# Week2 - Problem Set

**Question 1**

Consider the following five events:

1. Correctly guessing a random 128-bit AES key on the first try.
2. Winning a lottery with 1 million contestants (the probability is 1/106 ).
3. Winning a lottery with 1 million contestants 5 times in a row (the probability is (1/106)5 ).
4. Winning a lottery with 1 million contestants 6 times in a row.
5. Winning a lottery with 1 million contestants 7 times in a row.

What is the order of these events from most likely to least likely?

|  |
| --- |
| 2, 3, 1, 5, 4 |
| 2, 4, 3, 1, 5 |
| 2, 3, 4, 1, 5 |
| 2, 3, 1, 4, 5 |
|  |

210~=103

Prob1=1/2128

Prob2=1/106~=1/220

Prob3=Prob25=1/1030~=1/2100

Prob4=Prob26=1/1036~=1/2120

Prob5=Prob27=1/1042~=1/2140

🡺2>3>4>1>5

**Question 2**

Suppose that using commodity hardware it is possible to build a computer for about $200 that can brute force about 1 billion AES keys per second. Suppose an organization wants to run an exhaustive search for a single 128-bit AES key and was willing to spend 4 trillion dollars to buy these machines (this is more than the annual US federal budget). How long would it take the organization to brute force this single 128-bit AES key with these machines? Ignore additional costs such as power and maintenance.

More than an hour but less than a day

More than a million years but less than a billion (109) years

More than a billion (109) years

More than a 100 years but less than a million years

More than a month but less than a year

Number of machines: 4\*1012/200=2\*1010

Number of keys processed per sec=109\*(2\*1010)= 2\*1019

Seconds needed=2128/(2\*1019)

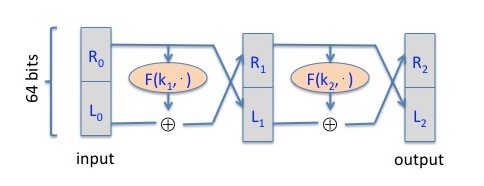
Year needed=2128/(2\*1019)/60/60/24/365~=540 billion years

### Question 3

Let *F*:{0,1}*n*×{0,1}*n*→{0,1}*n* be a secure PRF (i.e. a PRF where the key space, input space, and output space are all {0,1}*n*) and say *n*=128. Which of the following is a secure PRF (there is more than one correct answer):

|  |
| --- |
| *F*′(*k*,*x*)=*F*(*k*,*x*)  ∥∥  0  Not a PRF. A distinguisher will output not random whenever the last bit of F(k,0n) is 0. |
| *F*′(*k*, *x*)=*F*(*k*,*x*) when *x*≠0*n; =k* otherwise  Not a PRF. A distinguisher will query at x=0n and obtain k and then query at x=1n and output not random if the response is F(k,1n). This is unlikely to hold for a truly random function. |
| *F*′((*k*1,*k*2), *x*)=*F*(*k*1,*x*) when *x*≠0*n;= k*2 otherwise |
| *F*′(*k*,*x*)=*F*(*k*,*x*)[0,…,*n*−2]     (i.e., *F*′(*k*,*x*) drops the last bit of *F*(*k*,*x*)) |
| *F*′(*k*, *x*)=*k*⨁*x*  Not a PRF. A distinguisher will query at x=0n and x=1n and output not random if the xor of the response is 1n. This is unlikely to hold for a truly random function. |
| *F*′((*k*1,*k*2), *x*)=*F*(*k*1,*x*)⨁*F*(*k*2,*x*) |

### Question 4

Recall that the Luby-Rackoff theorem discussed in [Lecture 3.2](https://www.coursera.org/crypto/lecture/view?lecture_id=13) states that applying a **three** round Feistel network to a secure PRF gives a secure block cipher. Let's see what goes wrong if we only use a **two** round Feistel. Let *F*:*K*×{0,1}32→{0,1}32 be a secure PRF. Recall that a 2-round Feistel defines the following PRP   *F*2:*K*2×{0,1}64→{0,1}64:   
  
Here *R*0 is the right 32 bits of the 64-bit input and *L*0 is the left 32 bits.   
  
One of the following lines is the output of this PRP *F*2 using a random key, while the other three are the output of a truly random permutation *f*:{0,1}64→{0,1}64. All 64-bit outputs are encoded as 16 hex characters. Can you say which is the output of the PRP?   Note that since you are able to distinguish the output of *F*2 from random, *F*2 is not a secure block cipher, which is what we wanted to show.   
  
**Hint:** First argue that there is a detectable pattern in the xor of *F*2(⋅,064) and *F*2(⋅,132032). Then try to detect this pattern in the given outputs.

On input 064 the output is "5f67abaf 5210722b".    On input 132032 the output is "bbe033c0 0bc9330e".

On input 064 the output is "7b50baab 07640c3d".   On input 132032 the output is "ac343a22 cea46d60".

On input 064 the output is "7c2822eb fdc48bfb".    On input 132032 the output is "325032a9 c5e2364b".

On input 064 the output is "e86d2de2 e1387ae9".   On input 132032 the output is "1792d21d b645c008".

For F2

For R0=L0=032🡺

R1=F(K1,R0) XOR L0= F(K1,R0), L1=R0

R2=F(K2,R1) XOR L1= F(K2,R1), L2=R1= F(K1,R0)

For R0=032 L0=132🡺

R1=F(K1,R0) XOR L0= F(K1,R0) flipped, L1=R0

R2=F(K2,R1) XOR L1= F(K2,R1), L2=R1= F(K1,R0) flipped

The left parts of the two inputs is the flipped version of each other, i.e. F(⋅,064)left⨁F(⋅,132032)left=132.

### Question 5

Nonce-based CBC. Recall that in [lecture 4.4](https://class.coursera.org/crypto-014/quiz/attempt?quiz_id=58) we said that if one wants to use CBC encryption with a non-random unique nonce then the nonce must first be encrypted with an **independent** PRP key and the result then used as the CBC IV. Let's see what goes wrong if one encrypts the nonce with the **same** PRP key as the key used for CBC encryption.   
  
Let *F*:*K*×{0,1}ℓ→{0,1}ℓbe a secure PRP with, say, ℓ=128. Let *n* be a nonce and suppose one encrypts a message *m* by first computing *IV*=*F*(*k*,*n*) and then using this IV in CBC encryption using *F*(*k*,⋅). Note that the same key *k* is used for computing the IV and for CBC encryption. We show that the resulting system is not nonce-based CPA secure.   
  
The attacker begins by asking for the encryption of the two block message *m*=(0ℓ,0ℓ) with nonce *n*=0ℓ. It receives back a two block ciphertext (*c*0,*c*1). Observe that by definition of CBC we know that *c*1=*F*(*k*,*c*0). Next, the attacker asks for the encryption of the one block message *m*1=*c*0⨁*c*1 with nonce *n*=*c*0. It receives back a one block ciphertext *c*′0.   
  
What relation holds between *c*0,*c*1,*c*′0?   Note that this relation lets the adversary win the nonce-based CPA game with advantage 1.

*c*1=*c*0

*c*1=*c*′0

*c*′0=*c*0⨁1ℓ

*c*0=*c*′0

First, m=(0l,0l) with n=0l🡺(c0,c1):

IV=F(k,0l),c0=F(k,IV XOR 0l)=F(k,IV),c1= *F*(*k*,*c*0) XOR 0l= *F*(*k*,*c*0)

Then,m1=c0 XOR c1 with n=c0🡺c’0:

IV1=F(k,c0)=c1, c’0=F(k,(c1 XOR co XOR c1))=F(k,c0)=c1

This follows from the definition of CBC with an encrypted nonce as defined in the question.

### Question 6

Let *m* be a message consisting of ℓ AES blocks (say ℓ=100). Alice encrypts *m* using CBC mode and transmits the resulting ciphertext to Bob. Due to a network error, ciphertext block number ℓ/2 is corrupted during transmission. All other ciphertext blocks are transmitted and received correctly. Once Bob decrypts the received ciphertext, how many plaintext blocks will be corrupted?

0

2

ℓ/2

1+ℓ/2

ℓ

Take a look at the CBC decryption circuit. Each ciphertext blocks affects only the current plaintext block and the next.

**Question 7**

Let *m* be a message consisting of ℓ AES blocks (say ℓ=100). Alice encrypts *m* using randomized counter mode and transmits the resulting ciphertext to Bob. Due to a network error, ciphertext block number ℓ/2 is corrupted during transmission. All other ciphertext blocks are transmitted and received correctly. Once Bob decrypts the received ciphertext, how many plaintext blocks will be corrupted?

1+ℓ/2

ℓ

0

ℓ/2

1

Take a look at the counter mode decryption circuit. Each ciphertext block affects only the current plaintext block.

### Question 8

Recall that encryption systems do not fully hide the **length** of transmitted messages. Leaking the length of web requests [has been used](http://research.microsoft.com/apps/pubs/default.aspx?id=119060) to eavesdrop on encrypted HTTPS traffic to a number of web sites, such as tax preparation sites, Google searches, and healthcare sites. Suppose an attacker intercepts a packet where he knows that the packet payload is encrypted using AES in CBC mode with a random IV. The encrypted packet payload is 128 bytes. Which of the following messages is plausibly the decryption of the payload:

'If qualified opinions incline to believe in the exponential conjecture, then I think we cannot afford not to make use of it.'

'In this letter I make some remarks on a general principle relevant to enciphering in general and my machine.'

'The most direct computation would be for the enemy to try all 2^r possible keys, one by one.'

'The significance of this general conjecture, assuming its truth, is easy to see. It means that it may be feasible to design ciphers that are effectively unbreakable.'

len(1)=124; len(2)=108; len(3)=92; len(4)=165

The length of the string is 108 bytes, which after padding becomes 112 bytes, and after prepending the IV becomes 128 bytes.

### Question 9

Let *R*:={0,1}4 and consider the following PRF *F*:*R*5×*R*→*R* defined as follows:  
  
*F*(*k*,*x*):=⎧⎩⎨⎪⎪⎪⎪⎪⎪*t*=*k*[0] for i=1 to 4 doif (*x*[*i*−1]==1)*t*=*t*⊕*k*[*i*] output *t*   
  
That is, the key is *k*=(*k*[0],*k*[1],*k*[2],*k*[3],*k*[4]) in *R*5 and the function at, for example, 0101 is defined as *F*(*k*,0101)=*k*[0]⊕*k*[2]⊕*k*[4].   
  
For a random key *k* unknown to you, you learn that   
*F*(*k*,0110)=0011  and  *F*(*k*,0101)=1010  and  *F*(*k*,1110)=0110 .   
What is the value of *F*(*k*,1101)?    Note that since you are able to predict the function at a new point, this PRF is insecure.

Solution:

F(k,0110)=k[0] XOR k[2] XOR k[3]=0011

F(k,0101)=k[0] XOR k[2] XOR k[4]=1010

F(k,1110)=k[0] XOR k[1] XOR k[2] XOR k[3]=0110

F(k,1101)=k[0] XOR k[1] XOR k[2] XOR k[4]=F(k,0110) XOR F(k, 1110) XOR F(k,0101)=0101 XOR 1010=1111

# Programming Questions

**In this project you will implement two encryption/decryption systems, one using AES in CBC mode and another using AES in counter mode (CTR). In both cases the 16-byte encryption IV is chosen at random and is *prepended* to the ciphertext. For CBC encryption we use the PKCS5 padding scheme discussed**[**in class**](https://class.coursera.org/crypto-014/lecture/view?lecture_id=20)**(13:50).   
While we ask that you implement both encryption and decryption, we will only test the decryption function. In the following questions you are given an AES key and a ciphertext (both are**[**hex encoded**](http://en.wikipedia.org/wiki/Hexadecimal)**) and your goal is to recover the plaintext and enter it in the input boxes provided below.   
For an implementation of AES you may use an existing crypto library such as [PyCrypto](https://www.dlitz.net/software/pycrypto/) (Python),** [**Crypto++**](http://www.cryptopp.com/)**(C++), or any other. While it is fine to use the built-in AES functions, we ask that as a learning experience you implement CBC and CTR modes yourself.**

### Question 1

* CBC key: 140b41b22a29beb4061bda66b6747e14
* CBC Ciphertext 1:   
  4ca00ff4c898d61e1edbf1800618fb2828a226d160dad07883d04e008a7897ee2e4b7465d5290d0c0e6c6822236e1daafb94ffe0c5da05d9476be028ad7c1d81

Solution: Basic CBC mode encryption needs padding.

### Question 2

* CBC key: 140b41b22a29beb4061bda66b6747e14
* CBC Ciphertext 2:  
  5b68629feb8606f9a6667670b75b38a5b4832d0f26e1ab7da33249de7d4afc48e713ac646ace36e872ad5fb8a512428a6e21364b0c374df45503473c5242a253

Solution: Our implementation uses rand. IV

### Question 3

* CTR key: 36f18357be4dbd77f050515c73fcf9f2
* CTR Ciphertext 1:   
  69dda8455c7dd4254bf353b773304eec0ec7702330098ce7f7520d1cbbb20fc388d1b0adb5054dbd7370849dbf0b88d393f252e764f1f5f7ad97ef79d59ce29f5f51eeca32eabedd9afa9329

Solution: CTR mode lets you build a stream cipher from a block cipher.

### Question 4

* CTR key: 36f18357be4dbd77f050515c73fcf9f2
* CTR Ciphertext 2:   
  770b80259ec33beb2561358a9f2dc617e46218c0a53cbeca695ae45faa8952aa0e311bde9d4e01726d3184c34451

Solution: Always avoid the two time pad!

Notes:

How to use the pycrypto2.6.1 tools:

An example usage of an encryption algorithm (AES, in this case) is:

>>> from Crypto.Cipher import AES

>>> obj = AES.new('This is a key456', AES.MODE\_ECB)

>>> message = "The answer is no"

>>> ciphertext = obj.encrypt(message)

>>> ciphertext

'o\x1aq\_{P+\xd0\x07\xce\x89\xd1=M\x989'

>>> obj2 = AES.new('This is a key456', AES.MODE\_ECB)

>>> obj2.decrypt(ciphertext)

'The answer is no'

As of PyCrypto 2.1.0, PyCrypto provides an easy-to-use random number

generator:

>>> from Crypto import Random

>>> rndfile = Random.new()

>>> rndfile.read(16)

'\xf7.\x838{\x85\xa0\xd3>#}\xc6\xc2jJU'

A stronger version of Python's standard "random" module is also

provided:

>>> from Crypto.Random import random

>>> random.choice(['dogs', 'cats', 'bears'])

'bears'