Project 2: Applied Cryptography

CMSC 426/626 — Fall 2014

Summary

Gru and Dr. Nefario need a better encryption program. As Gru's favorite minion, you've been tasked to develop the new software. The systems is to use AES to encrypt the data and RSA to protect the AES keys. You will need to write one program to generate RSA keys, and a second that uses the RSA keys to perform the actual encryption or decryption of messages.

Project Details

To complete the project, you will need to write two Python programs:

`genkeys.py` — generate RSA public and private keys.

a. The program takes a single command line argument, the name of the user for which keys are being generated (referred to as `name` in the following). For test purposes, use the user names `gru` and `drn`.

b. The program must be runnable directly from the command shell, e.g. `./genkeys.py gru`

c. The program must randomly generate an RSA public / private key pair using code that you write (you *cannot* import RSA code from another module such as PyCrypto). It must use `random.SystemRandom` or `os.urandom()` as the source of pseudo-random bytes. The keys must be of cryptographic size.

d. The program must produce two output files, one containing the RSA private key (`name.prv`) and the other containing the RSA public key (`name.pub`). The format of the key files is up to you.

e. You will be provided with code to compute modular inverses and to test whether a number is prime.

`crypt.py` — encrypt and decrypt data using AES-128 and RSA.

a. The program takes four command line arguments: a single flag (`-e` or `-d`) indicating whether the program is being used to encrypt or decrypt a message, the name of the public or private key file to use (generated by `keygen.py`), the name of the file to encrypt or decrypt, and the name of the output file. For example, the following command will encrypt the file `secret.txt` using Gru's public key file `gru.pub` to produce the cipher text file `secret.cip`:

```
./crypt.py -e gru.pub secret.txt secret.cip
```

Then the following command would decrypt the file `secret.cip`:

```
./crypt.py -d gru.prv secret.cip secret.txt
```

b. To encrypt a file, the program must generate a random key *K* for AES-128 using `random.SystemRandom` or `os.urandom()`, use the key *K* with AES-128 to encrypt the data from the input file, use RSA with the public key file specified on the command line to encrypt *K* (we refer to the encrypted *K* as *K'* in the following), and write the encrypted data and *K'* to the output file. The format of the output file (how to store *K'* along with the encrypted data) is your choice.

c. To decrypt a file, the program must read the encrypted data and *K'* from the input file, RSA-decrypt *K'* to recover the key *K,* use *K* with AES-128 to decrypt the data, and write the decrypted data to the output file.

d. You must write the RSA code; you may *not* import RSA code from another module such as PyCrypto.

e. You must choose an appropriate mode of operation for AES-128.

In addition to the two Python programs, you must provide a written description of the design of your programs and a screen capture of a session demonstrating that your programs work. For example, a screen capture of the following sequence of commands would be sufficient (this assumes the files `message.txt` and `message2.txt` already exist):

```
./genkeys.py gru
./genkeys.py drn
cat message.txt
./crypt.py -e drn.pub message.txt message.cip
cat message.cip
./crypt.py -d drn.prv message.cip message.txt
cat message.txt
cat message2.txt
./crypt -e gru.pub message2.txt message2.cip
cat message2.cip
./crypt -d gru.prv message2.cip message2.txt
cat message2.txt
```

Provided Files

The file `proj2.py` contains functions to compute the modular inverse and to test whether an integer is a prime:

`modinv(a, m)` computes the inverse of `a` mod `m`.

`is_probable_prime(p)` returns `True` if `p` is a probable prime; `False` otherwise.

The file also includes the function `egcd(x, y)` which computes the gcd of `x` and `y` using the extended Euclidean algorithm. This function is used by `modinv()`.

Project Submission

The project will be submitted in two phases: a milestone and a final submission. Both the milestone and final submission are to be turned-in on Blackboard.

Milestone - Due Tuesday, November 18.

The Milestone submission will consist of one file: `genkeys.py`.

Final Submission - Due Tuesday, November 25.

The final submission will consist of three files:

1. A document (DOC, DOCX, or PDF) providing a brief description of the design of your programs and including a screen capture of the working programs as described above.
2. The program `genkeys.py`.
3. The program `crypt.py`.