Secure Programming, Assignment 2:

Implementing a Secure Vault

Strict deadline: November 26, 2018, 23:59 Amsterdam time

Use Python with the PyCrypto module to implement a *secure vault* that allows you to encrypt and sign files on your computer. To implement your secure vault, make a Python 2 tool named vault.py that offers the functionality described below.

For more information about PyCrypto, see:
https://pypi.org/project/pycrypto/
https://www.dlitz.net/software/pycrypto/api/current/
https://www.laurentluce.com/posts/python-and-cryptography-with-pycrypto/

1. Document signing Your tool must allow signing documents for integrity checking and authentication. To do this, take a filename and an RSA private key as input, then hash the file with SHA-256 and encrypt (sign) the hash with RSA. Print the resulting signature to stdout. Your tool must also allow the user to verify file signatures. The expected command line interface is as follows:

```
./vault.py -s <path-to-file> <path-to-private-key>
```

This should hash the file with SHA-256, use the given private key to sign the hash, and write the signature to stdout in a format of your choosing.

```
./vault.py -v <path-to-file> <path-to-public-key> <path-to-signature>
```

This should use the given public key to verify the signature of the given file. If the signature is correct, the program should exit with status code 0. If the signature is wrong, it should exit with status code 1. You don't need to print anything to screen. You can assume that the signature is in the same format that you used to print the signature to screen when signing.

Note: See https://www.laurentluce.com/posts/python-and-cryptography-with-pycrypto/ to learn how to generate a public/private keypair with PyCrypto. You may create a separate Python program to generate keys, but you don't need to hand it in as part of the assignment.

2. Document encryption Your tool must support the use of AES 256 in CBC mode to encrypt/decrypt files for confidentiality. This part of your tool must have the following command line interface:

```
./vault.py -e <path-to-file> <secret-key> <iv>
```

This encrypts the given file with the given secret key and initialization vector (both in hexadecimal format, e.g. 0xdeadbeef) and prints the encrypted file to stdout as *raw binary* (e.g., without any encoding like hexadecimal or Base64). You can save the encrypted file by redirecting stdout to a file.

```
./vault.py -d <path-to-file> <secret-key> <iv>
```

This decrypts the given file with the given secret key and IV and prints the result to screen (raw).

Submit A zip file on Canvas named <your vunet id>.zip that contains your implementation of vault.py. Your Python source should contain clear comments describing your implementation decisions. Your program must use the exact input/output interface described above. Please include your name, student number, Vunet ID, and e-mail address in a comment at the top of the file.