Written lab report

Author: Hmon Wutt

Introduction

The purpose of the lab is to explore asymmetric and symmetric encryption, message authentication and digital signatures to verify sender identity.

Implementation

Task 1 & 2

1. Split the “python_ciphertext_enc.bin” file into 4 parts.

```
with open("python_ciphertext_enc.bin", "rb") as f:
    enc_key = f.read(256)
    enc_iv = f.read(256)
    enc_hmac = f.read(256)
    ciphertext = f.read()

open("enc_key.bin", "wb").write(enc_key)
open("enc_iv.bin", "wb").write(enc_iv)
open("enc_hmac.bin", "wb").write(enc_hmac)
open("ciphertext_data.bin", "wb").write(ciphertext)
```

2. Retrieve receiver’s password-protected private key

```
with open("receiver_private_key.pem", "rb") as f:
    private_key = serialization.load_pem_private_key(
        f.read(),
        password=b"lab1enckeys"
    )
```

3. Use private key to decrypt AES key, Initiation vector(IV) and MAC

```
def rsa_decrypt(data, key):
    return key.decrypt(
        data,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA1()),
            algorithm=hashes.SHA1(),
            label=None
        )
    )


aes_key = rsa_decrypt(enc_key, private_key)
iv = rsa_decrypt(enc_iv, private_key)
hmac_key = rsa_decrypt(enc_hmac, private_key)
```

4. Use AES key and IV to decrypt the actual message.

```
cipher = Cipher(
    algorithms.AES(aes_key),
    modes.CBC(iv)
)

decryptor = cipher.decryptor()
padded_plaintext = decryptor.update(ciphertext) + decryptor.finalize()

unpadder = sym_padding.PKCS7(128).unpadder()
plaintext = unpadder.update(padded_plaintext) + unpadder.finalize()

open("plaintext.txt", "wb").write(plaintext)
```

5. Result

```
[hmon@Hmon Python files]$ cat plaintext.txt
Slip inside the eye of your mind
Don't you know you might find
A better place to play
You said that you'd never been
But all the things that you've seen
Will slowly fade away

So I'll start a revolution from my bed
Cos you said the brains I had went to my head
Step outside. Summertime's in bloom
Stand up beside the fireplace
Take that look from off your face
You ain't ever gonna burn my heart out

So Sally can wait, she knows its too late as we're walking on by
Her soul slides away, but don't look back in anger I heard you say

Take me to the place where you go
Where nobody knows if it's night or day
Please don't put your life in the hands
Of a Rock n Roll band
Who'll throw it all away

Gonna start a revolution from my bed
Cos you said the brains I had went to my head
Step outside the summertime's in bloom
Stand up beside the fireplace
Take that look from off your face
Cos you ain't ever gonna burn my heart out

So Sally can wait, she knows its too late as she's walking on by
Her soul slides away, but don't look back in anger I heard you say

So Sally can wait, she knows its too late as we're walking on by
Her soul slides away, but don't look back in anger I heard you say

So Sally can wait, she knows its too late as she's walking on by
My soul slides away, but don't look back in anger
Don't look back in anger
I heard you say
```

6. **Open question**: Why is it not a good idea to simply encrypt the plaintext with the receiver's public key? Why bother to generate Key1, IV, and encrypt them?

   Since asymmetric decryption is computationally expensive, the bulk of the data or the message is symmetrically encrypted using AES key and IV is used to locate the starting block of the message.

Task 3

1. Extract and print md5-hashed message authentication.

```
mac = hmac.new(hmac_key, plaintext, hashlib.md5).digest()
print(mac.hex())
```

2. Result

```
[hmon@Hmon Python files]$ python decryption.py
68d8aaef41c9c43ec483144cb8349800
```

   "Python_mac_1.txt" contains the correct mac.

Task 4

1. To verify the sender's digital signature, we extract out the sender's public key from its certificate. Then, the public key is used to check each digital signature.

```
with open("sender_certificate.pem", "rb") as f:
    cert = x509.load_pem_x509_certificate(f.read())

public_key = cert.public_key()
with open("sender_signature_1.bin", "rb") as f:
    signature = f.read()

try:
    public_key.verify(      Cannot access attribute "verify"
        signature,
        plaintext,
        padding.PKCS1v15(),      Expected 2 positional argumen
        hashes.SHA1()      Expected 3 positional arguments
    )
    print("Signature is valid")
except BaseException:
    print("Signature is invalid")
```

2. Result
   “Sender_signature_2.bin” contains the correct signature.

3. **Open question:** If the receiver does not get the digital signature, does a correct message authentication code authenticate the sender? Why?
   MAC only verifies that the message has not been tampered with during the transit. A correct MAC does not authenticate the sender. Sender’s signature identifies the sender.

   Source code is found in the “decryption.py” file.