Cryptography assignment 1

Ishan Agarwal
Sr.No 10939

March 2016

Answer 1

WEP does not provide data integrity because the check-sum used in WEP is the CRC 32 check-sum which is linear and RC4 to encrypt. Let $C(m)$ be the cipher text for message m. $C(m) = (m||IC(m)) \oplus RC4(v||k)$ where $RC4()$ is the RC4 encryption algorithm and $IC()$ is the linear checksum. Here v is the initialization vector and k is the key. Consider replacing m with some string $m \oplus d$. We show that given the cipher text of m the adversary can generate a 'valid' cipher text of $m \oplus d$ by xor-ing the previous cipher text with $d||IC(d)$

$$\begin{aligned} C(m \oplus d) =&((m \oplus d)||IC(m \oplus d)) \oplus RC4(v||k) \\ =&((m \oplus d)||(IC(m) \oplus IC(d)) \oplus RC4(v||k) \\ =&((m||IC(m)) \oplus RC4(v||k)) \oplus (d||IC(d)) \\ =&C(m) \oplus (d||IC(d)) \end{aligned}$$

Thus a 'valid' cipher text of $m \oplus d$ can be obtained by xor-ing the previous cipher text of m with $d||IC(d)$. Hence WEP does not provide data integrity as a attacker can generate a new 'valid' cipher text simply from a single 'valid' cipher text without any information about the message itself.

Answer 2

We claim that the round function in DES is such that $f(m,k) = f(\bar{m},\bar{k})$ where f is the round function. This is because each f_i proceeds by performing $E(m_i) \oplus k_i$ after a permutation. After this S boxes are used to get an output of the required length. $E()$ here is an expansion function. Now $m_i \oplus k_i = \bar{m_i} \oplus \bar{k_i}$. Hence the S boxes get the same input whether (m,k) or $(\bar{m}.\bar{k})$ is used. Hence $f(m,k) = f(\bar{m},\bar{k})$. Now consider the encryption method. In each round:

$$(m_i, m_{i+1}) \Rightarrow (m_{i+1}, m_i \oplus f_i(m_{i+1}, k_i)$$

If the messsage and key both are entirely complemented:

$$\begin{aligned}(\overline{m_i}, \overline{m_{i+1}}) &\Rightarrow (\overline{m_{i+1}}, \overline{m_i} \oplus f_i(\overline{m_{i+1}}, \overline{k_i}) \\ &\Rightarrow (\overline{m_{i+1}}, \overline{m_i} \oplus f_i(m_{i+1}, k_i) \\ &\Rightarrow (\overline{m_{i+1}}, \overline{m_i \oplus f_i(m_{i+1}, k_i)} \\ &\Rightarrow \overline{(m_{i+1}, m_i \oplus f_i(m_{i+1}, k_i)}\end{aligned}$$

As this holds for all rounds and all other operations involved in the encryption, for example permutation, preserve bit-wise complementarity. Hence $DES_k(m) = \overline{DES_{\bar{k}}(\bar{m})}$.

Answer 3

In triple DES there are two possible meet in the middle attacks:

- Compute $E_{h1}^{-1}(E_{h2}^{-1}(c))$ for all h1, h2 in the keyspace for these subkeys, each of length 56. We also store these. We then run through possible values of h3 through the subkey space and check if $E_{h1}^{-1}(E_{h2}^{-1}(c)) = E_{h3}(m)$. This was done for some m and c pair. If a match is found we will verify for 3 other message-cipher text pairs to get 2^{-156} and if it works for these there is a probability only of being wrong. Else we continue.
- Compute $E_{h1}^{-1}(c)$ for all h1 in the subkey space of size 56. These are stored. Then we run trough possible h2, h3 in the subkey space and keep checking if $E_{h1}^{-1}(c) = E_{h2}(E_{h3}(m))$. If such a match is found we check for three other message pairs, and this will lead to a probability of only 2^{-156} of being wrong. If it does not match for all 4 pairs we continue

For the analysis we assume keys are distributed uniformly at random. In case one we will need 2^{112} time to compute all $E_{h1}^{-1}(E_{h2}^{-1}(c))$. In the worst case we will have to compute for all h3, taking 2^{56} time. Finally we would have $\frac{2^{56*3}}{2^{64}} = 2^{104}$ matches. Assuming we check for another three pairs it will take $4*2^{104}$ time. As such total time is: $2^{112}+2^{56}+4*2^{104}$ which is still of the order 2^{112}. The space required is for storing the first 2 keys ,112 bits, and 64 bits for the intermediate cipher text i.e $2^{112} * (112 + 64)$. Thus space requirement is c which is of the order $2^{119.5}$.

In the second case we will need 2^{56} time to compute $E_{h1}^{-1}(c)$ and 2^{112} to compute $E_{h2}(E_{h3}(m))$ for all h2, h3 in the worst case. Again we get the same number of matches as in one and we need 3 more checks with message-ciphertext pairs to have only a 2^{-156} only probability of being wrong. This will take $4*2^{104}$ time. Thus total time taken is again $2^{112}+2^{56}+4*2^{104}$ which is still of the order 2^{112}. The space needed however will just be for one subkey and the intermediate cipher text leading to space requirement of only $2^{56} * (56 + 64)$ which is of order $2^{62.9}$. Thus the second attack takes the same time but needs less storage.

Answer 4

In AES CBC we have $m_i = Dec_k(c_i) \oplus (c_{i-1})$. Thus from this itself, the decryption of m_1 as well as m_3, m_4 and onwards are unaffected. The decryption of m_3 will also only be affected in the first bit which would be flipped. Apart from this the change will be in m_1. In this case $Dec_k(c_1)$ will change wholly due to the single change in m_1 as AES succeeds in confusion and diffusion. Hence the only net changes are that the first bit of m_3 will be flipped and m_2 will be affected throughout.

Answer 5

We claim that if H is not collision resistant then so is f, where H is the hash function and f is the compression function. According to Merkle's construction we use a fixed initialization vector as H_0 now f sends a $n+r$ bit string to an n bit output. We use f to send h_i concatenated with x_i to H_{i+1}. Here the final $H_t + 1$ obtained is the output and the x_i are the blocks of the message, m, with the last x stating the length of the message in bits (right justified). Suitable padding with zeros at the second last block. Here H_t is the last stage after which the message length carrying block is added.
Now we begin the proof.
Assume H is not collision resistant. Then there are two messages, m_1 and m_2 that are mapped to the same output.Consider both m_1 and m_2 have different lengths. Then as H maps them to the same output so f must have a collision at the last stage.
Assume that they have different lengths. Let i be the least value such that at the i^{th} stage the value of the compression function is the same. As these are a collision for H, there must be some such i such that $1 \leq i \leq t+1$. If i is more than 1, then H_{i-1} is different for the two messages which means there is a collision in f. If $i = 0$, since m_1 and m_2 are distinct, hence they must differ at some stage, say j. Now $j \geq 1$ so f will have a collision at stage j.
Hence we have proved that finding a collision in H implies finding a collision in f.

This proof requires that the length specifying block be appended. Else consider a message m, whose number of bits, b, is not a multiple of r.Assume we do not have the length specifying block. Consider the last block of the message x_t as 1 for one case and 10 in another (the second message is one bit longer than the first). Let the messages be identical before this last block. However after padding the last blocks, the messages will be indistinguishable, both being $1||0_{r-1}$. Hence this will create an 'artificial' collision for H simply because two different messages are input as the same message, after padding, to H. However, if this is allowed, the proof above does not work as such a collision for H will not imply a collision for f and such collisions have not been considered in the above proof.

Answer 6

Let $p(n,q)$ be the probability of having at least one collision after q balls have been thrown at n bins. Let D_i be the event that no collision has occurred after throwing i balls. Then $P[D_{i+1}|D_i] = \frac{n-i}{n} = 1 - \frac{i}{n}$. Also $P[D_1] = 1$. Also:

$$\begin{aligned}
1 - p(n,q) =& P[D_q] \\
=& P[D_q|D_{q-1}]\Pi_i^{q-1}(1 - \frac{i}{n}) \\
=& \Pi_{i=0}^{q-1}[D_{i+1}|D_i] \\
=& \Pi_{i=0}^{q-1}(1 - \frac{i}{n})
\end{aligned}$$

As $\frac{i}{n}$ is less than 1 we can use the inequality $1 - x \leq e^{-x}$ for each term of the product getting:

$$1 - p(n,q) \leq \Pi_{i=0}^{q-1} e^{-\frac{i}{n}} = e^{-\frac{q(q-1)}{2n}}$$

Thus we finally get the following lower bound:

$$p(n,q) \geq 1 - e^{-\frac{q(q-1)}{2n}}$$

This is the general lower bound. As $q \leq \sqrt{2n}$, thus $\frac{q(q-1)}{2n} \leq 1$. Thus we can use the inequality $1 - e^{-x} \geq (1 - e^{-1})x$ which yields:

$$p(n,q) \geq (1 - \frac{1}{e}).\frac{q(q-1)}{2n}$$

Answer 7

Consider the CBC MAC for block size n and key length l. The algorithm proceeds by requesting MAC tagged messages from the MAC-oracle, for strings of length exactly cn, and keep storing the results indexed by MAC tag. However we keep the last block the same and keep inputting the first two blocks at random but use only distinct initial blocks. The algorithm also checks if there has been a repetion of the tag. Once one such collision is found we have a two messages, $x||m$ and $y||m$. Here y denotes all but the last n bits and x and y are distinct. We now submit $x||m1$ where m1 and m are distinct. Since the way CBC-MAC operates we have $MAC(x||m) = Enc(MAC(x) \oplus m)$. Since we have got a collision, $MAC(x||m) = MAC(y||m)$ Hence, $Enc(MAC(x) \oplus m) = Enc(MAC(y) \oplus m)$. Thus MAC(x)=MAC(y). We now append m1 instead of m thus $x||m1$ and $y||m1$ are distinct strings such that:

$$MAC(x||m1) = Enc(MAC(x) \oplus m1)$$

Hence we have generated a valid MaAC tagged message. Infact, once we have one collission we have many, simply by changing m1 to any M as:

$$MAC(x||M) = Enc(MAC(x) \oplus M) = Enc(MAC(y) \oplus M) = MAC(y||M)$$

The probability of atleast one collision after q trials is, by the birthday paradox result atleast: $1-e^{-\frac{q(q-1)}{2n}}$. Hence if we take$\sqrt{2ln2.2^n}$ different messages we have atleast a probability of $\frac{1}{2}$ of success. this is because we assume the block cipher to be a PRP so we can ,for the purpose of analysis, assume a (almost) uniform distribution over the MAC space. thus the above result (birthday paradiox) can be used with n as 2^n. The time taken will be $\sqrt{2ln2.2^n}$ as can be checked by substituting this as q in the formula.. The space will be needed to store the MAC -oracle output and hence space complexity will also be $\sqrt{2ln2.2^n}$. We can easily tune the time and space complexity, which in this case are linked as the number of messages demanded from the MAC -oracle determines both of these, so as to fix our probability of success as high as we like.