# Midterm Solutions

March 19, 2014, 11:15–12:05 PM

There are five problems each worth five points for a total of 25 points. Show all your work, partial credit will be awarded. Space is provided on the test for your work; if you use a blue book for additional workspace, sign it and return it with the test. No notes, no collaboration.

Name: _____

| Problem | Credit |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| Total | |

1. *Unix passwords:* To log into your unix account you must have a password. In order to keep the password safe, Unix does not store the password, or the password encrypted, rather it uses the password as a key to a modified DES algorithm to encrypt the value 0, and it stores the result. The modification is dependent on a "salt", a two character value that is stored with the username. That is, what Unix stores is the triple,

$$(username, salt, DES_{salt,password}(0))$$

To login, the system looks up the username, gets the salt, recomputes the encryption of 0 and checks the result against what is stored.

Discuss why Unix does it this way, rather than storing the password encrypted. Give three reasons that would support the Unix method over storing the encrypted password.

What purpose is the salt? Give two reasons.

Solution

(a) Encryption of passwords requires a master key, which is a problem. Hashing has no key.

(b) A master key (encryption) must be stored on the machine, introduces a disclosure vulnerability. Hashing requires no information be kept secret.

(c) The master key might be lost, leading to inability of anyone to log in. Hashing does not have secret information to loose.

(d) If the master key is compromised, all passwords are known. Using hashing, each password can only be compromised separately,

(e) Disclosure of the encrypted password database allows from a brute force attack against the master, eventually disclosing all passwords. The disclosure of the password database under hashing only gives an attack on each password individually.

(f) Using encryption the password must be present at the same place as the master key, to do the encryption. Usually this means that password must be handled by more software. With hashing the software collecting the password can immediately hash it and discard the password, so it is handled by less software.

(g) (Continuing above) Insider attacks are more difficult. The password is never seen by the operating system, only specialized software that collect and hash the password.

(h) Hashing gives added security in a remote login situation, because only the hash needs to be passed between systems for authentication. Using encryption, the actual key must be passed between systems. (Although this makes the hash a password substitute, the user might be using the same password for something entirely different, and hashing contains this sort of error.)

(i) Salt: prevents disclosure of which accounts on a machine use the same password, and therefore which users are possibly the same.

(j) Salt: prevents disclosure or which users across different machines use the same password, giving information about who is who, or suggesting attacks outside of the entire system against particular users using weaker systems whose success can compromise the computer system.

(k) Salt: adds entropy to the user's password.

(l) Salt: makes dictionary attacks (or rainbow tables) less attractive by requiring a different table for each salt.

*Rubric:* I needed to see a reference to the problems of handling a master key; I needed to see an awareness of cracking the entire password database versus cracking a single password; I wanted to see a reference to the limited handling of the password to just the password collection software; I needed to see that salt prevents information disclosure by making the same password look different.

2. *OFB/ECB:* The OFB mode of encryption creates a stream of pseudo-random blocks by iteratively using an encryption by a secret key, and encrypts by XOR'ing with the resulting blocks:

$$\begin{aligned} R_1 &= E_K(IV) \\ R_i &= E_K(R_{i-1}), \qquad i = 1, 2, \ldots \\ C_i &= P_i \oplus R_i, \qquad i = 1, 2, \ldots \end{aligned}$$

This makes OFB susceptible to manipulation through the XOR. In order to avoid that, we can cascade an ECB mode encryption with a second key:

$$C_i = E_{K'}(P_i \oplus R_i), \qquad i = 1, 2, \ldots$$

(a) Show that this now is non-malleable, i.e., it is not possible to change $C_i$ and get a known change of $P_i$.

(b) Give the decryption equation.

(c) Show (use pseudo-code if needed for clarity) that the use of a second key does not give two keys' worth of strength. *Hint: Meet in the middle attack; space-time trade-off.*

Solution

Decryption algorithm:

$$\begin{aligned} C_i &= E_{K'}(P_i \oplus R_i) && \textit{encryption equation} \\ D_{K'}(C_i) &= D_{K'}(E_{K'}(P_i \oplus R_i)) && \textit{apply } D_{K'} \textit{ to both sides} \\ D_{K'}(C_i) &= P_i \oplus R_i && \textit{encryption and decryption cancel} \\ P_i &= D_{K'}(C_i) \oplus R_i && \textit{move } R_i \textit{ to other side of equals} \end{aligned}$$

If $D$ is non-malleable, then by the decryption algorithm, the $P_i$ cannot be reliably changed by changes in $C_i$. It depends on the non-malleability of $D$.

To use a meet-in-the-middle attack for the known pair $(P_i, C_i)$,

(a) Given block size $b$, allocate a table $\mathcal{T}$ of size $2^b$. For each $K$:

  i. Set $\mathcal{T}({E_K}^{i-1}(E_K(IV))) = K$.

(b) For each $K'$:

  i. Calculate $x = D_{K'}(C_i) \oplus P_i$ and look up in the table $\mathcal{T}$ the key that gives this value $x$: $K^* = \mathcal{T}(x)$.

  ii. Check that $x$ is a valid entry in the table. It could be $\mathcal{T}(x)$ is in fact empty; check by recalculating $x = {E_{K^*}}^{i-1}(E_{K^*}(IV))$. If valid, return $(K^*, K')$ as a candidate key.

If the keys $K$ and $K'$ are $k$ bits, and the encryption block size is $b$ bits, then step one gives a table of size $2^b$ and runs in time $2^k$.

The second step runs through the loop $2^k$ times and each time is $O(1)$. The results will include the correct key $(K, K')$ as well as a small number $r$ of spurious keys. Check against an additional plaintext-ciphertext pair to eliminate spurious keys.

The total time is $2^k + 2^k = 2^{k+1}$ with $2^b$ space.

*Rubric:* A clearly stated decryption equation was a must; it must be noted that non-malleability follows from the non-malleability of the ECB encryption; meet-in-the-middle should refer to a table of values based on one of the two keys $K$ or $K'$, and a loop over the other key looking for a match (preferably given by an equation).

3. *Transposition ciphers:* A transposition cipher uses a rearranging of letters in a message to encrypt the message. An example would be a *columnar cipher*: choose an integer $k$ and a permutation of the numbers $1, \ldots, k$; write the message from left to right with $k$ letters on a line; then read off the ciphertext by working down columns, beginning with the column marked 1, then 2, etc.

   Example: *this is a cipher* with key 3,1,4,2 would write:

| 3 | 1 | 4 | 2 |
|---|---|---|---|
| t | h | i | s |
| i | s | a | c |
| i | p | h | e |
| r | | | |

   and give ciphertext: HSPSCETIIRIAH.

   How can you break a columnar cipher?

   You must tell of statistics and their application and provide a notion of the amount of text and time needed to recover the key. You must tell of particular constraints on the message and how those can be exploited.

   Solution:

   The brute force would be to try difference $k's$ and for each $k$ the $k!$ arrangements of the columns. However, what I was hoping people would mention is the frequency of letter pairs. Certain letter pairs such

as *th* occur frequently, and rearranging the columns to make these pairs would help the search. Counting the gaps between $t$ and $h$ in the cipher text and trying to find I bias in the gap length might help identify $k$ quicker than brute force.

*Rubric:* Reference to letter frequencies and letter-pair frequencies desired; use of brute force search of trying various $k$ is expected; reference to the key complexity, $k$ and then the combinatorics of scrambling $k$ columns, is desired.

4. *Challenge-response:* A *challenge-response* protocol proves knowledge of a password without actually showing the password by answering a question that only the password holder could answer. For instance, the challenge is a random number $C$ and the response should be the encryption of the chanllege by the secret key $K$, $E_K(C)$. If the respondent answers with this value, it is inferred that the respondent knows $K$.

   *Note well:* The challenge must be *fresh*, that is, never used before. Else an eavesdropper could have witnessed the challenge and the response and simply replays what it has seen. This is called a *replay attack*.

   The Microsoft CHAP protocol used a challenge-response protocol but it was flawed and allowed the easy cracking of user passwords.

   The Microsoft user password $P$ is 128 bits. Three sub-keys are formed by concatenating 40 bits of zeros to the user password and then dividing the result into three 56 bit sub-keys. The response was formed by DES encrypting the challenge with each of the three resulting sub-keys, $K_1, K_2$ and $K_3$:

$$\begin{aligned} K_1|K_2|K_3 &= P|00\ldots0 \\ R &= DES_{K_1}(C)|DES_{K_2}(C)|DES_{K_3}(C) \end{aligned}$$

   This was claimed to have 128 bit strength, but it does not. Show how to recover the user password $P$ from a single challenge-response pair $(C, R)$ in much less than $2^{128}$ work, and give the exact amount of work required.

   Solution:

Given a challenge-response pair $(C, R)$, break up $R$ into thirds, $R = R_1|R_2|R_3$ where each $R_i$ is 64 bits, and solve by bruit-force the following three equations:

$$R_i = DES_{K_i}(C)$$

Note however that the search for $K_3$ can be constrained by the format $K_3 = K_3'|00\ldots0$ where $K_3'$ is only 8 bits.

The total time to bruit-force is the sum of individual search spaces or:

$$2^{56} + 2^{56} + 2^8 = 2^{56}(1 + 1 + 1/2^{48}) \sim 2^{57}$$

approximately. So the entire 128 bit key $P$ can be recovered in time $2^{57}$.

*Rubric:* Must notice that each of the three sub-keys can be forced independently; should do the math for work factor.

5. *Perfect secrecy:* Recall the definition of Shannon perfect secrecy.

   **Note: Problem revised!**

   Consider this cipher based on geometry. Let $r_1, r_2, \ldots$ be a random sequence of mod 27 integers, and let the plaintext be a sequence $p_1, p_2, \ldots$ of mod 27 numbers. (They can in fact be the letters $a$ through $z$ mapped to the numbers 0 through 25 and the space mapped to 26.)

   The encryption is: Plot the points $(1, p_i)$ and $(2, r_i)$ in the x-y plane. Draw a straight line through those two points and find the line's intersection with the y-axis, $(0, \hat{c})$. Let $c_i = \hat{c} \bmod 27$.

   (a) Write the equations for encryption and decryption. The equations should be in the form of a single function.

   (b) Show that the cipher has perfect secrecy, giving in your answer the assumptions on the sequence $r_i$.

   (c) Suppose I want to "prove" to an adversary that the message encrypted was not $p_1, p_2, \ldots$ but some other message $p_1', p_2', \ldots$. What (phony) squence of "random" numbers should I claim as the $r_i$ to support my subterfuge.

Solutions:

Encryption:

$$\begin{aligned} c_i &= p_i + (p_i - r_i) \bmod 27 \\ &= 2p_i - r_i \bmod 27 \end{aligned}$$

Decryption:

$$\begin{aligned} 2p_i &= c_i + r_i \bmod 27 \\ p_i &= 14\,(c_i + r_i) \bmod 27 \end{aligned}$$

*Note: the original problem was mod 26. This does not give a unique decryption since both $p_i$ and $p_i + 13$ would give the same $c_i$.*

Assuming the $r_i$ are chosen independently and uniformly then the system has perfect secrecy. Formally,

$$\begin{aligned} \mathcal{P}(p \,|\, c) &= \mathcal{P}(c \,|\, p)\mathcal{P}(p)/\mathcal{P}(c) \qquad \textit{Bayes} \\ &= \mathcal{P}(r)\mathcal{P}(p)/\mathcal{P}(c) \qquad \textit{unique } r \textit{ such that } E_r(p) = c \\ &= \mathcal{P}(r)\mathcal{P}(p)/\sum_{p'} \mathcal{P}(c|p')\mathcal{P}(p') \qquad \textit{law of total probability} \\ &= \mathcal{P}(r)\mathcal{P}(p)/\sum_{p'} \mathcal{P}(r')\mathcal{P}(p') \qquad \textit{unique } r' \textit{ such that } E_{r'}(p') = c \\ &= \mathcal{P}(p)/\sum_{p'} \mathcal{P}(p') \qquad \mathcal{P}(r') = \mathcal{P}(r) \textit{ for all } r' \\ &= \mathcal{P}(p) \qquad \textit{denominator sums to 1.} \end{aligned}$$

You can also cite the standard result that for plaintext, cipher text, and key space all of equal size, perfect secrecy follows from equal probability on the key space and for every $c$ and $p$ there is a unique key $r$ such that $c = E_r(p)$.

Given a crypto text $c_i$, the sequence $r_i = 2p_i{}' - c_i \bmod 27$ will decrypt the crypto text to $p_i{}'$.

*Rubric:* The decryption equation had to be sensible; the idea of always being able to calculate $r$ to fit $p'$ had to be clear; the independence and uniform probably on the $r$ had to be referenced.