Lab 6

(Due your class day of July 7/12)

1. Compute MD5 hash using command and program both.

a. md5sum to hash some message. **Ex**. $echo -n “Harry Porter” | md5sum

b. Use hash_comp.py to comp MD5 on the same message. Take a screen shot of their results.

2. Use openssl to generate RSA public/private key

We can generate RSA private key (p, q, d) using openssl:

$ openssl genrsa –aes128 -out private.pem 1024

This will generate a rsa instance (p, q, d, e, n) with p, q of 1024 bits and to prevent leaking the private key, the output private.pem is encrypted by aes128 cipher with password you will be prompted to provide. Now use the above command to generate a rsa private key and save it in file private.pem. Then, extract the public key (e, n) in a file public.pem:

$ openssl rsa –in private.pem –pubout >public.pem

You can display private key using

$openssl rsa –in private.pem –text -noout

You also can display public key using

$openssl rsa –in public.pem –pubin –text -noout

Take screen for the displays for these two files, as evidence of your work**.**

3. In this problem, you need to practice RSA encryption and decryption.

a. Encrypt messages using PKCS1_OAEP, which is an implementation of RSA. Use the key **RsaKey** derived above to do the encryption. The functions are described as follow.

- Cipher**=PKCS1_OAEP.new**(RsaKey):
  - For the encryption, RsaKey is a public-key. Return an encryption object **Cipher**.
- Cipher.**encrypt**(message):
  - This returns ciphertext of message (byte string) under encryption object **Cipher**.

Encrypt message=**’your name and ID’** and save ciphertext into a file. Take a screen shot for hexdump of your ciphertext (**$hexdump -C filename**).   Ref. **encrypt_RSA.py**.

b. Decrypt the ciphertext in (a). The functions are described as follow.

- Cipher=**PKCS1_OAEP.new**(RsaKey):
    - For the decryption, RsaKey is a private-key. Return an decryption object **Cipher**.
- Cipher.**decrypt**(ctxt):
    - This returns message=**'your name and ID'** under decryption object **Cipher**.

Take a screen shot for your decryption. Ref. **decrypt_RSA.py**.

4. **(optional)** In this problem, you practice RSA signature: generation and verification.

a. Generate RSA based signature. The functions are described as follows.

- Signer**=pss.new(**RsaKey):
    - This defines a signing object *signer* with RsaKey (imported from your RSA private key file).
- Signer.**sign**(*hashedmessage*):
    - This generates the RSA signature of the hashed message. Here you can use SHA512 to generate the hash value of your message.

M = I owe you $2000. Change $2000 to $3000 and sign the modified message. Compare both signatures. Are they similar? Save your signature into a file. Take a screen shot for your file content (using hexdump). Ref. **sign_RSA.py**

b. Verify the signature in (a). The functions are described as follows.

- Signer**=pss.new(**RsaKey):
    - This defines a signing object *signer* with RsaKey (imported from your RSA public key file).
- Signer.**verify**(*hashedmessage*, *signature*):
    - This verifies if *signature* is consistent with the *hashed message*.

Take a screen shot for the output result. Ref. **verify_RSA.py**

5. In this problem, you practice the hybrid encryption, following the steps below:

a. generate 32-byte string **sk** (using get_random_bytes())

b. encrypt **sk** using the code in problem 3

c. use **sk** as a secret key of AES to crypt the following message:

"I find the solution for P not equal NP"

d. use your RSA private key file and ciphertext in (b)(c) to the above message.

Show your complete code in one python file. Print out sk, ciphertext in b and c and the decrypted result for b and c. Ref. **endec_AES.py**