# Week 1 - Problem Set

**Question 1**

Data compression is often used in data storage and transmission. Suppose you want to use data compression in conjunction with encryption. Does it make more sense to:

The order does not matter -- neither one will compress the data.

The order does not matter -- either one is fine.

Encrypt then compress.

Compress then encrypt.

Ciphertexts tend to look like random strings and therefore the only opportunity for compression is prior to encryption.

**Question 2**

Let *G*:{0,1}*s*→{0,1}*n* be a secure PRG. Which of the following is a secure PRG (there is more than one correct answer):

*G*′(*k*1,*k*2)=*G*(*k*1)∥∥*G*(*k*2)     (here ∥∥ denotes concatenation)

a distinguisher for G′ gives a distinguisher for G.

*G*′(*k*)=*G*(0)

A distinguisher will output not random whenever its input is equal to G(0).

*G*′(*k*)=reverse(*G*(*k*))     where reverse(x) reverses the string x so that the first bit of x is the last bit of reverse(x), the second bit of x is the second to last bit of reverse(x), and so on.

a distinguisher for G′ gives a distinguisher for G.

*G*′(*k*)=*G*(*k*)∥∥0     (here ∥∥ denotes concatenation)

A distinguisher will output not random whenever the last bit of its input is 0.

*G*′(*k*)=*G*(*k*)∥∥*G*(*k*)     (here ∥∥ denotes concatenation)

A distinguisher will output not random whenever the first n bits are equal to the last n bits.

*G*′(*k*)=*G*(*k*)[0,…,*n*−2]     (i.e., *G*′(*k*) drops the last bit of *G*(*k*))

a distinguisher for G′ gives a distinguisher for G.

### Question 3

Let *G*:*K*→{0,1}*n* be a secure PRG. Define *G*′(*k*1,*k*2)=*G*(*k*1)⋀*G*(*k*2) where ⋀ is the bit-wise AND function. Consider the following statistical test *A* on {0,1}*n*:   
*A*(*x*) outputs LSB(*x*), the least significant bit of *x*.   
What is *Adv*PRG[*A*,*G*′] ?    You may assume that LSB(*G*(*k*)) is 0 for exactly half the seeds *k* in *K*.   
  
Note: Please enter the advantage as a decimal between 0 and 1 with a leading 0. If the advantage is 3/4, you should enter it as 0.75

For a random string x we have Pr[A(x)=1]=1/2 but for a pseudorandom string G′(k1,k2) we have Prk1,k2[A(G′(k1,k2))=1]=1/4.

### Question 4

Let (*E*,*D*) be a (one-time) semantically secure cipher with key space *K*={0,1}ℓ. A bank wishes to split a decryption key *k*∈{0,1}ℓ into two pieces *p*1 and *p*2 so that both are needed for decryption. The piece *p*1 can be given to one executive and *p*2 to another so that both must contribute their pieces for decryption to proceed.

The bank generates random *k*1 in {0,1}ℓ and sets *k*′1←*k*⊕*k*1. Note that *k*1⊕*k*′1=*k*. The bank can give *k*1 to one executive and *k*′1 to another. Both must be present for decryption to proceed since, by itself, each piece contains no information about the secret key *k* (note that each piece is a one-time pad encryption of *k*).

Now, suppose the bank wants to split *k* into three pieces *p*1,*p*2,*p*3 so that any two of the pieces enable decryption using *k*. This ensures that even if one executive is out sick, decryption can still succeed. To do so the bank generates two random pairs (*k*1,*k*′1) and (*k*2,*k*′2) as in the previous paragraph so that *k*1⊕*k*′1=*k*2⊕*k*′2=*k*. How should the bank assign pieces so that any two pieces enable decryption using *k*, but no single piece can decrypt?

*p*1=(*k*1,*k*2),*p*2=(*k*′1,*k*′2),*p*3=(*k*′2)

*p*1=(*k*1,*k*2),*p*2=(*k*1,*k*2),*p*3=(*k*′2)

*p*1=(*k*1,*k*2),*p*2=(*k*′1,*k*2),*p*3=(*k*′2)

*p*1=(*k*1,*k*2),*p*2=(*k*2,*k*′2),*p*3=(*k*′2)

*p*1=(*k*1,*k*2),*p*2=(*k*′1),*p*3=(*k*′2)

executives 1 and 2 can decrypt using k1,k′1, executives 1 and 3 can decrypt using k2,k′2, and executives 2 and 3 can decrypt using k2,k′2. Moreover, a single executive has no information about k.

### Question 5

Let *M*=*C*=*K*={0,1,2,…,255} and consider the following cipher defined over (*K*,*M*,*C*):   
*E*(*k*,*m*)=*m*+*k*(mod256);*D*(*k*,*c*)=*c*−*k*(mod256) .   
Does this cipher have perfect secrecy?

Yes.

as with the one-time pad, there is exactly one key mapping a given message m to a given ciphertext c.

No, only the One Time Pad has perfect secrecy.

No, there is a simple attack on this cipher.

### Question 6

Let (*E*,*D*) be a (one-time) semantically secure cipher where the message and ciphertext space is {0,1}*n*. Which of the following encryption schemes are (one-time) semantically secure?

*E*′( (*k*,*k*′), *m*)=*E*(*k*,*m*)∥∥*E*(*k*′,*m*)

an attack on E′ gives an attack on E.

*E*′(*k*,*m*)=*E*(*k*,*m*)∥∥LSB(*m*)

To break semantic security, an attacker would ask for the encryption of $0^n$ and $0^{n-1}1$ and can distinguish EXP(0) from EXP(1).

*E*′(*k*,*m*)=reverse(*E*(*k*,*m*))

an attack on E′ gives an attack on E.

*E*′(*k*,*m*)=*E*(*k*,*m*)∥∥*k*

To break semantic security, an attacker would read the secret key from the challenge ciphertext and use it to decrypt the challenge ciphertext. Basically, any ciphertext reveals the secret key.

*E*′(*k*,*m*)=compute *c*←*E*(*k*,*m*) and output  *c*∥∥*c* (i.e., output *c* twice)

an attack on E′ gives an attack on E.

*E*′(*k*,*m*)=*E*(0*n*,*m*)

To break semantic security, an attacker would ask for the encryption of $0^n$ and $1^n$ and can easily distinguish EXP(0) from EXP(1) because it knows the secret key, namely 0n.

**Question 7**

Suppose you are told that the one time pad encryption of the message "attack at dawn" is *6c73d5240a948c86981bc294814d* (the plaintext letters are encoded as 8-bit ASCII and the given ciphertext is written in [hex](http://en.wikipedia.org/wiki/Hexadecimal)). What would be the one time pad encryption of the message "attack at dusk" under the same OTP key?

See the code

Main idea:

Given:

pt1 = "attack at dawn"🡺.ascii1,

pt2 = "attack at dusk"🡺 ascii2,

ascii1 XOR k = 6c73d5240a948c86981bc294814d = ct1

* k = ct1 XOR ascii1

🡺ct2=k XOR ascii2=6c73d5240a948c86981bc2808548

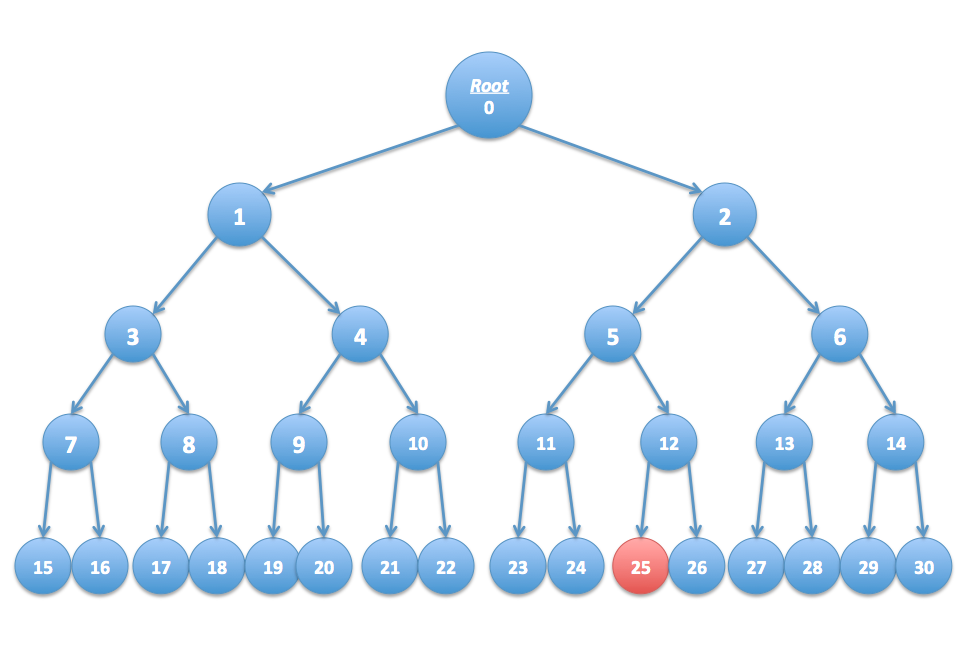
### Question 8

The movie industry wants to protect digital content distributed on DVD’s. We develop a variant of a method used to protect Blu-ray disks called [AACS](http://en.wikipedia.org/wiki/Advanced_Access_Content_System).

Suppose there are at most a total of *n* DVD players in the world (e.g. *n*=232). We view these *n* players as the leaves of a binary tree of height log2*n*. Each node in this binary tree contains an AES key *ki*. These keys are kept secret from consumers and are fixed for all time. At manufacturing time each DVD player is assigned a serial number *i*∈[0,*n*−1]. Consider the set of nodes *Si* along the path from the root to leaf number *i* in the binary tree. The manufacturer of the DVD player embeds in player number *i* the keys associated with the nodes in the set *Si*. A DVD movie *m* is encrypted as   
*E*(*k*root,*k*)∥∥*E*(*k*,*m*)   
where *k* is a random AES key called a content-key and *k*root is the key associated with the root of the tree. Since all DVD players have the key *k*rootall players can decrypt the movie *m*. We refer to *E*(*k*root,*k*) as the header and *E*(*k*,*m*) as the body. In what follows the DVD header may contain multiple ciphertexts where each ciphertext is the encryption of the content-key *k* under some key *ki* in the binary tree.

Suppose the keys embedded in DVD player number *r* are exposed by hackers and published on the Internet. In this problem we show that when the movie industry distributes a new DVD movie, they can encrypt the contents of the DVD using a slightly larger header (containing about log2*n* keys) so that all DVD players, except for player number *r*, can decrypt the movie. In effect, the movie industry disables player number *r* without affecting other players.

As shown below, consider a tree with *n*=16 leaves. Suppose the leaf node labeled 25 corresponds to an exposed DVD player key. Check the set of keys below under which to encrypt the key *k* so that every player other than player 25 can decrypt the DVD. Only four keys are needed.



9

6You cannot encrypt k under 2, but 6's children must be able to decrypt k.

11You cannot encrypt k under key 5, but 11's children must be able to decrypt k.

7

1You cannot encrypt k under the root, but 1's children must be able to decrypt k.

15

26You cannot encrypt k under any key on the path from the root to node 25. Therefore 26 can only decrypt if you encrypt k under key k26.

14

n: total of DVD players.

Consider the n players forms a binary tree of height log2n, each node contains key ki, which is fixed and kept secret.

During manufacturing, each DVD is assigned a serial number Si, 0<=i<=n-1.

Manufacturer embends i in the keys associated with the nodes in the set Si.

DVD m is encrypted as E(kroot,k)||E(k,m), k is random, kroot associated with the root.

All DVD has kroot, all can decrypt m.

### Question 9

Continuing with the previous question, if there are n DVD players, what is the number of keys under which the content key k must be encrypted if exactly one DVD player's key needs to be revoked?

n−1

log2nThe key will need to be encrypted under one key for each node on the path from the root to the revoked leaf. There are log2n nodes on the path.

n/2

2

n−−√

### Question 10

Continuing with question 8, suppose the leaf nodes labeled 16, 18, and 25 correspond to exposed DVD player keys. Check the smallest set of keys under which to encrypt the key k so that every player other than players 16,18,25 can decrypt the DVD. Only six keys are needed.

21

4Yes, this will let players 19-22 decrypt.

15Yes, this will let player 15 decrypt.

17Yes, this will let player 17 decrypt.

11Yes, this will let players 23,24 decrypt.

6Yes, this will let players 27-30 decrypt.

26Yes, this will let player 26 decrypt.

28

5

14

### Programming Question 1

**Many Time Pad**   
  
Let us see what goes wrong when a stream cipher key is used more than once. Below are eleven hex-encoded ciphertexts that are the result of encrypting eleven plaintexts with a stream cipher, all with the same stream cipher key. Your goal is to decrypt the last ciphertext, and submit the secret message within it as solution.   
  
**Hint:** XOR the ciphertexts together, and consider what happens when a space is XORed with a character in [a-zA-Z].   
  
**ciphertext #1:**

315c4eeaa8b5f8aaf9174145bf43e1784b8fa00dc71d885a804e5ee9fa40b16349c146fb778cdf2d3aff021dfff5b403b510d0d0455468aeb98622b137dae857553ccd8883a7bc37520e06e515d22c954eba5025b8cc57ee59418ce7dc6bc41556bdb36bbca3e8774301fbcaa3b83b220809560987815f65286764703de0f3d524400a19b159610b11ef3e

**ciphertext #2:**

234c02ecbbfbafa3ed18510abd11fa724fcda2018a1a8342cf064bbde548b12b07df44ba7191d9606ef4081ffde5ad46a5069d9f7f543bedb9c861bf29c7e205132eda9382b0bc2c5c4b45f919cf3a9f1cb74151f6d551f4480c82b2cb24cc5b028aa76eb7b4ab24171ab3cdadb8356f

**ciphertext #3:**

32510ba9a7b2bba9b8005d43a304b5714cc0bb0c8a34884dd91304b8ad40b62b07df44ba6e9d8a2368e51d04e0e7b207b70b9b8261112bacb6c866a232dfe257527dc29398f5f3251a0d47e503c66e935de81230b59b7afb5f41afa8d661cb

**ciphertext #4:**

32510ba9aab2a8a4fd06414fb517b5605cc0aa0dc91a8908c2064ba8ad5ea06a029056f47a8ad3306ef5021eafe1ac01a81197847a5c68a1b78769a37bc8f4575432c198ccb4ef63590256e305cd3a9544ee4160ead45aef520489e7da7d835402bca670bda8eb775200b8dabbba246b130f040d8ec6447e2c767f3d30ed81ea2e4c1404e1315a1010e7229be6636aaa

**ciphertext #5:**

3f561ba9adb4b6ebec54424ba317b564418fac0dd35f8c08d31a1fe9e24fe56808c213f17c81d9607cee021dafe1e001b21ade877a5e68bea88d61b93ac5ee0d562e8e9582f5ef375f0a4ae20ed86e935de81230b59b73fb4302cd95d770c65b40aaa065f2a5e33a5a0bb5dcaba43722130f042f8ec85b7c2070

**ciphertext #6:**

32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd2061bbde24eb76a19d84aba34d8de287be84d07e7e9a30ee714979c7e1123a8bd9822a33ecaf512472e8e8f8db3f9635c1949e640c621854eba0d79eccf52ff111284b4cc61d11902aebc66f2b2e436434eacc0aba938220b084800c2ca4e693522643573b2c4ce35050b0cf774201f0fe52ac9f26d71b6cf61a711cc229f77ace7aa88a2f19983122b11be87a59c355d25f8e4

**ciphertext #7:**

32510bfbacfbb9befd54415da243e1695ecabd58c519cd4bd90f1fa6ea5ba47b01c909ba7696cf606ef40c04afe1ac0aa8148dd066592ded9f8774b529c7ea125d298e8883f5e9305f4b44f915cb2bd05af51373fd9b4af511039fa2d96f83414aaaf261bda2e97b170fb5cce2a53e675c154c0d9681596934777e2275b381ce2e40582afe67650b13e72287ff2270abcf73bb028932836fbdecfecee0a3b894473c1bbeb6b4913a536ce4f9b13f1efff71ea313c8661dd9a4ce

**ciphertext #8:**

315c4eeaa8b5f8bffd11155ea506b56041c6a00c8a08854dd21a4bbde54ce56801d943ba708b8a3574f40c00fff9e00fa1439fd0654327a3bfc860b92f89ee04132ecb9298f5fd2d5e4b45e40ecc3b9d59e9417df7c95bba410e9aa2ca24c5474da2f276baa3ac325918b2daada43d6712150441c2e04f6565517f317da9d3

**ciphertext #9:**

271946f9bbb2aeadec111841a81abc300ecaa01bd8069d5cc91005e9fe4aad6e04d513e96d99de2569bc5e50eeeca709b50a8a987f4264edb6896fb537d0a716132ddc938fb0f836480e06ed0fcd6e9759f40462f9cf57f4564186a2c1778f1543efa270bda5e933421cbe88a4a52222190f471e9bd15f652b653b7071aec59a2705081ffe72651d08f822c9ed6d76e48b63ab15d0208573a7eef027

**ciphertext #10:**

466d06ece998b7a2fb1d464fed2ced7641ddaa3cc31c9941cf110abbf409ed39598005b3399ccfafb61d0315fca0a314be138a9f32503bedac8067f03adbf3575c3b8edc9ba7f537530541ab0f9f3cd04ff50d66f1d559ba520e89a2cb2a83

**target ciphertext (decrypt this one):**

32510ba9babebbbefd001547a810e67149caee11d945cd7fc81a05e9f85aac650e9052ba6a8cd8257bf14d13e6f0a803b54fde9e77472dbff89d71b57bddef121336cb85ccb8f3315f4b52e301d16e9f52f904

For completeness, here is the python script used to generate the ciphertexts. (it doesn't matter if you can't read this)

import sys

MSGS = ( --- 11 secret messages --- )

def strxor(a, b): # xor two strings of different lengths

if len(a) > len(b):

return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(a[:len(b)], b)])

else:

return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(a, b[:len(a)])])

def random(size=16):

return open("/dev/urandom").read(size)

def encrypt(key, msg):

c = strxor(key, msg)

print

print c.encode('hex')

return c

def main():

key = random(1024)

ciphertexts = [encrypt(key, msg) for msg in MSGS]

Enter the decrypted message in the box below.