

# CS4471 Computer and Network Security

Fall 2008

## Project 3: Group Access to Encrypted Files

### 1 Overview

Often a number of users require access to one or more encrypted files. Several implementation options are possible. One option is to give a passphrase (or key) to all users of a particular file. However, this can become problematic when many files are being protected and the group of users accessing any given file differs. In the worst case, a unique passphrase is required for each file and a single user must remember a passphrase for each of these files. It would be far easier (and more secure) to require users to remember only a single passphrase. This could be achieved by encrypting each file using the passphrase of each user allowed to access the file. Hence, there would be as many encrypted copies of a given file as there are users allowed to view the contents of that file. However, this option uses disk space inefficiently and incurs significant overhead in keeping the copies consistent. A third approach is to generate a unique key for each file, and then encrypt this file key (as opposed to the entire file) using the passphrase of each user. This eliminates the need for users to remember a unique passphrase for each file, and eliminates the need to keep multiple encrypted copies of the file. In this project, you will implement this third scheme for group access to encrypted files.

### 2 Requirements

Files are protected and accessed via four commands: `groupsecure`, `addsecure`, `rmsecure`, and `getsecure`. These commands, respectively, initialize a file for application of the group encrypted access scheme, grant access for a file to a particular user, remove access for a file from a particular user, and get the clear text for a protected file. Each of the commands is described in more detail below.

#### 2.1 Initialization

The command `groupsecure file` performs initialization to apply the group scheme to `file`. It is a fatal error if the user executing the command is not owner of the file or if the file is not an ordinary file. The following actions are performed. A 448 bit key is generated using `/dev/urandom`. The contents of `file` are encrypted using the Blowfish symmetric encryption algorithm and the encrypted data is written to `file.enc` in the same directory as `file`. The 448 bit key is encrypted using the GPG public key for the file owner and the encrypted key is written to a key file. It is a fatal error if the owner's GPG key is not accessible in the file `pubring.gpg` within the directory `.gnupg` of the owner's home directory. Assume (test and fail) the `userid` associated with the owner is sufficient to recover the appropriate key. Output a message to the user that indicates which key is being used to encrypt the file key. The key file is written in the same directory as `file.enc`. It is up to the owner of `file` to delete it (the clear text) after this operation. The "key file" and `file.enc` should have group read/write access so that `setuid` programming is not necessary.

## 2.2 Adding and Removing Users

The command `addsecure(rmsecure) userid file grants(removes) access for userid to file` (or more specifically, `file.enc`). It is a fatal error if the user executing the command is not the owner of the file or if the command `groupsecure` was not previously executed against the file. You may assume the existence of `file.enc` and an appropriate key file is sufficient to determine the appropriate initialization has been performed. The following actions are executed when a user is added. The symmetric key for `file` is obtained from the key file using the GPG private key for the file owner. A copy of this key is encrypted with the GPG public key for `userid`. It is an error if the public key for `userid` is not on the GPG keyring of the file owner. The public key encrypted symmetric key is written to a key file. When a user is removed, the public key encrypted symmetric key for `userid` is removed from the key file. The “key file” may be comprised of more than one physical file.

Since the “key file” has group read/write access, a user could copy his encrypted version of the key used to create `file.enc` and later use that key to access the file even if the file owner uses `rmsecure` to remove access. Your project must prohibit decrypting `file.enc` from a user whose access has been removed. You may not use `setuid` to do this.

## 2.3 Accessing Protected Files

The command `getsecure file` attempts access to the contents of file `file.enc` by the user executing the command. The symmetric key encrypted with the public key for the user is recovered from the key file. Assume (test and fail) the `userid` associated with the real `uid` is sufficient to recover the appropriate key. It is a fatal error if the user does not have a key in the key file. The symmetric key is recovered and used to decrypt `file.enc`. The unencrypted file contents are written to `stdout`.

## 3 Collaboration Rules

This project is to be performed in groups of two or individually. Of course, there are no restrictions on interactions among group members. However, each group must work independently. A group may neither show any else its code nor look at the code of another group. (This policy extends to any external resource, including code found on the web or individuals who are not enrolled in the course.) You may have anyone help you to test your code.

## 4 Submissions

You must prepare a `Makefile` and all necessary source files so that I can simply do a `make` and build the required four command binaries. To that end, create a directory called `gaef.d` in which your `Makefile` and all required source files will reside. Make `gaef.d` your working directory and tar the contents of the directory with “`tar -zcvf gaef.tgz *`”. Use the submission rules discussed previously to “submit” this file via Blackboard. Each group must send me their names by Friday, Nov. 14 at 1pm. No late work will be accepted after 5:00pm on December 12. Include comments in your code at points where security mechanisms are implemented. Include a `README` that gives an overview of your implementation and identifies and defends any security-related decisions you had to make during the implementation.