Random Encryption Algorithm

Damien Lefebvre

July 2016 - France

Click here to access the C++ source code on GitHub

1 How the algorithm works

This program simulates the creation of a user profile, with username and password. This information is stored in a file. Once users have created their profile, they can access it by entering their username and the corresponding password. Of course, the information is encrypted, ie. you cannot read the passwords simply by opening up the file. This encryption is achieved through the algorithm. How does it work?

The algorithm first prompts the user for a username, checking that it doesn't already exist. If it's a new username, it then asks for a password, which is taken in as a string. It then creates a new entry in two files, named keys and values. These entries are differentiated by a header, which is simply the username.

In the values file, it generates a certain number of random characters. This number is fixed and chosen in the source code. It will make room in this block by taking out as many characters as there are in the password. It then randomly inserts every password character into this random content. Every position where a character is inserted is recorded, printed into the keys file.

When trying to open a profile, the algorithm first checks that the profile exists. If it does, it will prompt for a password. It will then use the keys file to retrieve the password out of the random content, and then compare the real password with the user's entry. If they match, the program welcomes the user by his username. If it doesn't, it allows them to try again indefinitely.

2 C++ tools used

- file streams
- string streams
- template functions
- bools

- for loops
- while
- do-while loops
- vectors
- random function

3 Discussing the strength

Imagine a company makes use of REM for their online platform. They are using three files:

- the algorithm in a .cpp file,
- the values in a .txt file,
- and the keys in a .txt file.

If hackers steal all three files, all is lost. But if hackers get their hands on two of the three files, the outcome depends on which ones they have. Let's examine the 3 possible scenarios.

3.1 Scenario 1

If the hackers get their hands on the keys and the algorithm, they still can't do anything. They're missing an essential piece: the values file. They're basically on a ship with a map and working compass, but their ship is in the middle of the jungle, stuck deep inland and unable to sail waters. So they can't go anywhere.

Defense mechanisms: unnecessary.

3.2 Scenario 2

If the hackers get their hands on the values and the algorithm, they won't know how to differentiate random filling characters from true password characters. Their only option is to randomly combine all the characters, creating potential passwords they can test. If they have sufficient computing power and enough time, they will eventually combine exactly the right characters in the right order, and hack into the profile.

Defense mechanisms:

- Use as much filling characters as possible, decreasing the probability that they find the exact combination of characters. Finding a password with N characters, among M total characters, allows for $N \times M$ combinations. So if $M = 5,000$, with the average password being 8 characters long, that allows for $5,000 \times 8 = 40,000$ combinations. At a rate of 1 combination every second, testing all cases will take

$$\frac{40,000}{3,600} = 11.111 \tag{1}$$

hours.

- Restrict the number of attempts to access a profile. For instance, forcing a 24h lock after 5 failed attempts. This adds more time in the equation, which delays a potential hack and can discourage the hackers. In our past example, assuming the last combination is the right one, it will take

  $$\frac{40,000 - 1}{5} = 191,995.2 \tag{2}$$

  hours to crack, which is significantly better.

3.3 Scenario 3

If the hackers get their hands on the values and the keys, all they have to do is reverse-engineer the process to understand how the algorithm works. They essentially have the key and the lock in each hand, and all they need is to figure out how to use both.

Defense mechanisms:

- Double encrypt the keys file. This means you apply the REM to the keys as well, while storing the keys position of the keys, either in:
  - the keys file,
  - the values file,
  - a new file.

  Notice these options go from worst to best. This still does not prevent the hackers from figuring out the algorithm, which is a problem we have yet to solve.

4 Epilogue

In Spring 2016, I started learning C++ in an introductory class to programming. After gaining some experience, I felt compelled to exercise with a personal project. As I returned from Los Angeles to France for summer vacation, I had the idea of developing a password encryption algorithm in London Heathrow.