

Information Security Programming Assignment 1

Dr. Edwin Sha

Due Date: **1pm, Feb. 21, 2012**

This is an interesting programming project. The key issues is how efficient your program is. This project can be done by a group of at most two students. This project must be developed in the school's SUN Unix Environment.

TA will test your programs. It is YOUR responsibility to make your programs workable and runnable by others. Some deduction of points will be given if TA does not know how to run your programs. If you have questions, please ask TA first.

You should submit a **HARD COPY** of a report containing your programs, **AND** an **ELECTRONIC COPY** of your programs before the due date (read how to submit your electronic copy in the end). **Each team please just turn in one copy.**

PUT DOWN YOUR and YOUR PARTNER's EMAIL ADDRESSES IN YOUR REPORT.

Programming Assignment: (120 points + 30 possible bonus points)
(20 points for programming; 40 for credit cases; 60 points for t0, t1; 30 points for longest)

I have several encrypted files. You are asked to find the original text files by trying to find the **KEYS** that were used to encrypt them. Try to be a hacker to finish the project. **FIRST**, login to any school SUN workstation such as apache.utdallas.edu and copy the whole directory to your **UNIX** account:

```
cp -rf /people/cs/e/edsha/security/GR2012S/ass1 .
```

Totally, there are encrypted files: credit0_e, credit1_e, credit2_e, t0_e, t1_e and longest_e.

You are asked to find the keys that can correctly decrypt those files. Lest start with simple credit files representing credit card numbers.

(1) You can find encrypted files: credit0_e , credit1_e, credit2_e. Each of original files are just simple 16 numerical digits (0 to 9) as credit card numbers are. But the last character in the file might has an additional LF or CR.

- For credit0, it was encrypted by standard UNIX program
crypt KEY < credit0 > credit0_e
where KEY is a string of 2 lower-case or upper-case English characters including numbers;
eg Ab, z8, xX, etc.
- For credit1 or credit2, they were encrypted by
crypt KEY < credit1 > credit1_e (same for credit2)
where KEY is a string of 3 lower-case or upper-case English characters including numbers;
eg 9aX, z83, wxY, etc.

You can do **man crypt** in UNIX to find out how to use the program. The decryption is using the same format as encryption in crypt.

You probably want to use *sprintf()* and *system ()* in your C or C++ program. Something like

```
sprintf (syscommand, "crypt %s < %s > %s \n", ckey, infile, outfile);
ret1 = system (syscommand);
```

Write a C or C++ program in our UNIX environment to find correct keys for each `credit0.e`, `credit1.e` and `credit2.e`. Just try all the combinations of possible keys. The main issue is **efficiency**. You must try to reduce the network and disk-file operations as much as possible. Let me know what you have done to try to make your program run faster? **If you know UNIX pipe, you can do Unix pipe, fork and dup() in your C program.** Let us know if this helps? I hope that you know what I meant here. If not, it is okay; you still can get the basic points. Please note that DO NOT fork more than 4 processes on apache; otherwise the school might be upset about our using too much CPU resource.

(2) You can find the encrypted files: `t0.e`, `t1.e` at that directory. These files were encrypted by different keys. But all keys have the same length: 4 ASCII characters. From some inside information, you already know:

- These files (and also `longtest.e`) were encrypted using a simple but specially designed encryption program by us. The source code of decryption program can be obtained from that directory `decrypt.c`. You may revise it and include it as a function in your program for you to decrypt with a guessed key. The source code and executable code for encryption, called `encrypt.c` and `encrypt`, can be found at that directory. Please take a look at "example" file in the directory and see how you can encrypt and decrypt it.
- The length of each key used for encryption of these three files is exactly 4 ASCII characters. These 4 characters are in the set of {a,...,z, A, ..., Z} (English characters) and {0,...,9}. Because key length is short, you can try all the possible combinations in your program. The number of possible combinations is 62^4 .
- The original plain-text is written in English. It may have numbers, mathematical notations, small or capital letters or some characters such as SPACE, LF, ", ', -, :, etc. as an ordinary English articles or report may have. It is hard for anyone to know which characters are appeared in the texts and which are not so you probably have to come out your own analysis scheme in your program to guess. You can find the table of ASCII code in the course web page for your reference. It is pretty much English, so it satisfies most of English property. Here I am giving you an example that the original plaintext may look like: "example" in the directory. Please take a look and try to decrypt `example.e`.
- The original text and the encrypted one have the same length.

Now you may try all the different combinations of keys to find out the original files `t0` and `t1`. It is very important that you should write the fastest programs. Try to minimize the number of disk I/O. Your programs should be runnable in the school UNIX environment. The grading will be based on the correctness and efficiency of your program.

Some suggestions:

- You can include `decrypt.c` into your program as a function. But the original `decrypt.c` is not efficient for your program because it reads from hard-disk character by character while decrypting it. You can read the whole file into a buffer before calling `decrypt` function. Change `decrypt.c` so your program can run faster with the minimum number of disk I/O.
- You may try the `encrypt` program (`encrypt`) yourself that can be found in the same directory as `decrypt.c`. See `Readme` in that directory for how to use it.
- You guess a key, and use it to decrypt the message, and then analyze the decrypted message if the decrypted message is a possible English text. One suggestion (may not be the best) for your analysis part is to decide whether the decrypted characters follow the distribution of English characters occurring in an English article. Think by yourself if you can find some efficient way to make a determination. For example, the frequency of “spaces” can be an unseful information. Or even the dictionary file `/usr/dict/words` might be useful as well. Write down your technique in the report.

The link of ASCII table and the English letter distribution can be found in the course web page.

- Be honest. If your program does not work, SAY SO. TA will test your programs. If your program does not work but you ”claim” yours works, you will have severe penalty!

(3) A longer file: `longtest_e` is also in the directory. It is encrypted using the same `encrypt` program, `encrypt`, with a key of 10 characters. If you can successfully decrypt this file and your program is finished in **5 minutes**, you will get **30 extra points**. Good luck. A brute force way will not work.

Hint: you need to look into the `decrypt` source program and see how you can avoid trying all combinations of keys. Can you make 62^{10} complexity become something like 10×62 ?

What you need to turn in:

1. The hardcopy of your programs. Put how to compile and run your programs.
2. The correct keys of all encrypted files.
3. If you can successfully decrypt the `longtest` file, please say so and tell us what the key is.
4. The method that you determine if the decrypted one is the original text, and anything special that you like us to know; for example, how you make the program run faster?
5. What is the running time for you to find each of the original texts? You can use `"/bin/time yourprogram"` to find out the execution time.

Each team just submits one copy: a **HARD COPY** of your programs, and an **ELECTRONIC COPY** of your programs before the due date.

How to submit your program electronically:

1. In the SUN unix environment, go to the directory that contains all your source code, makefile, executables, `README`, required files, etc. You must explain your programs clearly in the `README` file. **You must write down your names and email addresses in the `README` file.** If TA does not know how to run your programs, that is your fault.
2. Make sure the directory contains **ALL** the files that are able to be compiled and executed for the assignment. The TA will use your `Makefile` to compile your program. You should remove

any redundant files so the tar file will not be too large! Let me say again. Please remove any unnecessary files before submission.

3. In that working directory, issue the command `~edsha/submit/submit.ass1` (Note the `~` in the beginning.) or `/people/cs/e/edsha/submit/submit.ass1` on “apache.utdallas.edu” or one of the school SUN machines. We will look at the time of the submission file to know if you submit your programs before or after the due date. This command must be issued in the directory that contains the files for the assignment.

No cheating is allowed. I will be very upset if we find any cheating. You must do the homework by yourself. TA will read and test each of your programs.