Question 1:

The purpose of this question is to become more familiar with the algorithm for generating the Simplified AES S-Box. Each part of this problem is to write one part of the algorithm in Sage, and in the last part put them all together. The construction closely follows the description of the algorithm specification in the text.

1. Consider the positive integers between 0 and 15 (inclusive) as 4 bit strings, so that 3 is 1100 (this ordering is known as little endian.) Define the mapping from {0,1,2,…,15} to GF(24) by mapping the element with bit string b0b1b2b3 to the element b0+ b1a + b2a2 + b3 a3 of GF(24). The following snippet of sage code sets F to the finite field with two elements L the finite field with 16 elements (extension of F with modulus a4 + a + 1) and primitive element a (we use a here because a is a special value in Sage.) And V is the vector space of dimension 4 over F (you can think of this as 4 bit strings with addition defined on them.)  
     
     
     
     
     
   As in the example code for Simplified AES we can map a bit list b to the corresponding element of L by L(V(b)). Write a sage function that maps a positive integer in {0,1,2,…,15} to an element of L. (Hint: If x is a sage Integer z.bits() is a little-endian list of the bits of z, however z only has as many elements as the bit length of z. So, for example if z is 0, this function returns an empty list. However, L(V(b)) only works if b is a bit list of length 4. You will have to work around this.)

F = GF(2);

L.<a> = GF(2^4);

V = L.vector\_space();

1. Use the function from part (a) to write a Sage function to initialize a 2 dimensional array (either a list of lists or a matrix over L) so that the element at position (r,c) is the element of GF(24) mapped to by 4r+c.
2. Write a function that takes M, a 2 dimensional array of elements in L = GF(24) (either a list of lists or a matrix over L) and maps each nonzero element to its inverse and 0 to 0. I.e. Your function should return M’ the 2 dimensional array of elements in L where the element at row r and column c of M’ is the inverse of Mrc (or 0 if Mrc = 0). (Hint: If z is a nonzero element of L, in sage, z^(-1) is the multiplicative inverse of z. If z is zero, this will raise an error.)
3. Write a function that takes a 2 dimensional array of elements in L = GF(24) and for each each element converts it to an element of V and applies the Linear transformation in step 4 of the S-Box generation algorithm. Then returns the resulting 2 dimensional array. The sage code to initialize A and b would be:  
     
     
     
     
     
     
     
     
   And the linear equation is A\*v+b. Then take the resulting element of V and map it back to an element of L (if v is an element of V, then L(v) is the corresponding element of L.) (Hint: If z is an element of L and v = z.vector(), the linear transformation as is defined in the algorithm expects that the bits of v are reverse of how Sage orders them. To deal with this, you will either have to reverse the bits of v, or appropriately modify the A matrix.)

A = Matrix(F, [

[ 1, 0, 1, 1],

[ 1, 1, 0, 1],

[ 1, 1, 1, 0],

[ 0, 1, 1, 1] ]);

b = V([1, 0, 0, 1]);

1. Use the functions you just wrote to write a function that initializes the SBox matrix for simplified DES. Check your answer vs the SBoxes of the simplified AES function in the book.

Solution:

The purpose of this question is to become more familiar with the algorithm for generating the Simplified AES S-Box. Each part of this problem is to write a part of the algorithm in sage, and in the last part we put them all together. The Construction closely follows the description of the algorithm specification in the text.

1. Consider the positive integers between 0 and 15 (inclusive) as 4 bit strings, so that 3 is 1100 (this ordering is known as little endian.) Define the mapping from {0,1,2,…,15} to GF(24) by mapping the element with bit string b0b1b2b3 to the element b0+ b1x + b2x2 + b3 x3 of GF(24). The following snippet of sage code sets F to the finite field with two elements L the finite field with 16 elements (extension of F with modulus x4 + x + 1) and primitive element a (we use a here because x is a special value in Sage.) And V is the vector space of dimension 4 over F (you can think of this as 4 bit strings with addition defined on them.)  
     
     
     
     
     
   As in the example code for Simplified AES we can map a bit list b to the corresponding element of L by L(V(b)). Write a sage function that maps a positive integer in {0,1,2,…,15} to an element of L. (Hint: If x is a sage Integer z.bits() is a little-endian list of the bits of z, however z only has as many elements as the bit length of z. So, for example if z is 0, this function returns an empty list. However, L(V(b)) only works if b is a bit list of length 4. You will have to work around this.)

def SAES\_NibToGF16(x):

x\_bit\_list = x.bits();

x\_bit\_list.extend([0,0,0,0]);

bit\_list = [x\_bit\_list[j] for j in xrange(4)];

output = L(V(bit\_list));

return output;

F = GF(2);

L.<a> = GF(2^4);

V = L.vector\_space();

1. Use the function from part (a) to write a Sage function to initialize a 2 dimensional array (either a list of lists or a matrix over L) so that the element at position (r,c) is the element of GF(24) mapped to by 4r+c.

def SAES\_InitSBoxMatrix():

row\_list = [];

for j in xrange(4):

row = [SAES\_NibToGF16(4\*j+k) for k in xrange(4)];

row\_list.append(row);

return row\_list;

1. Write a function that takes M, a 2 dimensional array of elements in L = GF(24) (either a list of lists or a matrix over L) and maps each nonzero element to its inverse and 0 to 0. I.e. Your function should return M’ the 2 dimensional array of elements in L where the element at row r and column c of M’ is the inverse of Mrc (or 0 if Mrc = 0). (Hint: If z is a nonzero element of L, in sage, z^(-1) is the multiplicative inverse of z. If z is zero, this will raise an error.)

def SAES\_MapToInverse(M):

next\_M = [];

for j in xrange(4):

row = []

for k in xrange(4):

Mjk = M[j][k]

if(Mjk != 0):

row.append(Mjk^-1);

else:

row.append(0);

next\_M.append(row);

return next\_M;

1. Write a function that takes a 2 dimensional array of elements in L = GF(24) and for each each element converts it to an element of V and applies the Linear transformation in step 4 of the S-Box generation algorithm. Then returns the resulting 2 dimensional array. The sage code to initialize A and b would be:  
     
     
     
     
     
     
     
     
   And the linear equation is A\*v+b. Then take the resulting element of V and map it back to an element of L (if v is an element of V, then L(v) is the corresponding element of L.) (Hint: If z is an element of L and v = z.vector(), the linear transformation as is defined in the algorithm expects that the bits of v are reverse how sage exports them. To deal with this, you will either have to reverse the bits of v, or appropriately modify the A matrix.)

A = Matrix(F, [

[ 1, 0, 1, 1],

[ 1, 1, 0, 1],

[ 1, 1, 1, 0],

[ 0, 1, 1, 1] ]);

b = V([1, 0, 0, 1]);

def SAES\_LinearTransformElements(M):

next\_M = [];

for j in xrange(4):

row = [L(A.transpose()\*V(M[j][k]) + b) for k in xrange(4)];

next\_M.append(row);

return next\_M;

A = Matrix(F, [

[ 1, 0, 1, 1],

[ 1, 1, 0, 1],

[ 1, 1, 1, 0],

[ 0, 1, 1, 1] ]);

b = V([1, 0, 0, 1]);

1. Use the functions you just wrote to write a function that initializes the SBox matrix for simplified DES. Check your answer vs the SBoxes of the simplified AES function in the book.

def SAES\_ComputeSBoxMatrix():

M0 = SAES\_InitSBoxMatrix();

M1 = SAES\_MapToInverse(M0);

M2 = SAES\_LinearTransformElements(M1);

SBox\_matrix\_output = Matrix(L, M2);

return SBox\_matrix\_output;