CAN304 Lab 2

Learn to Use Crypto Libraries

A cryptography library (e.g., OpenSSL, JCA/JCE, PyCryptodome, Crypto++) is a package which provides cryptographic recipes and primitives to developers. It is also a good source for beginners learning and understanding the principles and applications of primitives.

In this lab, we take the PyCryptodome library as an example and show how to implement security measures using cryptographic primitives.

Dependencies

- Python 3

1. The PyCryptodome

PyCryptodome is a self-contained Python package of low-level cryptographic primitives. It supports Python 2.7, Python 3.5 and newer, and PyPy.

You can install it with:

```
pip install pycryptodome
```

All modules are installed under the Crypto package.

PyCryptodome is not a wrapper to a separate C library like OpenSSL. To the largest possible extent, algorithms are implemented in pure Python. Only the pieces that are extremely critical to performance (e.g. block ciphers) are implemented as C extensions.

For more information, see the following links:

- Homepage: https://www.pycryptodome.org/en/latest/
- GitHub: https://github.com/Legrandin/pycryptodome

2. Encryption and decryption using AES with CBC mode

AES-CBC (cipher block chaining) mode is one of the most used symmetric encryption algorithms. The data size must be nonzero and multiple of the size of a “block” (16 bytes).

Before the encryption is started, add padding to the data so that the length is multiple of the block size. Then split the data into 16-byte blocks. Each block is encrypted as following:

- For block 1, XOR plaintext with a random string called IV (initialization vector), then encrypt the result
- For block i+1, XOR the plaintext with the ciphertext of block i, then encrypt the result.

Decryption is similar.

Practices:

(a) Run aescbcEncrypt.py to encrypt the data. You should see the following:

```
The ciphertext is:
b'\xa7\x8a1\xdf\xf3\xe9\x07\x87\n\xb9\xad\x92!^\xfb\xff\xd5\x90\x84\xa3\xa8\xe5\x8
6\x88Rnm\x15\x8a@\xa8\x15'
>>>
```

(b) Run aescbcDecrypt.py to decrypt the data. You should see the following:

```
The plaintext is:
Learn cryptography with fun!
>>>
```

(c) Exercise encryption and decryption with your own messages.

(d) Exercise data encryption/decryption with other modes

3. Encryption and decryption using RSA with PKCS#1 OAEP

PKCS#1 OAEP is an public-key cipher combining the RSA algorithm and the Optimal Asymmetric Encryption Padding (OAEP) method. It is described in RFC8017 (obsoletes RFC 3447) where it is called RSAES-OAEP. It can only encrypt messages slightly shorter than the RSA modulus (a few hundred bytes).

The RSA algorithm

The RSA algorithm was invented by Ronald L. Rivest, Adi Shamir, and Leonard Adleman in 1977. The security of the algorithm is based on the hardness of factoring a large composite number and computing eth roots modulo a composite number for a specified odd integer e.

An RSA public key consists of a pair (n, e) of integers, where n is the modulus and e is the public exponent. The modulus n is a large composite number (a bit length of at least 1024 is the current recommended size), while the public exponent e is normally a small prime such as 3, 17, or 65537.

An RSA private key may have one of two different representations. Both representations contain information about an integer d satisfying $(x^e)^d \equiv x \pmod{n}$ for all integers x. This means that the private key can be used to solve the equation $x^e \equiv c \pmod{n}$ in x, i.e., compute the eth root of c modulo n.

The RSA encryption primitive RSAEP takes as input the public key (n, e) and a positive integer $m < n$ (a message representative) and returns the integer $c = m^e \bmod n$.

The RSA decryption primitive RSADP takes as input the private key and an integer $c < n$ (a ciphertext representative) to return the integer $m = c^d \bmod n$, where d is an integer with the above specified properties.

RSA encryption with OAEP

```
RSAES-OAEP-ENCRYPT ((n, e), M, L)
   Options:
      Hash     hash function (hLen denotes the length in octets of
               the hash function output)
      MGF      mask generation function
   Input:
      (n, e)   recipient's RSA public key (k denotes the length in
               octets of the RSA modulus n)
      M        message to be encrypted, an octet string of length mLen,
               where mLen <= k - 2hLen - 2
      L        optional label to be associated with the message; the
               default value for L, if L is not provided, is the empty
               string
   Output:
      C        ciphertext, an octet string of length k
   Errors:  "message too long"; "label too long"
   Assumption:  RSA public key (n, e) is valid
```

RSA decryption with OAEP

```
RSAES-OAEP-DECRYPT (K, C, L)
   Options:
      Hash     hash function (hLen denotes the length in octets of
               the hash function output)
      MGF      mask generation function
   Input:
      K        recipient's RSA private key (k denotes the length in
               octets of the RSA modulus n), where k >= 2hLen + 2
      C        ciphertext to be decrypted, an octet string of length k
      L        optional label whose association with the message is to
               be verified; the default value for L, if L is not
               provided, is the empty string
   Output:
      M        message, an octet string of length mLen, where
               mLen <= k - 2hLen - 2
   Error:  "decryption error"
```

Practices:

(a) Run rsaEncrypt.py to encrypt the data.

(b) Run rsaDecrypt.py to decrypt the data. You should see the following:

```
The plaintext is
Learn Cryptography with fun!
>>>
```

(c) Exercise encryption and decryption with your own messages. Note the same message will be encrypted into different ciphertext in different run of rsaEncrypt.py.

4. Homework

Programme using a cryptography library to realize secure communications between Alice and Bob. In addition to confidentiality, integrity should be also achieved. That is, the receiver should be able to detect any unauthorized modification. Besides, each communication session should use a fresh session key to encrypt data.