Final Assignment: Security Copy Utility

CS52800 – Network Security
Spring 2024 – Prof. Dave Tian

Due date: 12 AM EDT, April 30th, 2024

In this assignment, you will create a file encryption/decryption/transmission suite similar to the UNIX *scp* utility, using `gcrypt` libraries provided by the Linux operating system. Your application will use industry-standard cryptographic algorithms and libraries. `gcrypt` is used to build GnuPG and can be used to develop ssh and other secure programs - it and OpenSSL are used extensively in Linux distributions.

1 Assignment Details

1. The programs are to be written in C and will use the `libgcrypt` library. The `make` utility should be used to create the program.
2. The file encryption programs *purenc* and *purdec* should take the following inputs:

   ```
   purenc <input file> [-d <output IP-addr:port>] [-l]
   purdec [-l <input file>] <port>
   ```

   where *purenc* takes an input file and transmits it to the IP address/port specified on the command line (`-d` option) or dumps the encrypted contents of the input file to an output file of the same name, but with the added extension "`.pur`". For example, if the input file is `hello.txt`, the output file should be `hello.txt.pur`. *purdec* should run as a network daemon, awaiting incoming network connections on the command-line specified network port. *purdec* can also be run in local mode (`-l` option) in which it bypasses the network functionality and simply decrypts a file specified as input. In either mode, the output should be the original name of the file, minus the `.pur` extension.
3. On each invocation, *purenc* and *purdec* should prompt the user for a password to encrypt or decrypt the file under. The key used to encrypt the file should be computed from the password by hashing it using the PBKDF2 function. You should also attach an HMAC to the file and verify the HMAC with *purdec*. Encryption must be done using AES256.
4. Both *purenc* and *purdec* should display an error and abort the operation if the output file already exists.
5. When both [-l] and [-d] arguments are provided for *purenc*, you need to save the ciphertext locally and send it to the remote daemon. For *purdec*, only one argument should be provided once, either [-l] or ¡port¿, not both.
6. Since salt is needed for decryption, you need to save it into the file if running the local mode, or send it over to *purdec* if running the remote mode. The format of salt+ciphertext is left to your own decision, as long as your *purdec* can recover the plaintext correctly. E.g., the output of *purdec* should be just the original plaintext without salt.

2 Developing Your Code

You will need to run Linux in order to perform this assignment. If you do not have a machine running Linux that you have root access on (in order to potentially install libraries as necessary), you should use a virtual machine. This also has the benefit that if you crash the system, it is straightforward to restart, and allows you to take snapshots of system state right before you do something potentially risky or hazardous, so that if something goes horribly awry you can easily roll back to a safe state.

3 Submission Instructions

You **must** turn in the following:

1. A `README` file. The `README` will contain the following:
   - Your name

    - A list of submitted source files
    - Overview of work accomplished
    - Description of code and code layout
    - General comments and design decisions made, as well as anything else that can help us grade your code.
    - An answer to the following question: There will be a particular decision you'll need to make for dealing with PBKDF2. What extra input does it require and why is this used? How does your program deal with it?
    - Number of hours spent on the project and level of effort required.

2. A `Makefile`. We should be able to compile your code by simply typing `make`. If you do not know what a `Makefile` is or how to write one, consult one of the many online tutorials.

3. The source code for your programs. These, along with your Makefile, should be well-commented.

Put all of your project files into their subdirectory called `final` and make sure that your `Makefile` has a "make clean" command to clear out any object files and other unneeded intermediate files. Run "make clean" on this subdirectory.

cd up a directory so that `final` is a subdirectory of your working directory, and run the following command to create your submission tarball, but replacing "PURDUEEMAIL" with your purdue.edu email account name.

```
tar cvzf final_submission_PURDUEEMAIL.tar.gz final
```

For example, since my purdue email account is "daveti", I would run the command:

```
tar cvzf final_submission_daveti.tar.gz final
```

Please submit your tarball using the BrightSpace.

4 Grading Guidelines

This programming assignment will be graded as follows:

- 10% Documentation
- 20% Proper use of APIs and general code design
- 70% Functionality and correct operation of your shell

We should be able to run the following commands after retrieving your tarball:

```
tar xvzf <final_submission_PURDUEEMAIL>.tar.gz
cd <PURDUEEMAIL>-final
make
cp README* testfile
./purenc testfile -l
rm testfile
./purdec testfile.pur -l
diff testfile README*
```

Warning: If we cannot do these minimal operations successfully, you will not receive a passing grade. This is not all that we will test–we have another set of tests that we will do when grading, including carefully reviewing your code.

Note that general deductions may occur for a variety of programming errors including memory violations, lack of error checking, poor code organization, etc. Also, do not take the documentation lightly, as it provides those evaluating your project with a general roadmap of your code; without good documentation, it can be quite difficult to grade the implementation. These assignments will be be graded on machines running the **Ubuntu 18.04** Linux distribution.

Late penalty applies as usual – points will be deducted at 20% per day.

5 Note on Academic Dishonesty

As with all assignments in this class, you are prohibited from copying any content from the Internet. You are also prohibited from sharing code, configurations, or text, or getting help with the basics of the assignment (using the library and developing your code) from anyone apart from the instructor and the TA. This is an unfortunate necessity to cut down on cheating. Consulting online sources is acceptable, but under no circumstances should *anything* be copied.

We will be using some tools to find similarities between the submitted project and those submitted by others in the class, so please do not be tempted to cheat.

Failure to abide by these requirements will result in academic sanctions up to dismissal from the class and involvement of Academic Affairs.

6 Example Operation

An example session using these tools may look like:

```
daveti-A% ./purenc testfile -d 192.168.1.48:8888
Password: hello
read 1024 bytes, wrote bytes 1040,
read 1024 bytes, wrote bytes 1040
read 1024 bytes, wrote bytes 1040
read 1024 bytes, wrote bytes 1040
read 1024 bytes, wrote bytes 1040
read 1024 bytes, wrote bytes 1040
read 256 bytes, wrote bytes 272
Successfully encrypted testfile to testfile.pur(6512 bytes written).
Transmitting to 192.168.1.42:8888
Successfully received

-----

daveti-B% ./purdec 8888
Waiting for connections.
Inbound file. Password: hello
read 1040 bytes, wrote bytes 1024
read 1040 bytes, wrote bytes 1024
read 1040 bytes, wrote bytes 1024
read 1040 bytes, wrote bytes 1024
read 1040 bytes, wrote bytes 1024
read 1040 bytes, wrote bytes 1024
read 272 bytes, wrote bytes 256
Successfully received and decrypted testfile.pur to testfile (6400 bytes written).
```

7 Acknowledgement

Special thanks go to Prof. Kevin Butler at the University of Florida for making this assignment.