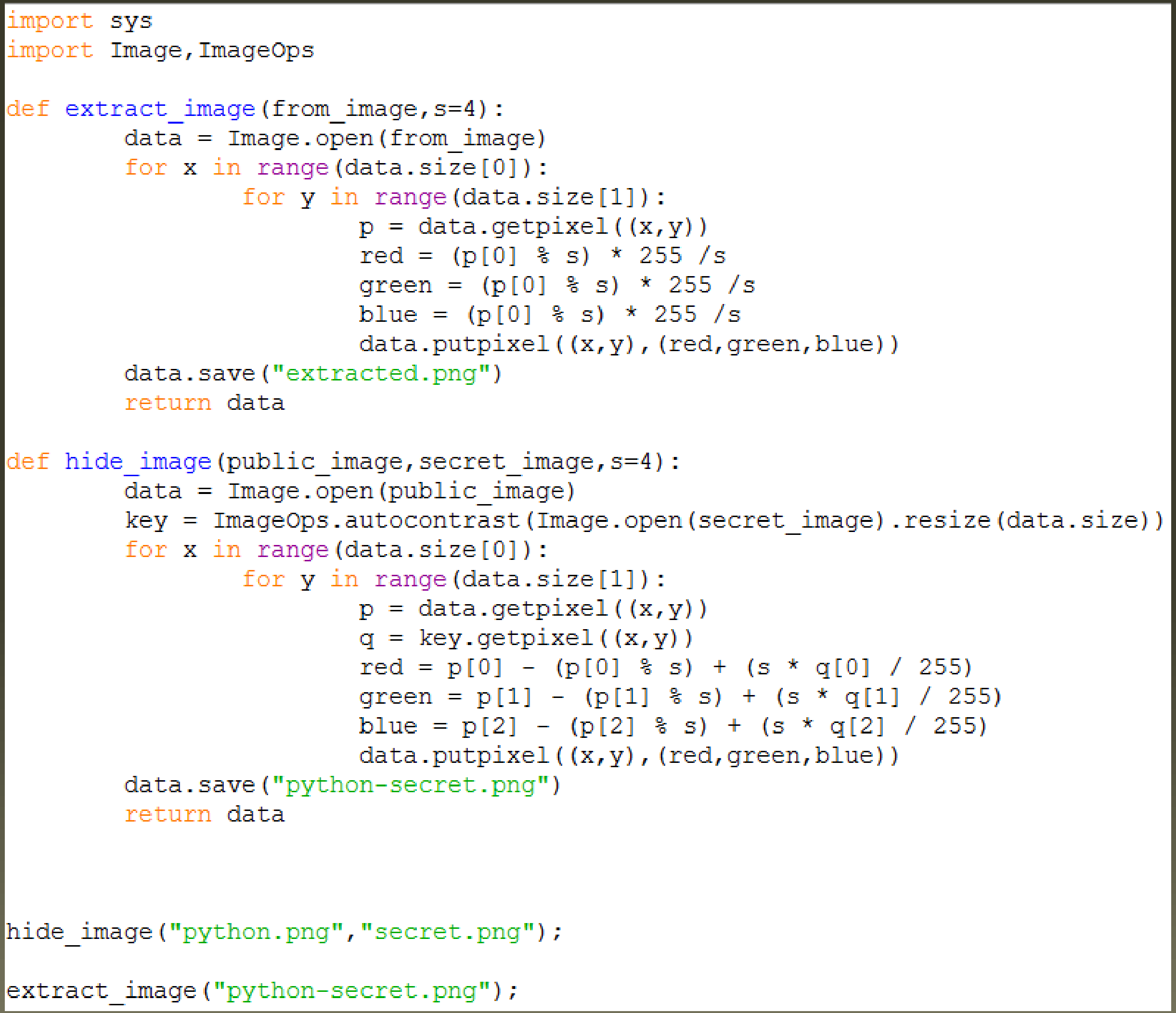
```
def hide(input_image_file, message):
    """
    Hide a message (string) in an image with the
    LSB (Least Significant Bit) technique.
    """
    img = Image.open(input_image_file)
    encoded = img.copy()
    width, height = img.size
    index = 0

    message = str(len(message)) + ":" + message
    #message_bits = tools.a2bits(message)
    message_bits = "".join(tools.a2bits_list(message))

    npixels = width * height
    if len(message_bits) > npixels * 3:
        raise Exception("""The message you want to hide is too long (%s > %s).""" % (len(message_bits), npixels * 3))

    for row in range(height):
        for col in range(width):

            if index + 3 <= len(message_bits) :

                # Get the colour component.
                (r, g, b) = img.getpixel((col, row))

                # Change the Least Significant Bit of each colour component.
                r = tools.setlsb(r, message_bits[index])
                g = tools.setlsb(g, message_bits[index+1])
                b = tools.setlsb(b, message_bits[index+2])
```

Reveal text from image(LSB)

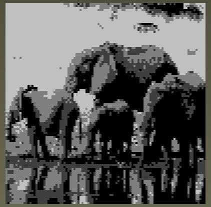


```
def reveal(input_image_file):
    """
    Find a message in an image
    (with the LSB technique).
    """
    img = Image.open(input_image_file)
    width, height = img.size
    buff, count = 0, 0
    bitab = []
    limit = None
    for row in range(height):
        for col in range(width):

            # color = [r, g, b]
            for color in img.getpixel((col, row)):
                buff += (color&1)<<(7-count)
                count += 1
                if count == 8:
                    bitab.append(chr(buff))
                    buff, count = 0, 0
                    if bitab[-1] == ":" and limit == None:
                        try:
                            limit = int("".join(bitab[:-1]))
                        except:
                            pass

            if len(bitab)-len(str(limit))-1 == limit :
                return "".join(bitab)[len(str(limit))+1:]
    return ""
```

Hide image inside an image

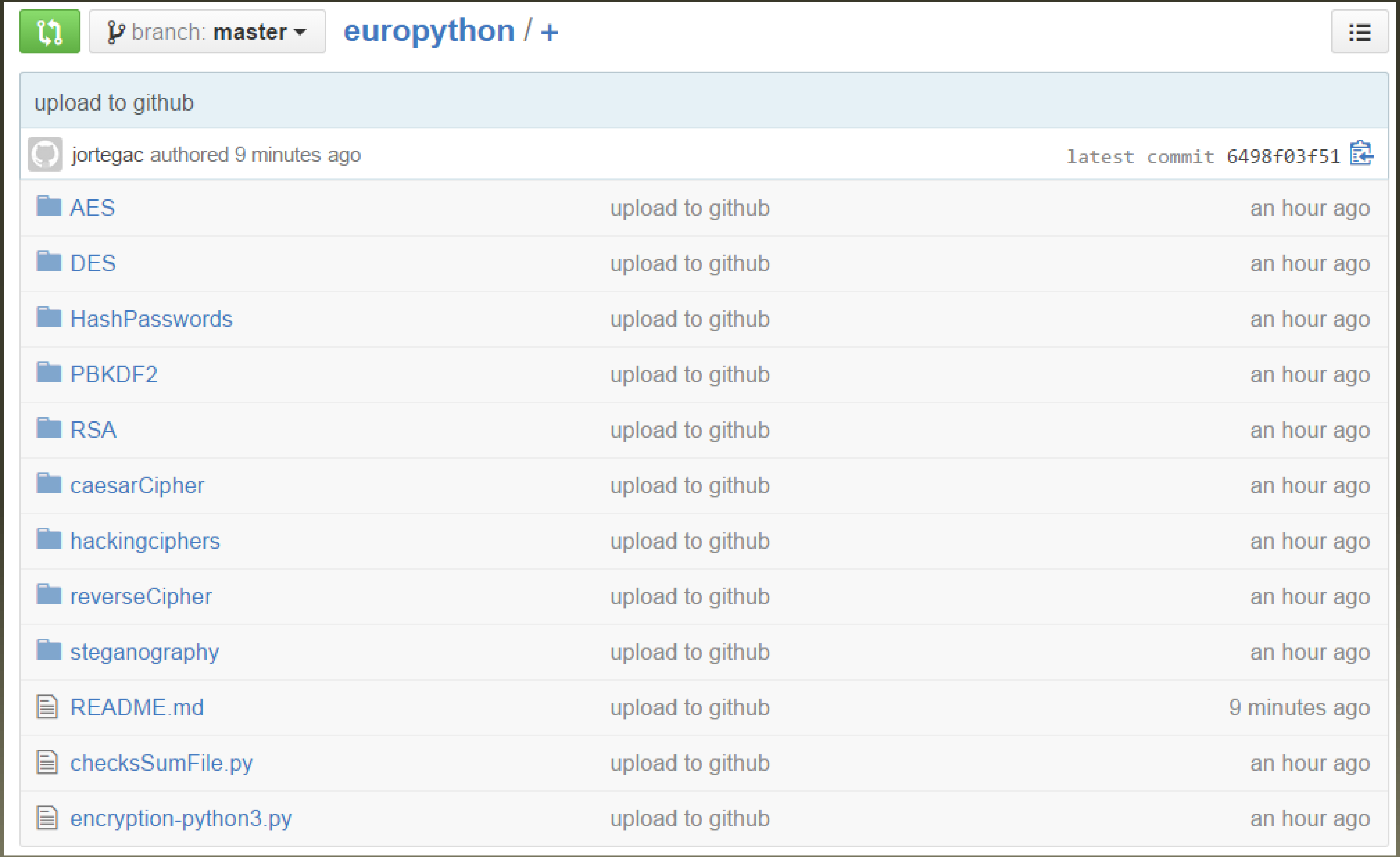


```python
import sys
import Image,ImageOps

def extract_image(from_image,s=4):
        data = Image.open(from_image)
        for x in range(data.size[0]):
                for y in range(data.size[1]):
                        p = data.getpixel((x,y))
                        red = (p[0] % s) * 255 /s
                        green = (p[0] % s) * 255 /s
                        blue = (p[0] % s) * 255 /s
                        data.putpixel((x,y),(red,green,blue))
        data.save("extracted.png")
        return data

def hide_image(public_image,secret_image,s=4):
        data = Image.open(public_image)
        key = ImageOps.autocontrast(Image.open(secret_image).resize(data.size))
        for x in range(data.size[0]):
                for y in range(data.size[1]):
                        p = data.getpixel((x,y))
                        q = key.getpixel((x,y))
                        red = p[0] - (p[0] % s) + (s * q[0] / 255)
                        green = p[1] - (p[1] % s) + (s * q[1] / 255)
                        blue = p[2] - (p[2] % s) + (s * q[2] / 255)
                        data.putpixel((x,y),(red,green,blue))
        data.save("python-secret.png")
        return data



hide_image("python.png","secret.png");

extract_image("python-secret.png");
```



GitHub

https://github.com/jmortega/europython



branch: master europython / +

upload to github

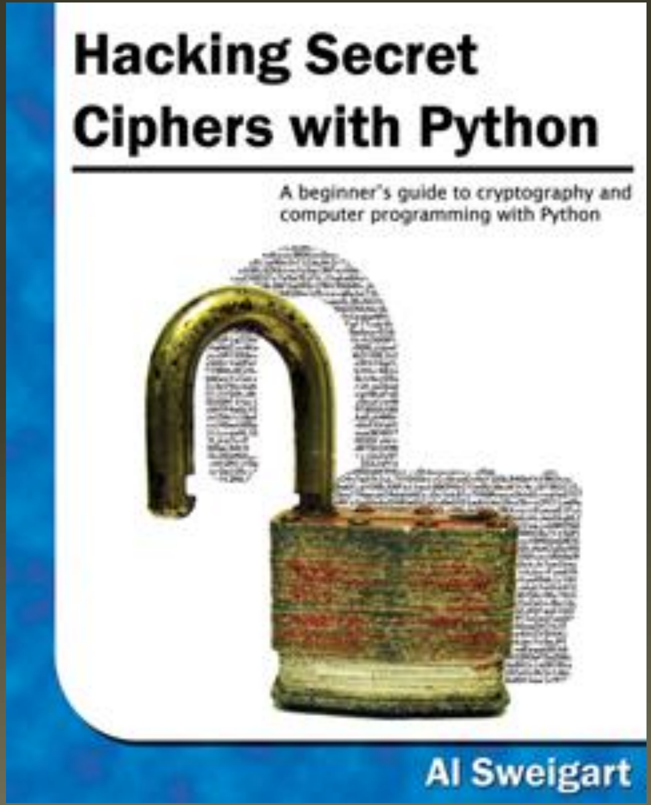
jortegac authored 9 minutes ago latest commit 6498f03f51

AES	upload to github	an hour ago
DES	upload to github	an hour ago
HashPasswords	upload to github	an hour ago
PBKDF2	upload to github	an hour ago
RSA	upload to github	an hour ago
caesarCipher	upload to github	an hour ago
hackingciphers	upload to github	an hour ago
reverseCipher	upload to github	an hour ago
steganography	upload to github	an hour ago
README.md	upload to github	9 minutes ago
checksSumFile.py	upload to github	an hour ago
encryption-python3.py	upload to github	an hour ago

Book

http://inventwithpython.com/hacking/index.html

- Hacking Secret Ciphers with python
- Free online


Hacking Secret Ciphers with Python

A beginner's guide to cryptography and computer programming with Python

Al Sweigart

Thank you!

José Manuel Ortega| @jmortegac