Assignment 2: Secure Copy Utility

Narayana Ravikumar

January 23, 2013

1 Source Files

1.1 Encryption

- encrypt.h
- ncsuenc.c
- Makefile
- encryptfunction.c
- fileoperations.c
- generateky.c
- hmac.c
- transferfunction.c

1.2 Decryption

- decrypt.h
- ncsudec.c
- Makefile
- decryptfunction.c
- fileoperations.c
- generateky.c
- hmac.c
- receivefunction.c
- hashverify.h

2 Overview

All the project requirements has been completed. The project implements AES 256 for data encryption, PBKDF2 for deriving key, SHA512 to calculate the HMAC and libgcrypt random byte generator for 'salt' and 'initialization vector'(IV) generation. The code first reads the file, whose filename is provided. The password is accepted from the user. A 32byte key is derived by using the password and 4 bytes of randomly generated 'salt'. This key is then used to encrypt the plaintext in 'cipher block chaining'(CBC) mode with flags to provide automatic padding. The 'Hash Based Message Authentication Code' (HMAC) is calculated using the cipher text and the key. A file with suffix '.ncsu' is created and the following are attached in order of salt, IV, HMAC, ciphertext. If the outgoing address is specified then the data is sent to the address and port specified, else the file written in the same directory.

During decryption the same process is performed with additional HMAC verification. Initially the options are checked to determine if a local file is going to be decrypted or should the server be started to get a remote file. Depending on the option the appropriate action is performed. If '-i' option is selected then the specified file is read from the local directory and decryption is performed. If '-d' is selected then a server listening at the port specified is started. When the file is determined, first the 'salt' value is extracted from the file, then the IV, then attached HMAC and finally the ciphertext. A password is requested from the user and the key is derived from it. It is then hashed with the ciphertext to calculate the HMAC. This calculated HMAC is compared with the attached HMAC and thus performing the integrity check. If the integrity check is cleared then the the code is decrypted using the generated key.

3 Description of the code

The code is completely modularised with all essential functionalities written in different files and having comprehend able function names. The project is comprised of two folders 'encryption' which contains all the files and Makefile related to encryption and 'decryption' which contains all the files and Makefile related to decryption. There are separate files for each of the following functions encryption/decryption, file operations, key generation, HMAC calculation, HMAC verification, transfer/receive data and main functions. Most error cases and input validation is performed else an error message is displayed. The code is well commented and provides correct notifications at appropriate times.

4 Design Decisions

- CBC mode is used with cipher text stealing so that CBC works with arbitrary size of input data.
- Used 4 bytes of salt to protect against rainbow table attacks which is chosen randomly using libgcrypt apis.
- Used randomly generated IV which is unique for each encryption-decryption cycle.
- Used SHA512 for generating 64bytes of HMAC which should provide sufficient integrity check.
- During encryption the file is formatted as 4bytes of salt, 16 bytes of IV, 64 bytes of HMAC and rest is cipher text. Proper file parsing is done at the receiving side to extract the relevant sections of the file.
- TCP is used to transfer the data between client and server.
- Facility for server to reuse port numbers is provided in case the connection is aborted.
- Extensively referred Libgcrypt manual for usage of APIs and Beej's guide to network programming for establishing network utilities.

5 Answer to Question

The additional input required by PBKDF2 is called 'salt'. I have used 4 bytes of random salt for each encryption-decryption cycle. 'Salt' input is used to make rainbow table attacks infeasible. Since hash algorithm is publicly known, it is possible to create a rainbow table with alternating hash-reduce chains and then backtrack to the password matching the hash-reduce chain of the required hash value. Using 'salt' makes such an attack infeasible.

6 Hours and Effort

More hours were spent understanding how algorithms work and debugging than implementing because each API worked in a specific way and using it was much easier than making it work. Implementing and testing took 4 full days which included a weekend.