**CS6903, Spring 2014: Modern Cryptography. Instructor: Giovanni Di Crescenzo, gdc258@nyu.edu**

**Extra Credit Problem n. 1**

**Basic Information:** This problem asks for new approaches to develop secure methods for password-based computer access in client-server environments. The problem is based on the observation that in conventional password schemes, an intruder only needs one line of the server's password file to be able to later impersonate a legitimate client. Instead, a different approach, based on a huge password file, makes such attacks much harder, if not practically impossible. This problem is mainly based on Lecture 8.

**Problem Statement:** Consider the Secure Computer Access case study in Lecture 8. The solution provided there is based on a triple (Initialize, Registration, Access) of algorithms run by the server, using input by the one particular client that is asking to register or access his/her computer account, and using a keyed hash function $H_k$ (e.g., SHA2 or SHA3), as follows:

1. *Initialize:* the server randomly chooses $salt \in \{0,1\}^\ell$.
2. *Register:* on input a client's login name $log \in \{0,1\}^\ell$ and password $pw \in \{0,1\}^\ell$, the server computes $htag = H_k(salt|log|pw)$ and stores on its password file a new line containing the sequence $(salt, log, H_k(salt|log|pw))$.
3. *Access:* on input a client's login name $log'$ and password $pw'$, the server computes $htag' = H_k(salt|log'|pw')$ and searches for a sequence $(salt, log', htag')$ in the password file; the server grants access to the client only if such a sequence is found.

One problem with this scheme is that an intruder reading one such $(salt, log, H_k(salt|log|pw))$ sequence from the password file is capable of running a dictionary attack (since the space of all passwords is small) and impersonating the client with login $log$ and password $pw$. To solve this problem, an alternative approach was recently proposed in the cryptography literature, which, after some simplifications, can be summarized as the following alternative scheme (Init,Reg,Acc):

1. *Init:* the server randomly choose $salt \in \{0,1\}^\ell$ and stores a huge $m$-location array $R$ of $\ell$-bit random strings in the password file.
2. *Reg:* on input a client's login name $log \in \{0,1\}^\ell$ and password $pw \in \{0,1\}^\ell$, the server computes $(i_1|\cdots|i_t) = H_k(salt|log|pw)$ and $htag = R[i_1]$ xor $\cdots$ xor $R[i_t]$, and stores on its password file a new line containing the sequence $(salt, log, htag)$.
3. *Access:* on input a client's login name $log'$ and password $pw'$, the server computes $(i'_1|\cdots|i'_t) = H_k(salt|log'|pw')$ and $htag' = R[i'_1] \oplus \cdots \oplus R[i'_t]$), and searches for a sequence $(salt, log', htag')$ in the password file; the server grants access to the client only if such a sequence is found.

The intuition in the new scheme is that even if an intruder reads all of the $(salt, log, htag)$ sequences and a large number $q$ (e.g., $q = m/2$) of the entries of array $R$ on the password file, he won't be able to run a dictionary attack as before (as he will likely miss at least one of the necessary entries in array $R$ that are needed to compute the value $htag$ for almost all passwords $pw$), and thus won't be able to impersonate any one of the clients, unless with very small probability $\epsilon$. Moreover, if the intruder tries to retrieve all entries of array $R$, this would take a long time and would create an event that is noticeable to an intrusion detection system.

**Assignment:** Complete as many as possible among tasks (1)-(5).

1. Write a C or C++ program for algorithms Init, Reg and Acc, and make sure that correctness is satisfied (i.e., a password stored by Reg is later successfully verified by Acc).
2. Perform a running time analysis of the running time of Acc, with relative plots for various values of the parameters $m, t$ and fixing parameter $\ell = 128$. In particular, for $j = 1, 2, 3$, compute and plot a dataset $D_j$, of the running time of Acc, when $t = j * \ell$, as a function of $m$, for large and increasing values of $m$ (for instance, you could set values $m_i$, for $i \geq 0$, where $m_0 = 1000$, $m_i = 2 \cdot m_{i-1}$).
3. Compute or estimate the running time that an adversary algorithm would take to retrieve the content of $q = m/2$ arbitrary locations from array $R$, stating your reasonable assumptions on your system capabilities, computing power and connection capabilities.
4. Compute or estimate the maximum value of $m$ after which your Init program does not work because it cannot find enough memory to store $R$.
5. Submit code and analysis results, using a report similar to the one used in your project.
6. (Extra extra credit:) Consider increasing your available storage and storing $R$ on disk. Repeat (1)-(5).