

## Homework 3

Due: March 12, 2009, 10:00 PM EST

### Problem 1

1. Alice creates and stores membership proofs for the  $n$  elements of a set. Each element has constant size.  
For the sign-all approach:
  - *hash complexity*:  $O(n)$ .
  - *signature complexity*:  $O(n)$  signing.For the hash-tree approach:
  - *hash complexity*:  $O(n)$ ,  $2n - 1$  to be exact.
  - *signature complexity*:  $O(1)$  signing.
2. Bob queries the membership of  $k$  elements.  
For the sign-all approach:
  - *communication complexity*:  $O(k)$ .For the hash-tree approach:
  - *communication complexity*:  $O(k)$ .
3. Alice returns the  $k$  proofs of membership of these elements.  
For the sign-all approach:
  - *communication complexity*:  $O(k)$  signatures.For the hash-tree approach:
  - *communication complexity*:  $O(k \log n)$  hashes and  $O(1)$  signatures.
4. Bob verifies the  $k$  proofs.  
For the sign-all approach:
  - *hash complexity*:  $O(k)$ .
  - *verification complexity*:  $O(k)$  signature verifications.For the hash-tree approach:
  - *hash complexity*:  $O(k \log n)$ .
  - *verification complexity*:  $O(1)$  signature verification.

### Problem 2

- The main difference is that, in the CBC mode, the output of the encryption of one block  $B_i$  participates in the encryption of the next block  $B_{i+1}$  whereas in the ECB mode the successive encryptions are completely independent. All ciphertexts  $C_i, C_{i+1}, \dots, C_n$  are going to be affected. However, during decryption, only  $P_1$  would be corrupted.
- Consider an attacker that intercepts  $(m, t)$  and  $(m', t')$ , where  $t$  and  $t'$  are the MACs that correspond to  $m$  and  $m'$  respectively. Then without knowing the symmetric key, he produces the message  $m || t \oplus m'$  and the MAC for this message equal to  $t'$ . Note

that this is a valid MAC since

$$Q_k(m||t \oplus m') = Q_k(E_k(m) \oplus t \oplus m') = Q_k(t \oplus t \oplus m') = Q_k(m') = t'$$

### Problem 3

1. With  $O(1)$  space, you can do a brute-force attack on all the combinations of keys  $(k_1, k_2)$ . This would take  $O(2^{2\ell})$  time. The pseudocode is as follows:  
**for** all keys  $k_1$  ( $2^\ell$  in total)  
**for** all keys  $k_2$  ( $2^\ell$  in total)  
**If**  $E_{k_1}(E_{k_2}(M)) == C$   
output  $(k_1, k_2)$  and return;
2. If we have  $2^\ell$  space, we can store the output of the first encryption procedure using all possible keys  $k_1$  in a hash table. That will take  $O(2^\ell)$  time and will use  $O(2^\ell)$  space. Then we use all possible keys  $k_2$ , we decrypt the ciphertext until we hit an entry in the hash table. For the first hit, we output the respective keys  $k_1, k_2$ . We are going to need  $O(2^\ell)$  decryptions and therefore, by using  $O(2^\ell)$  space we have a more efficient algorithm that runs in  $O(2^\ell)$ .
3. Yes, there can be more than one pair of keys that map plaintext  $M$  to ciphertext  $C$ . This depends on the size of the keys  $\ell$ : Given a final ciphertext  $C$ , the probability that, under some decryption key  $k_2$ ,  $C$  maps to a certain first ciphertext, is  $2^\ell/2^n$ . Since we can produce  $2^\ell$  first ciphertexts, we have that the expected number of keys  $(k_1, k_2)$  that map plaintext  $M$  to ciphertext  $C$  is  $2^\ell 2^\ell / 2^n$ . Therefore if  $2^{2\ell} > 2^n$  it is likely there are more than one valid pairs of keys.

### Problem 4

1. Your program should first hash the words in the dictionary. Then it should try to match the hashed password with a dictionary hashed word. After a match is found, the password is decided.
2. One solution would be not to use dictionary words for passwords. Other solutions involve not making password files available to adversaries, changing password regularly, making sure that common password-cracking software tools fail cracking a chosen password.