Seguridad de la Información

Introducción a Pycryptodome

Criptografía Asimétrica (PKC)

Criptografía Asimétrica: RSA

- Existen varios objetos para implementar las funciones de RSA
- **Creación de claves**
  - Crypto.PublicKey.RSA.generate(bits, randfunc=None, e=65537)
  - Parámetros :
    - *bits*: Bits de las claves
    - *randfunc*: Función aleatoria para crear las claves
    - *e*: el exponente público
  - Es necesario indicar el número de bits de las claves

```
def crear_RSAKey():
    key = RSA.generate(2048)

    return key
```

Criptografía Asimétrica: RSA

- **Clave privada**: key
- **Clave pública**: key.publickey()
- **Exportar claves (para guardar en ficheros)**:
  - Privada: key.export_key(format='PEM', passphrase, pkcs, protection)
    - *format*: Formato de la exportación
      - PEM: Fichero de texto (rfc1421/3)
      - DER: Fichero binario
    - *passphrase*: Contraseña con la que proteger la clave
    - *pkcs*: Formato de la exportación
      - pkcs=1 : PKCS#1 (rfc3447). Sólo RSA, sin contraseña
      - pkcs=8 : PKCS#8 (rfc5208). Otros PKC. Contraseña opcional
    - *protection*: Cifrado con el que proteger la contraseña
  - Pública: key.publickey().export_key()
- **Importar claves**: RSA.import_key(objeto fichero)

Criptografía Asimétrica: RSA

```
def guardar_RSAKey_Privada(fichero, RSAKey, password):
    key_cifrada = key.export_key(passphrase=password, pkcs=8,\
        protection="scryptAndAES128-CBC")
    file_out = open(fichero, "wb")
    file_out.write(key_cifrada)
    file_out.close()

def cargar_RSAKey_Privada(fichero, password):
    key_cifrada = open(fichero, "rb").read()
    key = RSA.import_key(key_cifrada, passphrase=password)
    return key

def guardar_RSAKey_Publica(fichero, RSAKey):
    key_pub = key.publickey().export_key()
    file_out = open(fichero, "wb")
    file_out.write(key_pub)
    file_out.close()

def cargar_RSAKey_Publica(fichero):
    keyFile = open(fichero, "rb").read()
    key_pub = RSA.import_key(keyFile)
    return key_pub
```

Criptografía Asimétrica: RSA

- **Cifrado y Descifrado**:
  - Crypto.Cipher.PKCS1_OAEP
    - *engine = PKCS1_OAEP.new(key)*: Crea un engine OAEP
    - *objeto.encrypt(datos)*: Cifra unos datos binarios
    - *objeto.decrypt(datos)*: Descifra un datos binarios
      - El proceso de cifrado sólo se puede hacer con la clave pública del destinatario
      - El tamaño de datos que se puede enviar depende de las funciones y los parámetros subyacentes
        - » P.ej.: Si RSA 2048 y SHA-256, 190 bytes

PKCS#1 OAEP sigue un cifrado asimétrico basado en RSA y padding. Está definido en el RFC8017 conocido con el nombre de RSAES-OAEP

Criptografía Asimétrica: RSA

OJO: La estructura de la clave privada también guarda la clave publica

```
def cifrarRSA_OAEP(cadena, key):
    datos = cadena.encode("utf-8")
    engineRSACifrado = PKCS1_OAEP.new(key)
    cifrado = engineRSACifrado.encrypt(datos)
    return cifrado

def descifrarRSA_OAEP(cifrado, key):
    engineRSADescifrado = PKCS1_OAEP.new(key)
    datos = engineRSADescifrado.decrypt(cifrado)
    cadena = datos.decode("utf-8")
    return cadena
```

Criptografía Asimétrica: RSA

CIFRADO

```
>>> from Crypto.Cipher import PKCS1_OAEP
>>> from Crypto.PublicKey import RSA
>>>
>>> message = b'You can attack now!'
>>> key = RSA.importKey(open('public.pem').read())
>>> cipher = PKCS1_OAEP.new(key)
>>> ciphertext = cipher.encrypt(message)
```

DESCIFRADO

```
>>> key = RSA.importKey(open('private.pem').read())
>>> cipher = PKCS1_OAEP.new(key)
>>> message = cipher.decrypt(ciphertext)
```

Fuente: https://pycryptodome.readthedocs.io/en/latest/src/cipher/oaep.html

Criptografía Asimétrica: RSA

- **Firma y Comprobación**:
  - Crypto.Signature.pss
    - *engine = pss.new(key)*: Crea un engine PSS
      - Para la firma es necesario que key contenga una clave privada, mientras que para la comprobación es solamente necesaria una clave pública
    - *Firma = objeto.sign(hash)*: Realiza una firma del parámetro hash
      - hash es un objeto que contiene una función hash realizada sobre unos datos binarios
    - *objeto.verify(hash, firma)*: Comprueba si el parámetro hash corresponde con la firma adjunta
      - En caso de error, el método lanza una excepción

Criptografía Asimétrica: RSA

```
def firmarRSA_PSS(texto, key_private):
    h = SHA256.new(texto.encode("utf-8"))
    # Ya veremos los hash la semana que viene
    print(h.hexdigest())
    signature = pss.new(key_private).sign(h)
    return signature

def comprobarRSA_PSS(texto, firma, key_public):
    h = SHA256.new(texto.encode("utf-8"))
    # Ya veremos los hash la semana que viene
    print(h.hexdigest())
    verifier = pss.new(key_public)
    try:
        verifier.verify(h, firma)
        return True
    except (ValueError, TypeError):
        return False
```

Bibliografía básica

- “Python 3 documentation”

  https://docs.python.org/3/tutorial/

- PyCryptodome

  https://pycryptodome.readthedocs.io/en/latest/src/util/util.html