CSCI 5271 : Introduction to Computer Security

Programming Assignment 3: File Locker **Due: 11:59:59 pm December 4, 2020**

Ground Rules. You may choose to work with up to two other student if you wish. Only one submission is required per group, please ensure that both group members names are on the submitted copy. Work must be submitted electronically via canvas. The choice of programming language is yours, but your software will be expected to operate in an environment of my choosing, specifically an arch linux virtual machine. As per usual, you are expected to have a file named groupMembers.txt in the root of your archive.

Projects will be graded by unzipping the archive in the home directory of "theboss". After that your Makefile will be invoked with sudo make. The result of running make should be the correct executables in the current working directory and any dependencies installed. You are allowed to use all functionality in your crypto library of choice. Java natively comes with a crypto provider, pycrypto and openssl are installed on the virtual machine already. Any other crypto library should be installed during the invocation of make.

1. "Certificate" Generation

The first thing we will need are public keys. We are **not** going to require full X.509 certificates (the standard of certificates) for this programming assignment. Instead, we will be doing a simple "certificate" that contains only three things: the subject of the certificate, the public key data, and the public key algorithm. The internal mechanics of how you represent this in the certificate file are left to your discretion. Note, you will also need to save private keys for later use, they can be saved in a similar maner to public keys, but in a seperately named file. You are expected to generate RSA key pairs, which will be used for encrypting/decrypting the key file as well as signing the keyfile. You are allowed to use any cryptography library to generate these keys. Please generate 2048 bit RSA keys.

Your build should generate one program: **keygen** which generates a public key/private key pair for a subject and writes them to files. It should take the following arguments.

- -s `<subject>` The subject of the key pair, will just be an alphanumeric string.
- -pub `<public key file>` The path to where I want the public key "cert" writen to.
- -priv `<private key file>` The path to where I want the private key "cert" writen to.

2. Directory Locking

We are going to be implementing a *highly* simplified encrypted file system. The basic idea is two users on a shared File System want to leave encrypted files for each other. To pull this all together, I want you to build a simple "folder locker". In "lock" mode it will take an existing directory, encrypt all files using a symmetric key and generate MAC tags for each file. In "unlock" mode it will take an existing "locked" directory and unencrypt all of the files in addition to verifying the MACs on all files. I will expect directory structure to be preserved. In otherwords, if a file exists inside a sub-folder of the given directory, the cipher text should exist in the same subdriectory. After unlocking the file should appear in the correct sub-directory. This encryption and MACing should be done using AES-GCM, from a crypto library of your choice. AES-GCM is a symmetric key mode that generates a cipher text and a tag at the same time.

Of course the problem is you do not have a shared secret with the person who will be consuming the locked directory. You do have access to their public key generated from part 1. We will use public key crypto to bootstrap a shared key used for operations on the files. Specifically, you will encrypt the AES key used into a keyfile, using the recipeint party's public key. You should then sign that file with your private key. This keyfile should exist at the root of the encrypted directory using the name `keyfile`, the signature on that file should exist in the file `keyfile.sig`.

Your build should generate two programs:

- **lock** Which will encrypt a directory so that it can only be read by the person who holds the private key matching a particular public key.
- **unlock** Which will decrypt a directory using a particular private key.

Your programs should have the following flags:

- -d `<directory to lock/unlock>` The directory to lock or unlock.
- -p `<action public key>` The path to the public key for the locking party in **unlock** mode (the one used to verify keyfile), and the public key of the unlocking party in **lock** mode (the one used to encrypt the keyfile).
- -r `<action private key>` The path to the private key that can decrypt the keyfile in **unlock** mode or that will be used to sign the keyfile in **lock** mode.
- -s `<the action subject>` For **lock** this is the subject you want to encrypt the directory for, for **unlock** the subject you expect the directory to be from.

Your program in "lock" mode should do the following, if any step does not verify, abort:

- Validate that the subject in the action public key file matches the subject given in the -s argument.
- Generate a random AES key for encryption and tagging, encrypt that key with the unlocking party's public key, write that cipher text to a file called `keyfile`.
- Sign the keyfile with the locker's private key, write that signature to a file called `keyfile.sig`.
- Encrypt all files in the given directory using AES in CBC-GCM mode, replacing the plain text files with the cipher text files.

Your program in "unlock" mode should do the following, if any step does not verify, abort:

- Validate that the subject in the action public key file matches the subject given in the -s argument.
- Verify the integrity of the keyfile using the locking party's public key and keyfile.sig.
- Fetch the AES key from keyfile.
- Delete keyfile and keyfile.sig at this point.
- Decrypt the encrypted files in the directory, replacing the cipher text files with the plain text files.