# Lab - Asymmetric Cryptography

In this guide we will develop programs that use cryptographic methods, relying in the Python3 Cryptography module. The module can be installed using the typical package management methods (e.g, `apt install python3-cryptography`), or using the `pip3` tool (e.g. `pip3 install cryptography`).

It will be useful to visualize and edit files in binary format. For that purpose, if you are using Linux, you may install `GHex` or `hexedit` from the repositories.

We will be exploring the low level interface of the `python cryptography` library, for educational purposes. If you plan to use this library in real world application, stay with the Fernet interface.

As the documentation clearly states:

```
This is a “Hazardous Materials” module. You should ONLY use it if you’re 100% absolutely sure that you know what you’re doing because this module is full of land mines, dragons, and dinosaurs with laser guns.
```

## Key pair generation

Create a small program to generate an `RSA` key pair, with a key length specified by the user, that could be one of the following values: 1024, 2048, 3072 and 4096. The program must save the key pair in two files, one for the private key and the other for the public key, whose names should also be specified by the user. The keys can be saved as binary blobs, but probably, the `PEM` format will be more appropriate.

Run the program, several times, varying the key length.

**Questions:**

- What do you think of using 4096 bit keys by default in relation to speed?
- How the actual key size varies with the number of bits?

## RSA Encryption

Create a small program to encrypt a file using the `RSA` algorithm.

The user must indicate the following data: (i) the name of the original file to encrypt, (ii) the name of the file with the public key, (iii) the name for the encrypted file.

**Note:** Pay attention to the size of the original file, i.e., the file to encrypt. Using the PKCS#1, the block size is equal to the **key size** minus eleven bytes (eleven bytes for padding). So, using 1024 bits `RSA` key (128 bytes) the block size is 117 bytes (128 - 11). Other methods may be able to encrypt a different number of bytes at once.

## RSA Decryption

Create a small program to decrypt the contents of a file, whose name is provided by the user, using the `RSA` algorithm. The user must also indicate (i) the name of the file containing the private key to use, and (ii) the name for the file to save the decrypted content.

## Different cryptograms?

Run the `RSA` encryption program you developed, to encrypt a file. Run again the program to encrypt the same file, using the same key, and save the encrypted content in another file. Using a binary file editor, open the two encrypted files you just generated and compare them. Are they similar or exactly the same?

Now use the `RSA` decryption program you developed, and decrypt the two encrypted files you generated. What is the result? Are the decrypted files equal to the original file?

**Questions:**

- Can you explain the reason for what you observed in this exercise?

## How to encrypt a very large file?

Has you know, by now, `RSA` encryption is not efficient and is not used to encrypt data bigger than its block size. If you wish to see how inefficient it is, just take some data and measure the time it takes to encrypt and decrypt that data with `RSA` and `AES`.

Imagine you have a large file (500 MBytes, for example) that you want to send to some person, with the guarantee that only that person can decrypt the file. More, you have the public key of that person, and you are not able to contact the person before sending the file. So, you have a big file to transmit to a person, from which you know the public key, but the process will be very slow if you encrypt the file using `RSA` . However, you can use any other encryption algorithm to encrypt the file, but remember you want that only the receiver must be able to decrypt the file.

Can you imagine some combination of encryption technologies that allows you to efficiently send the file to the other person, with the guarantee that only that person can decrypt the file?

A typical solution is called Hybrid Encryption, which combines two ciphers, a symmetric to encrypt the file with a random key, and a asymmetric to encrypt the key used in the previous step. Implement a function that applies this method.

**Questions:**

- With this method, what is sent to the destination?
- Should we always send the pubic key?

2021

**PREVIOUS**
Lab - Symmetric Cryptography

**NEXT**
Lab - Hash Functions

Last updated on 1 Oct 2021

## Related

- Project 2 - Authentication
- Assignment 1 - Vulnerable App
- Assignment 2 - Application Exploitation
- Assignment 3 - Binary Vulnerabilities
- Lab - Hash Functions