Hash & crypto

Sergey Kirgizov

September 3, 2024

The goal of these exercises is to learn how to use cryptographic hashing and asymmetric encryption. You can use any programming language of your choice. My examples will be in Python.

---

1 Hash functions

There is a piece of code with helper function to hash a string.

```
from hashlib import sha384
def strhash (v):
  return sha384 (v.encode()).hexdigest()

# example
strhash ('DIJON')
```

☞ EXERCISE 1 : Use a cryptographic hash function to organize a simple password store. Your system should have at least three functions:

- **register (user, password)** to create a new user with a password;
- **login (user, password)** to check if a user entered a good password;
- **change (user, oldpassword, newpasswod)** to change the password.

You can store the data in a file or directly in RAM.

☞ EXERCISE★ 2 : Think and implement an non-synchronous version of the game rock-paper-scissors. How we should proceed in order to be able to play this game by email without giving the other person an advantage?

☞ EXERCISE 3 : Write a basic transaction storage system. The system should store the data in a file and support the following functions

- **add (person1, person2, amount, date)** to add a new transaction.
- **list_transactions ()** to show the list of all transactions.
- **verify ()** to check the integrity of the transaction list.

Every transaction should be stored together with a hash value calculated from the data of the current transaction and the hash value of the previous transaction. If someone or something changes the data in the file the function **verify()** can be used to detect the problem. In what cases integrity verification will not be possible?

☞ EXERCISE★ 4 : Add a new function, **balance (person)**, to be able to calculate how much money the specified user has.

2 RSA by hand

The simple version of RSA cryptosystem is described below.

Key generation

1. Choose two prime numbers p and q.
2. Calculate their product $n = pq$.
3. Calculate the Euler totient function $\varphi(n) = (p-1)(q-1)$.
4. Choose an integer e such that $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$.
5. Find d such that $de = 1 \pmod{\varphi(n)}$
6. The couple (e, n) is a public key.
7. The couple (d, n) is a private key.

Encryption
To encrypt the message represented by an integer m such that $0 \leq m < n$, we need to calculate $m^e \pmod{n}$.

Decryption
To decrypt the message represented by an integer w, we calculate $w^d \pmod{n}$.

☞ **EXERCISE 5 :** Implement the simple version of RSA cryptosystem using integers. The key generation can be done manually.

Example

Public key is (5,119).
Private key is (77,119).
The message is $m = 99$.
Encrypted message is $w = m^5 \pmod{119} = 29$.
Decryption $29^{77} \pmod{119}$
Can you find which p and q are used in this case?

☞ **EXERCISE★ 6 :** Implement the automatic key generation for the simple version of RSA.

BE CAREFUL with small numbers, brute-force attacks are real!

In the real-world applications one must choose very big p, q and scrupulously implement some additional mechanisms (proper padding scheme, etc) to confront known problems of naive RSA. See `https://en.wikipedia.org/wiki/RSA_(cryptosystem)` for more details.

Other asymmetric encryption algorithm also exists: Elliptic-curve cryptography, Diffie–Hellman key exchange, ElGamal, Unbalanced oil and vinegar scheme. etc...

See also `https://en.wikipedia.org/wiki/Post-quantum_cryptography`

3 RSA by lib

Here we will use a library “pycryptodome” that implements RSA. You can choose another library if you wish.

EXERCISE 7 : Install the library:

```
pip install pycryptodome
```

.

EXERCISE 8 : Read, understand and test the following code

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

# Key generation
key_pair = RSA.generate(4096)
pubk = key_pair.public_key()
print (f'Public key ({pubk.e}, {pubk.n})')
print (f'Private key ({key_pair.d}, {key_pair.n})')

# store the pair of keys
f = open('private_and_public.pem','wb')
f.write(key_pair.export_key('PEM'))
f.close()

# store only the public key
f = open('public.pem','wb')
f.write(pubk.export_key('PEM'))
f.close()

# Read the key if needed
# f = open('public.pem','r')
# key = RSA.import_key(f.read())
# f.close()


# Encryption
m = 'A very secret message'
m_in_bytes = m.encode('utf-8')

encryptor = PKCS1_OAEP.new(pubk)
w = encryptor.encrypt (m_in_bytes)
print ('Encrypted text:')
print (w)

# of course we can store or send w.

decryptor = PKCS1_OAEP.new(key_pair)
decrypted_message = decryptor.decrypt (w)
print('Decrypted text:')
print(decrypted_message)
```

See also:
https://cryptobook.nakov.com/asymmetric-key-ciphers/rsa-encrypt-decrypt-examples
https://pycryptodome.readthedocs.io/en/latest/src/public_key/rsa.html

EXERCISE* 9 : Learn how to sign messages with RSA https://pycryptodome.readthedocs.io/en/latest/src/signature/pkcs1_v1_5.html