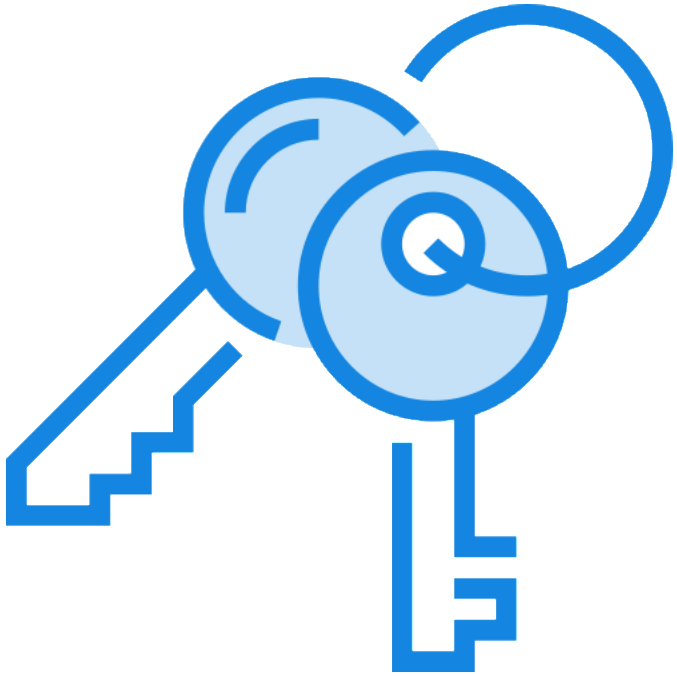
# Seguridad de la Información

## Introducción a Pycryptodome

## Criptografía Asimétrica (PKC)



# GENERAR CLAVES

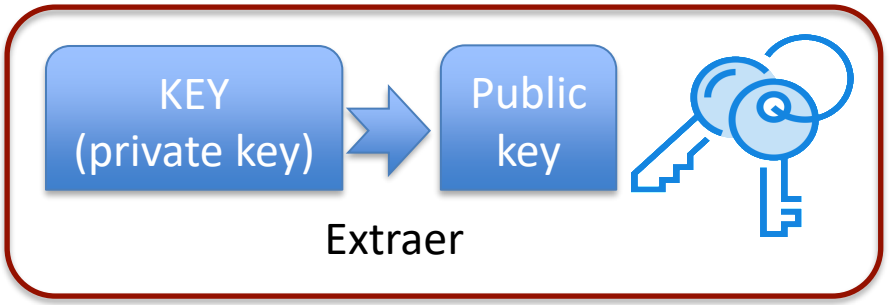
# Generación de Claves RSA

- En el paquete **Crypto.PublicKey**
  - Crypto.PublicKey.RSA.**generate**(bits, randfunc=None, e=65537)
    - Parámetros:
      - ***bits***: longitud (en bits) de las claves. Mínimo 1024
      - ***randfunc***: función aleatoria para crear las claves
      - ***e***: el exponente RSA público
    - Devuelve: un objeto **Rsakey**

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

    return key
```

# Manejo de Claves 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 (formato ASN.1)
    - ***passphrase***: contraseña con la que proteger la clave
    - ***pkcs***: estándar criptográfico
      - 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)

# INCISO: Formatos de exportación, explicación adicional

- **DER (Distinguished Encoding Rule)**: codificación en **binario**, utilizando el formato ASN.1
  - ASN.1: protocolo estandarizado para redes de telecomunicaciones, "similar" a XML (define estructuras de datos)
- **PEM (Privacy-Enhanced Mail)**: codificación en **base64** (texto, RFC1421/3) del format DER
  - Fue diseñado para transportar información de criptografía asimétrica a través del correo electrónico
  - Tiene una **cabecera y un pie** que indican el contenido y su función

# INCISO: Formatos de exportación, explicación adicional

- **DER (Distinguished Encoding Rule)**: codificación en **binario**, utilizando el format ASN.1
  - ASN.1: protocolo estandarizado para redes de telecomunicaciones, "similar" a XML (define estructuras de datos)
- **PEM (Privacy-Enhanced Mail)**: codificación en **base64** (texto, RFC1421/3) del format DER
  - Fue diseñado para transportar información de criptografía asimétrica
  - Tiene una **cabecera y un pie** que indican el contenido y su función
- **PKCS#X (Public Key Cryptography Standards)**:
  - **PKCS#1** *(RFC8017)*: claves públicas y privadas de RSA en formato DER
  - **PKCS#7** *(RFC2315)*: contiene un certificado en formato DER
  - **PKCS#8** *(RFC5958)*: contiene la clave privada en formato PEM. **Puede estar protegido con contraseña**
  - **PKCS#12** *(RFC7292)* : suele contener un certificado y su correspondiente clave privada (en formato DER). **Puede estar protegido con contraseña**

# INCISO: Formatos de exportación, explicación adicional

```
-----BEGIN PRIVATE KEY-----
MIICdgIBADANBgkqhkiG9w0BAQEFAASCAmAwggJcAgEAAoGBAKNwapOQ6rQJHetP
HRlJBIh1OsOsUBiXb3rXXE3xpWAxAha0MH+UPRblOko+5T2JqIb+xKf9Vi3oTM3t
KvffaOPtzKXZauscjq6NGzA3LgeiMy6q19pvkUUOlGYK6+Xfl+B7Xw6+hBMkQuGE
nUS8nkpR5mK4ne7djIyfHFfMu4ptAgMBAAECgYA+s0PPtMq1osG9oi4xoxeAGikf
JB3eMUptP+2DYW7mRibc+ueYKhB9lhcUoKhlQUhL8bUUFVZYakP8xD21thmQqnC4
f63asad0ycteJMLb3r+z26LHuCyOdPg1pyLk3oQ32lVQHBCYathRMcVznxOG16VK
I8BFfstJTaJu0lK/wQJBANYFGusBiZsJQ3utrQMVPpKmloO2++4q1v6ZR4puDQHx
TjLjAIgrkYfwTJBLBRZxec0E7TmuVQ9uJ+wMu/+7zaUCQQDDf2xMnQqYknJoKGq+
oAnyC66UqWC5xAnQS32mlnJ632JXA0pf9pb1SXAYExB1p9Dfqd3VAwQDwBsDDgP6
HD8pAkEA0lscNQZC2TaGtKZk2hXkdcH1SKru/g3vWTkRHxfCAznJUaza1fx0wzdG
GcES1Bdez0tbW4llI5By/skZc2eE3QJAFl6fOskBbGHde3Oce0F+wdZ6XIJhEgCP
iukIcKZoZQzoiMJUoVRrA5gqnmaYDI5uRRl/y57zt6YksR3KcLUIuQJAd242M/WF
6YAZat3q/wEeETeQq1wrooew+8lHl05/Nt0cCpV48RGEhJ83pzBm3mnwHf8lTBJH
x6XroMXsmbnsEw==
-----END PRIVATE KEY-----
```

**PKCS#8 (PEM), sin cifrar**

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
MIIC3TBXBgkqhkiG9w0BBQ0wSjApBgkqhkiG9w0BBQwwHAQIkErtXjGCalMCAggA
MAwGCCqGSIb3DQIJBQAwHQYJYIZIAWUDBAEqBBApOUG3MKrBC/5bDBH/s5VfBIIC
gN5o1aJxvJYbp2oA/quz+BnCFn8ts3wPPOcqarHddy0L/VH3BdqFNbnPZEaDnvDl
kqChRsti4AAeX18ItMeAyNLNFv0J4mfI8Q5E7iEnPp+dTsZqNfVIJe2NGxOS7zp2
oQQIZVgjW0akDehv6ZDN796qDBlMY2g80wbBrzVgMJu/byG9IQQjngUE9QNGwrsj
7lYSprxjfTZOk1aGBD0d/HsmetIJvCeJ2i/5xAiGgZRrSWMC6aN7Zlra3eIvHQTB
aKZ8/0IT3iKSr6FpkEopOQae8biiTEVGw9D339P3qOSs2ChWWF+OV2sEA67w6q5j
pz6Poc5jgq4FOcf06GdcVa4tst2uykNJCW0wHpcUR1Tr9ILLhrZPYBYVQWW53Eee
o4+mqW2YORdG3a/jLHpEjL0Vdg95QNpdZoMv8plotN1MUBLebd05aCe5hJUb/x74
3GTwmRGmKoHOhOO3hhUaMCmZIg1xPlNT3jqxrZDoATBeONbaFP8OOkeucVYHbdUO
Ad7z6J8XuZDoxM0BVrGykEiQL2nAOccdsGoC33C9hjkqgU8G9jWElbynJlVqv+1a
lFHWjX5lB6ueiY/rClpVlLmXGB83OVPlo70FV0B9rhRChyB1IJJRYPFDJHSHJNu+
Pqay8zw82Gh/G+TWH/JCLm5YjX55ZSFMUmvwOLaxyQpmAGNH6dIBTAaSctVA7UrV
jm7m+5T7seiNYNEi19vDJipgr0GbX8+np47VrsJDxsS20wTeA/9ltD0xXwNrEKHd
2Nv/1OaCgnBQHIGULgEn9pT3/vU87bBHYjVdrWoUmqd9zFYtdImQE9u8IKTxTe4R
UPRGHqltz4uOjbD1epkSGe0=
-----END ENCRYPTED PRIVATE KEY-----
```

**PKCS#8 (PEM), cifrado**

# Ejemplo Claves RSA

```
def guardar_RSAKey_Privada(fichero, key, 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, key):
    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
```

# CIFRADO Y DESCIFRADO EN RSA

# Cifrado y Descifrado en RSA

- En el paquete **Crypto.Cipher** tenemos el módulo PKCS#1 OAEP, que utiliza un **cifrado asimétrico basado en RSA y padding**

  - Crypto.Cipher.PKCS1_OAEP

    - *engine* = ***PKCS1_OAEP.new****(key)*: crea un engine OAEP
    - *engine*.***encrypt****(datos)*: cifra datos binarios
    - *engine*.***decrypt****(datos)*: descifra 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

# Ejemplo de cifrado y descifrado

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

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

# FIRMA Y VERIFICACIÓN EN RSA

# Firma y verificación RSA

- En el paquete **Crypto.Signature** tenemos el módulo PKCS#1 PSS, que ofrece un esquema de firma probabilístico basado en RSA
  - 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 verificación es solamente necesaria una clave pública
    - *firma* = *engine****.sign****(****hash****)*: realiza una firma del parámetro hash
      - hash es un objeto que contiene el valor resultante de aplicar una función hash sobre unos datos binarios (p.ej. SHA256)
    - *engine****.verify****(hash,* ***firma****)*: comprueba si la firma es correcta, i.e., si al descifrar la firma se obtiene el mismo hash pasada como argumento
      - Si la verificación falla, el método lanza una excepción

# Ejemplo de firma y verificación

```
def firmarRSA_PSS(texto, key_private):
    # La firma se realiza sobre el hash del texto (h)
    h = SHA256.new(texto.encode("utf-8"))
    print(h.hexdigest())
    signature = pss.new(key_private).sign(h)
    return signature

def comprobarRSA_PSS(texto, firma, key_public):
    # Comprobamos que la firma coincide con el hash (h)
    h = SHA256.new(texto.encode("utf-8"))
    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