TT

TEXTBOOK TESTBANKS

Test Bank for Network Security 3rd Kaufman

TEST BANK SAMPLE PREVIEW

PUBLISHER

Pearson

COPYRIGHT

2023

RESOURCE TYPE

Test Bank

ISBN

9780136643524

2 INTRODUCTION TO CRYPTOGRAPHY

2.1 What is the dedication to this book?

It's a simple cryptogram.

2.2 Random J. Protocol-Designer has been told to design a scheme to prevent messages from being modified by an intruder. Random J. decides to append to each message a hash of that message. Why doesn't this solve the problem?

Anyone who knows which hash function is being used can forge a message.

2.3 Suppose Alice, Bob, and Carol want to use secret key technology to authenticate each other. If they all used the same secret key K, then Bob could impersonate Carol to Alice (actually any of the three can impersonate the other to the third). Suppose instead that each had their own secret key, so Alice uses K_A, Bob uses K_B, and Carol uses K_C. This means that each of Alice, Bob, and Carol, to prove his or her identity, responds to a challenge with a function of his or her secret key and the challenge. Is this more secure than having them all use the same secret key K? (Hint: what does Alice need to know in order to verify Carol's answer to Alice's challenge?)

Without the use of public key technology, they will still need to know each other's keys to do the verification, which means they can still forge each other's messages.

2.4 As described in §2.4.4 *Efficient Digital Signatures*, it is common, for performance reasons, to sign a hash of a message rather than the message itself. Why is it so important that it be difficult to find two messages with the same hash?

You can forge a signature on any message with the same hash as one that has been legitimately signed, since the signature on the second message will be the same as that on the first.

2.5 Assume a cryptographic algorithm in which the performance for the good guys (the ones that know the key) grows linearly with the length of the key, and for which the only way to break it is a brute-force attack of trying all possible keys. Suppose the performance at a certain keysize is adequate for the good guys (*e.g.*, encryption and decryption can be done as fast as the bits can be transmitted over the wire). Then suppose advances in computer technology make computers twice as fast. Given that both the good guys and the bad guys get faster computers, does this advance in computer speed work to the advantage of the good guys, the bad guys, or does it not make any difference?

If the good guys keep the same keysize, then it's an advantage for the bad guys. But if the good guys double the keysize, doubling their work while still taking the same amount of time as it used to, then it's much worse for the bad guys, since their work squares.

For example, suppose the good guys were using an n-bit key. With a computer twice as fast, they can use a 2n-bit key with the same performance, since doubling the length of the key just doubles their work. However, the bad guys have 2^n times as much work to do with a key twice as long, so it works to the advantage of the good guys.

2.6 Suppose you had a source of really good randomness, and you copied the random output into two places and ⊕'d these. How random would the result be? What if you concatenated the two quantities and hashed the result? Would that be random?

The ⊕ of two identical quantities is zero, not particularly random. With a good hash algorithm, the hash output should be just as random as the random output.

2.7 Suppose the seed for a PRNG (see §2.6.3 *Calculating a Pseudo-Random Stream from the Seed*) is *n* bits long. How many bits of output of the PRNG would you need to see in order to verify a guess of a seed that it is using (with high probability)?

It depends on the PRNG algorithm, but it's almost certainly at least n. You would need to see the output starting from when the seed was set, since many PRNGs produce the same loop of bits for every seed, just starting at different places. With fewer than n bits of output, it's likely that many different seeds will produce that same output. The more bits you see, the greater the probability that only one seed could generate them. Note: If you know how the seed is being generated, for instance as a hash of the clock in units of microseconds, then the effective size of the seed is much smaller than n.

2.8 Suppose you could see some, but not all of the output of a PRNG. Assuming you know the algorithm the PRNG is using, would you be able to verify a guess of a seed based solely on seeing every tenth bit? Would this require trying more potential seeds than if you were able to see all the bits of the PRNG output (assuming you were able to see enough output bits)?

Again, it depends on the PRNG algorithm, but you probably wouldn't need any more bits than if you could see every bit. And again, you'd need to know how many bits were skipped starting from when the seed was set.

2.9 Suppose you could see 400 bits of output of a PRNG every second, and you knew that it started with eight bits of randomness and mixed in eight bits of new randomness every second. How difficult would it be, after sixteen seconds, to calculate what the state of the PRNG is, compared to a PRNG that mixed in 128 bits of randomness every sixteen seconds?

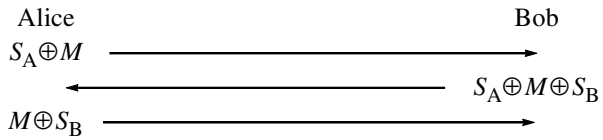
For the 8-bit PRNG, by brute-force search, you should be able to determine the initial eight bits of random input from the first 400 bits of output. On average, this would take about 128 guesses. Knowing the state after one second plus the next 400 bits of output, you can then determine the next eight bits of random input with about 128 guesses. And so on. So it should

take about 16×128 (2^{11}) guesses to calculate the state after 16 seconds. For the 128-bit PRNG, brute-force search would take about 2^{127} guesses.

2.10 Suppose a process generates about eight bits of randomness every second, and suppose a PRNG used the 8-bit random output every second, and suppose you could see the output of the PRNG. Why is it better to wait until you have, say, 128 bits of randomness, before using the randomness to reseed your PRNG? In other words, if the random seed were a function of sixteen 8-bit random chunks, how many possible seeds would there be? What if you waited until there were 128 bits of randomness?

In either case, there are 2^{128} possible seeds. But see the previous question.

2.11 Suppose Alice wants to send a secret message M to Bob, but has not agreed upon a secret with Bob. Furthermore, assume that Alice can be assured that when she sends something to Bob, it will arrive at Bob without anyone having tampered with the message. So the only threat is someone reading the transmissions between Alice and Bob. Alice chooses a random secret S_A, and sends $S_A \oplus M$ to Bob. Nobody (including Bob) can read this message. Bob now creates his own secret S_B, ⊕s what he received from Alice with S_B, and transmits $S_A \oplus M \oplus S_B$ to Alice. Alice now removes her secret by ⊕ing with S_A, and returns $M \oplus S_B$ to Bob. Bob can now ⊕ with his secret (S_B) in order to read the message. Is this secure?



No. The ⊕ of the first two messages is S_B. The ⊕ of the three messages is M.

2.12 Use the Euclidean algorithm to compute gcd(1953,210).

1953 210 63 **21** 0, *i.e.*
1953 mod 210 = 63; 210 mod 63 = 21; 63 mod 21 = 0. *So the gcd is* 21.

2.13 Use the Euclidean algorithm to compute the multiplicative inverse of 9 mod 31.

A=31, B=9:

$a=u_aA+v_aB$	$b=u_bA+v_bB$	u_a	v_a	u_b	v_b	
31	9	1	0	0	1	
9	4	0	1	1	−3	
4	**1**	1	−1	**−2**	**7**	$1 = -2\times31+\mathbf{7}\times9$

So 7 *is the inverse of* 9 mod 31.

2.14 A number is 11 mod 36 and also 7 mod 49. What is the number mod 1764? How about if a number is 11 mod 36 and also 11 mod 49—what would that number be mod 1764?

Running the Euclidean algorithm on A=49 *and* B=36:

$a=u_aA+v_aB$	$b=u_bA+v_bB$	u_a	v_a	u_b	v_b	
49	36	1	0	0	1	
36	13	0	1	1	−1	
13	10	1	−1	−2	3	
10	3	−2	3	3	−4	
3	**1**	3	−4	**−11**	**15**	$1 = -11\times49 + 15\times36$

$7\times15\times36-11\times11\times49 =$ **1379** mod 1764

$11\times15\times36-11\times11\times49 =$ **11** mod 1764

3 SECRET KEY CRYPTOGRAPHY

3.1 Write a program that implements an oracle that encrypts and decrypts, using 8-bit blocks and 8-bit keys (as described in §3.2.3 *Looking Random*).

```
__all__=['encrypt','decrypt']

from random import sample

class encoracle() :
  def __init__(self) :
    self.p = set(tuple(range(256)))  # unused plaintext values
    self.c = set(tuple(range(256)))  # unused ciphertext values
    self.encryption = [None for _ in range(256)]
    self.decryption = [None for _ in range(256)]

  def encrypt(self,p) :
    c = self.encryption[p]      # look up plaintext p's ciphertext value
    if c is None :              # but if no ciphertext has been assigned:
      c = sample(self.c,1)[0]   #   pick an unused ciphertext value, c
      self.c.remove(c)          #   remove c from the unused ciphertext values
      self.encryption[p] = c    #   assign c to p for encryption
      self.decryption[c] = p    #   assign p to c for decryption
      self.p.remove(p)          #   remove p from the unused plaintext values
    return c

  def decrypt(self,c) :
    p = self.decryption[c]      # look up the ciphertext c's plaintext value
    if p is None :              # but if no plaintext has been assigned:
      p = sample(self.p,1)[0]   #   pick an unused plaintext value, p
      self.p.remove(p)          #   remove p from the unused plaintext values
      self.decryption[c] = p    #   assign p to c for decryption
      self.encryption[p] = c    #   assign c to p for encryption
      self.c.remove(c)          #   remove c from the unused ciphertext values
    return p

cipher = [encoracle() for _ in range(256)]    # a cipher for each key
encrypt = lambda k,p: cipher[k].encrypt(p)
decrypt = lambda k,c: cipher[k].decrypt(c)
```

3.2 If both the keysize and the blocksize were 128 bits, for how many keys, on average, would the encryption of a block with given value x be a block with given value y?

1

3.3 If the keysize is 256 bits and the blocksize is 128 bits, for how many keys, on average, would the encryption of a block with given value x be a block with given value y?

2^{128}

3.4 Keeping in mind that a useful encryption algorithm requires the ability to decrypt, which of the following are possible?
a) An encryption algorithm for which there are two tuples, $\langle key_1, block_1 \rangle$ and $\langle key_2, block_2 \rangle$, that map to the same ciphertext block.
b) An encryption algorithm for which there are two tuples, $\langle key_1, block_1 \rangle$ and $\langle key_1, block_2 \rangle$, that map to the same ciphertext block.
c) An encryption algorithm that maps plaintext blocks of size k bits to ciphertext blocks of size j bits. Is this possible if $k < j$? How about if $k > j$?
d) An encryption algorithm that takes as input a triple ⟨key, plaintext block of size k, random number R⟩, and outputs ⟨R, ciphertext block of size k⟩ (where the ciphertext depends on all three inputs).

a, c if $k \leq j$, d

3.5 Is it possible for an encryption algorithm, when encrypting with key K, to map plaintext P to itself?

yes

3.6 Show that if each round transformation in an encryption algorithm is linear, that the encryption algorithm itself will be linear.

The composition of linear transformations is a linear transformation.

3.7 Suppose one had a piece of hardware that did 3DES implemented using EDE. How could you use that hardware to implement DES?

Use the same key three times.

3.8 Suppose in a Feistel cipher, the mangler function mapped every 32-bit value to zero, regardless of the value of its input. What function would encryption compute if there was only one round? What function would encryption compute if there were eight rounds?

With one round, it would exchange left and right halves. With eight rounds, ciphertext = plaintext.

3.9 Show that DES encryption and decryption are identical except for the order of the 48-bit keys. Hint: Running a round backwards is the same as running it forwards but with the halves swapped, and DES has a swap after round 16 when run forwards (see §3.5.1 *DES Overview*).

The initial and final permutations are inverses of each other, so that part is identical. For decryption, if we replace each backward round with a swap followed by a forward round followed by a swap, then pairs of swaps cancel and we're left with DES encryption, but with the key order reversed.

4 MODES OF OPERATION

4.1 In our example mode *Randomized ECB* (see Figure 4-4), suppose that a bit of one of the ciphertext blocks, say c_i, is flipped (intentionally or unintentionally). What will that do to the decrypted message? Suppose a bit of an r_i is flipped? What would that do to the decrypted message? Suppose the ciphertext were being transmitted as a stream of octets, and one octet was lost (and the receiver was not notified that an octet was lost)? What would that do to the decrypted message?

Flipping a bit in c_i garbles m_i. Flipping a bit in r_i flips the corresponding bit in m_i. Losing an octet would garble the message starting with the first affected block.

4.2 In CBC mode, suppose one bit of ciphertext is flipped. What will that do to the decrypted message? Suppose the ciphertext were being transmitted as a stream of octets, and one octet was lost (without the receiver being notified that an octet was lost)? What would that do to the decrypted message? Suppose an entire block was lost. What would that do to the decrypted message?

If a bit of ciphertext is flipped, the corresponding plaintext block will be garbled and the following plaintext block will have the corresponding bit flipped. If an octet of ciphertext were lost, the plaintext will be garbled starting with the block where the ciphertext octet was lost. If an entire block was lost, the corresponding plaintext block and the next block would be replaced with one block of garbage, and the rest of the message would be intact.

4.3 Consider the following alternative method of encrypting a message. To encrypt a message, use the algorithm for CBC decryption with an implicit IV of zero. To decrypt a message, use the algorithm for CBC encryption with an implicit IV of zero. Would this work? ("Working" means that decrypt reverses the encrypt operation.) What are the security implications of this, if any, as contrasted with the "normal" CBC? Hints: How many ciphertext blocks change if you change one block of message in normal CBC compared to our proposed CBC variant? If there are repeated blocks of plaintext, are there detectable patterns in the ciphertext?

It would work. In normal CBC, changing a block of plaintext changes the corresponding ciphertext block and all following ciphertext blocks. In this variant, changing a block of plaintext changes only the corresponding ciphertext block and the following ciphertext block; if there are n identical plaintext blocks in a row, there will be n−1 identical ciphertext blocks in a row; if a consecutive pair of plaintext blocks is repeated, the corresponding pairs of ciphertext blocks will have identical second blocks.

4.4 Suppose you know that the employee database is encrypted using randomized ECB (see §4.2.2.1), and suppose you know that your salary is in plaintext block m_7, that you have

access to the ciphertext, and you know the syntax of the plaintext. How can you modify the ciphertext to increase your salary?

You can calculate the $\oplus$ of the m_7 you want with the m_7 you have, and $\oplus$ that quantity into the corresponding r_7.

4.5 Why is it possible to decrypt, in parallel, blocks of ciphertext that were encrypted using CBC mode, but it is not possible to encrypt, in parallel, blocks of plaintext with CBC mode?

A plaintext block only depends on the corresponding ciphertext block and its predecessor. A ciphertext block depends on the corresponding plaintext block and all previous plaintext blocks.

4.6 Suppose a message is encrypted with CTR mode, and you know that plaintext block m_7 contains your salary. Suppose you have access to the ciphertext. How can you modify the ciphertext to increase your salary?

You can calculate the $\oplus$ of what you want with what you have, and $\oplus$ that quantity into the corresponding ciphertext.

4.7 In CBC mode, suppose you reused the same IV for two messages. What are the security implications?

An eavesdropper could tell if the two messages were identical. Also, if the two messages were similar, an eavesdropper could see where they began to differ.

4.8 Suppose you wanted to decrypt only block n of a file that was encrypted with CBC mode (and you knew the key with which the file was encrypted). Which blocks of the encrypted file would you need to read? What cryptographic operations would you need to do?

You'd need to see blocks n and n–1. You'd $\oplus$ ciphertext block n–1 with the decryption of ciphertext block n.

4.9 How does $\oplus$ing K_1 into the final block in CMAC prevent the attack described in section §4.3.1.1 *CBC Forgery Attack*?

Doing this breaks the simple relationship between the CBC of a message and that of a longer message, because the CBC residue of the shorter message is no longer the last ciphertext of that message.

4.10 For each scenario, compare the number of AES operations required in XTS *versus* XEX:
a) Encrypting an n-block plaintext that is an integral number of blocks.
b) Encrypting an n-block plaintext that is not an integral number of blocks.
c) Decrypting an n-block ciphertext.
d) Decrypting only block n. (Two questions: if the ciphertext is an integral number of blocks, or if it isn't.)