Question 3:

1. Consider EP, the expansion permutation, find an inverse contraction permutation. I.E. Find a function that takes 8 bits down to 4 and inverts EP. Note that these are not unique. Implement this function EPinv as in the example Sage code.
2. Take the function f\_K from the example Sage code and modify it so that instead of calling the SBoxes, it calls EPinv after the round key is XORed in. Rename the modified function f\_K\_NoSBox.
3. Modify the functions SDESEncrypt [see example Sage code], and SDESDecrypt [see question 1] so that they the call f\_K\_NoSBox (from part b). Call the new functions SDESEncryptNoSBox and SDESDecryptNoSBox.
4. Do these new functions function as Encrypt/Decrypt functions of each other? I.E. Will SDESDecryptNoSBox give you back the input of SDESEncryptNoSBox (given that they are using the same key)?
5. Does SDESEncryptNoSBox make a good Encryption function, why or why not? [Hint: can you mount a known or chosen plaintext attack on the functions you wrote in part (d)?]

Solutions:

Question 3:

1. Consider EP, the expansion permutation, find an inverse contraction permutation. I.E. Find a function that takes 8 bits down to 4 and inverts EP. Note that these are not unique. Implement this function EPinv as in the example Sage code.

#

# An inverse function for EP permutation:

# Note that their can be many inverse

# Permutations for EP

#

EPinv\_data = [8, 5, 4, 1];

def EPinv(block):

return ApplyPermutation(block, EPinv\_data);

1. Take the function f\_K from the example Sage code and modify it so that instead of calling the SBoxes, it calls EPinv after the round key is XORed in. Rename the modified function f\_K\_NoSBox.

def f\_K\_NoSBox(block, K):

left\_block = LeftHalfBits(block);

right\_block = RightHalfBits(block);

temp\_block1 = EP(right\_block);

temp\_block2 = XorBlock(temp\_block1, K);

temp\_block3 = EPinv(temp\_block2)

temp\_block4 = P4(temp\_block3)

temp\_block5 = XorBlock(temp\_block4, left\_block);

output\_block = concatenate(temp\_block5, right\_block)

return output\_block;

#

# Simplified DES with no SBoxes Decrypt function

#

def SDESDecrypt\_NoSBox(plaintext\_block, K):

(K1, K2) = SDESKeySchedule(K);

temp\_block1 = IP(plaintext\_block);

temp\_block2 = f\_K\_NoSBox(temp\_block1, K2);

temp\_block3 = SW(temp\_block2);

temp\_block4 = f\_K\_NoSBox(temp\_block3, K1);

output\_block = IPinv(temp\_block4);

return output\_block;

1. Modify the functions SDESEncrypt [see example Sage code], and SDESDecrypt [see question 1] so that they the call f\_K\_NoSBox (from part b). Call the new functions SDESEncryptNoSBox and SDESDecryptNoSBox.

#

# Simplified DES with no SBoxes Decrypt function

#

def SDESDecrypt\_NoSBox(plaintext\_block, K):

(K1, K2) = SDESKeySchedule(K);

temp\_block1 = IP(plaintext\_block);

temp\_block2 = f\_K\_NoSBox(temp\_block1, K2);

temp\_block3 = SW(temp\_block2);

temp\_block4 = f\_K\_NoSBox(temp\_block3, K1);

output\_block = IPinv(temp\_block4);

return output\_block;

#

# Simplified DES with no SBox Encrypt Function

#

def SDESEncrypt\_NoSBox(plaintext\_block, K):

(K1, K2) = SDESKeySchedule(K);

temp\_block1 = IP(plaintext\_block);

temp\_block2 = f\_K\_NoSBox(temp\_block1, K1);

temp\_block3 = SW(temp\_block2);

temp\_block4 = f\_K\_NoSBox(temp\_block3, K2);

output\_block = IPinv(temp\_block4);

return output\_block;

1. Do these new functions function as Encrypt/Decrypt functions of each other? I.E. Will SDESDecryptNoSBox give you back the input of SDESEncryptNoSBox (given that they are using the same key)? If your answer is no, then given an example of a Key and Plaintext block that will not decrypt to the same value after encrypted. If your answer is yes either give a proof or provide some evidence to back up your assertion.  
     
   The new functions are keyed inverses. This can be easily seen by enumerating over all 218 possible plaintext or key pairs, which should take mere minutes using a modern computer. Or this can be seen rigorously by noting that f\_K\_NoSBox applied twice is the identity. This can be seen by tracing through the algorithm (it may be helpful to draw the new algorithm out as the diagrams for the original *fK*. )
2. Ignoring all aspects of key size, does SDESEncryptNoSBox make a good Encryption function, why or why not? [Hint: can you mount a known or chosen plaintext attack on the functions you wrote in part (d)?]  
     
   No, SDESEncryptNoSBox is not an effective encryption function. This is a fairly open ended question. However a few things are: It has no nonlinearity, so it is vulnerable to linear cryptanalysis. This can be seen noting that by taking the SBox out, the whole function becomes a linear transformation of the input block XORed with a linear transformation of the key. Furthermore, only 4 of the bits of each round key are used, so many keys will decrypt the same message.