6.857 Homework	**Problem Set 2**	# 2-1 Rainbow Tables
April 11, 2014		

a) We can create a one-to-one table that maps $\{0,1\}^6$ to a set of the alphanumeric symbols ([A-Za-z0-9_-]). Given that table, we can map $\{0,1\}^{60}$ to an element of S, we will call the mapping l and the inverse mapping l'. We can then use repeated applications of h to generate f. We take the first 60 bits of the input to f, and apply $h(l(N))$ to get a value of the form $\{0,1\}^{128}$. We can then take then take that value and XOR it with the input to f. We then take the next 60 bits of the new input and repeat the operation, continuing until we have less than 60 bits remaining in the input. At this point, we simply take the final 60 bits of the new input, and apply $l(N)$, resulting in an output in S. Since each of the intermediate values is psuedorandom, as a property of h, and the XOR operation does not reduce this, the final output is a psuedorandom function on the entire input.

b) We have the following elements within a chain of k passwords and k password hashes:

$$P_1 \xrightarrow{h} H_1 \xrightarrow{f} P_2 \xrightarrow{h} H_2 \xrightarrow{f} ... \xrightarrow{h} H_k$$

where P_i is the i'th password in the hash chain and H_i is the i'th hash in the hash chain, and $h(P_i) = H_i$. Given a table mapping H_k to P_1, we can retrieve the password of any given intermediate hash, H_j, in this chain by applying functions f and h alternately to the given hash until we reach a P_j such that $h(P_j) = H_j$. If we reach the case where the output of applying h gives the entry in the table (another words, we have reached H_k), we restart this process with the P_1 in the table, and apply h and f alternately. Since the chain has length k, the longest running time is $O(k)$, because we will at most apply k operations of h and f before discovering the entire chain.

c) The logic for this question does not differ much from the logic in part (b). If we are given an H' which occurs in one of the chains, we can alternately apply f and h until we reach an entry within the table. Once we reach one of the H_k which occurs in the table, we know to which chain it belongs, and can alternately apply h and f, starting with the P_1 from the table, until we reach a P_j such that $h(P_j) = H'$. Again, we are guaranteed that we apply at most k operations of f and h before discovering the entire chain to which H' belongs.

If we create $|S|/k$ chains of length k, we will have $|S|$ passwords, though they are very unlikely to be distinct. For every element in $|S|$, h will deterministically map it to exactly one element in $\{0,1\}^d$. Another words, we will get at most $|S|$ elements in the output space of h (the hash space), despite it's size of 2^d. Therefore, we can consider both our password and hash spaces to be at most of size $|S|$.

Taking f to be pseudo-random, the probability of two distinct elements of $\{0,1\}^d$ to map to the same element of output is $1/|S|$. More formally,

$$Pr(f(H_i) = f(H_j), H_i \neq H_j) = \frac{1}{|S|}$$

Since we consider at most $|S|$ elements of the hash space, we will have $\binom{|S|}{2} = |S|(|S|-1)/2$ possible pairs in our hash space, so we expect there to be collisions in applying f from our hash space back to our password space. We expect:

$$E(\text{number of collisions}) = \frac{|S|(|S|-1)}{2} \cdot \frac{1}{|S|} = \frac{|S|-1}{2}$$

Because we assumed $|S|$ unique hashes to begin with (which may already be an over-estimate) and expect almost half the pairs to contain a collision, we are very unlikely to have $|S|$ distinct passwords.

d) We note that previously in part (c), if we were to use the same f at each step, once $f(H') = f(H'')$, two chains will converge, regardless of whether H' and H'' were in the same step of the chain or not. One chain may end earlier than the other, but there is guaranteed to be overlap between them.

However, in this scenario, where a different f_i is used for each step, we are much more likely to get unique passwords and therefore hashes. For example, two chains will converge in the case where $f_i(H') = f_j(H'')$ and $i = j$ because this is the case where a collision happened in the same round. But when $i \neq j$ and $f_i(H') = f_j(H'')$, we are not guaranteed convergence of the chains because f_{i+1} and f_{j+1} are not guaranteed to produce the same result (and in fact, are very unlikely to).

e) One solution is to store $h(h(password)||salt)$, where $||$ is the concatenation operator. *salt* is generated randomly and is not secret. For the adversary to crack the password, he/she would have to generate a rainbow table for each possible value of *salt*. If the *salt* is large enough, it would be infeasible for the adversary to do so.