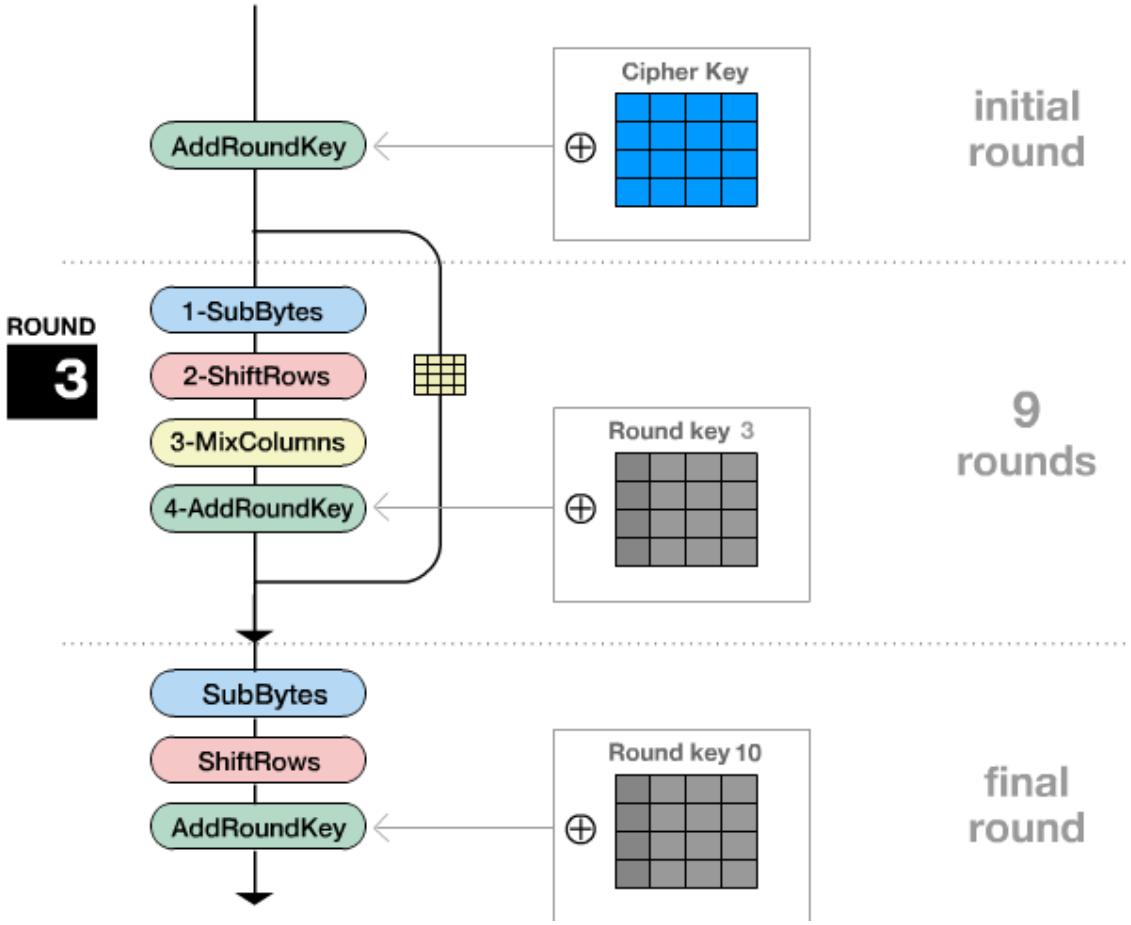
James Le
11/19/2012
100068794

AES Project

How to test: For testing look at TEST CASE 2 (line 482) and manually change the Plaintext[] and EnterKey[] to the requested hex values to test with. Make sure that there is a 0x in front of the hex values or else it won't work. There is an example in TEST CASE 1 (line 461) of how values should be inputted.

Implementation:

For the implementation of the AES project I created different functions that would emulate the steps that are required to have AES run correctly. I followed a specification read me from http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf that lists what each function should do. It also provided the expected inputs and outputs after each step within Appendix B. I decided to implement the AES project in C++ by using vectors as it is easier to manage than using pointers. The following diagram is what I followed for rounds 1 to 9 of the AES implementation.



James Le
11/19/2012
100068794

AES Project

The implementation of AES starts in the main where a key and plaintext are specified. Once both are specified they are converted into a 4 by 4 block array as every calculation are done by blocks. The function that does this is the convertToByteArray() which takes in an array and returns a vector. Within the function it will also pad with zeros if the plaintext is not the correct 16 byte size. The function looks like so:

```
vector< vector<unsigned char> > convertToByteArray(const unsigned char
array[], bool lastByte, int type){
	vector< vector<unsigned char> > Block(blockSize, vector<unsigned
char>(blockSize));
	int increment = 0;
	if(lastByte != 0)
		increment = LastByteTouched;
	for(int i = 0; i < blockSize; i++){
		for (int j = 0; j < blockSize; j++){
			if((increment >= (PlainTextSize)) && (type == 1)){
				Block[j][i] = 0x00;
			}
			else{
				Block[j][i] = array[increment++];
			}
		}
	}
	if(lastByte != 0)
		LastByteTouched = increment;
	return Block;
	}
```

Throughout the implementation of AES everything is done by passing and using vectors as this made it a lot easier to manage.

A for loop was created to loop through all the rounds in the AES with condition statements that specified which round was to do what.

After the key and plaintext are converted to their Block forms the next thing to do was to do key expansion. The function keyExpansion() does the expanding by taking in the initial key block and an S-Box which was created statically, by hardcoding it. I hardcoded the S-box instead of figuring out how to automate the generation of the S-box is because it made things easier to code. Another important thing to do was to have the Rcon, which is calculated each time keyExpansion ran, but of course the Rcon could have been created as a static 2D array that held

values specified, but choose not to. After the required values have been calculated and the inputs have been passed, the function will follow the AES standard for expanding the key schedule. There are also separate functions that were created to do rotword and the subBytes for the key expansion. After the calculations, keyExpansion() will return the fully expanded key schedule.

Once key Expansion was done, the next thing to do was to add the round key for round 0 which was just xoring the initial input with the first four columns of the key schedule. To do this a function addRoundKey() would take in the input state and the current round key block which is held by a global counter that keeps track of which round AES is currently in. After round 0, rounds 1 to 9 will use the functions subBytes(), shiftRows(), getRoundkey(), mixColumns() and addRoundKey(). The first two functions are self-explanatory. For shiftRows() there is actually a another function that it calls which is oneShift(), it takes in the current row passed as a vector where it will shift if once and returned. shiftRows() just specifies how many times a row will shift which it calls oneShift() based on the row. The oneShift() also acts as the RotWord used for the key Expansion. The getRoundKey() function will get the next keyblock within the key schedule. The round key is based on the current round of the AES algorithm specified by a global variable that keeps track of the current round. The mixColumns() function just does matrix multiplication between a constant matrix specified by the AES standard and the inputted matrix passed to the function. However the calculations done are done in binary with a restraint of max one byte which is an integer of 255 so a function getr() will do the calculations by value based on that restraint. The values calculated by getr() are within the most inner loop where the other two loops process that information for a new matrix to be calculated. The code looks like so:

```
vector<vector<unsigned char> > mixColumns(vector< vector<unsigned char> >
inputState){
	vector<vector<unsigned char> > state(blockSize, vector<unsigned
char>(blockSize)); //make sure to declare sizes first
	unsigned char rValues[4];
	//unsigned char r[4];
	unsigned char mixColumn[4][4] = {
			{0x02, 0x03, 0x01, 0x01},
			{0x01, 0x02, 0x03, 0x01},
			{0x01, 0x01, 0x02, 0x03},
			{0x03, 0x01, 0x01, 0x02}
	};
```

James Le
11/19/2012
100068794

AES Project

```
	//rule: multiplication of a value by x(i.e 02) is a 1-bit left shift
followed by XOR with 1B(00011011)
	//Only used when leftmost bit is a 1 so >> 7 times;
	for(int k = 0; k < blockSize; k++){
		for(int i =0; i<blockSize; i++){
			for(int j=0; j < blockSize; j++){
				//Values are stored than calculated out of this loop
				rValues[j] = getr(inputState[j][k], mixColumn[i][j]);
			}
			state[i][k] = rValues[0] ^ rValues[1] ^ rValues[2] ^
rValues[3];
		}
	}
	return state;
```

After mix columns the addRoundKey() is very simple as you will take in a input state and the current round block and return the xored value between those two. After rounds 1-9 are done, round 10 is done without the roundKey functions.

Test Case #1:

For test one I specified a plaintext and a provided key from the following website, which than I matched if the output was the same as my output. If they are correct that means that my implementation was correctly implemented. The website that I used is http://testprotect.com/appendix/AEScalc to match my code. I also used the Appendix B of the fips-197.pdf to follow through my code, so based on the output I had for each round I matched it with that of the pdf, step by step.

```
/*############################TEST CASE 1 Start################################*/
	/*fips-197.pdf (http://csrc.nist.gov/publications/fips/fips197/fips-
197.pdf)
	Appendix B – Cipher Example
	PlainText: 32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34
	Cipher Key: 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c
	*/
	/*Used the website http://testprotect.com/appendix/AEScalc to test if
it works
	copy and paste the above PlainText and Cipher key into the two
associated boxes.
	*/
printf("Plain Text\n");
	const unsigned char Plaintext[] = {0x32, 0x43, 0xf6, 0xa8, 0x88, 0x5a,
0x30, 0x8d, 0x31, 0x31, 0x98, 0xa2, 0xe0, 0x37, 0x07, 0x34};
	printf("%s\n", Plaintext);
	PlainTextSize = sizeof(Plaintext);
	const unsigned char EnterKey[] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae,
0xd2, 0xa6, 0xab, 0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};
	for(int i = 0; i < KeySize; i++){
```

James Le
11/19/2012
100068794

AES Project

```
            key[i] = EnterKey[i];
      }
      /*#########################TEST CASE 1 End############################*/
```

Based on the above Plaintext and Key:
Based on the two blocks, Round 0 output is displayed and matched with the one with the pdf for round 1

```
INITIAL KEY
2b 28 ab 9
7e ae f7 cf
15 d2 15 4f
16 a6 88 3c

HEX VERSION OF PLAINTEXT
32 88 31 e0
43 5a 31 37
f6 30 98 7
a8 8d a2 34

Round 0

19 a0 9a e9
3d f4 c6 f8
e3 e2 8d 48
be 2b 2a 8
```

Round Key Value

input

32	88	31	e0
43	5a	31	37
f6	30	98	07
a8	8d	a2	34

⊕

2b	28	ab	09
7e	ae	f7	cf
15	d2	15	4f
16	a6	88	3c

=

19	a0	9a	e9
3d	f4	c6	f8
e3	e2	8d	48
be	2b	2a	08

I do that for the next 9 rounds until round 10 where the output is shown.

This shows what is specified by the pdf.

10

eb	59	8b	1b
40	2e	a1	c3
f2	38	13	42
1e	84	e7	d2

e9	cb	3d	af
09	31	32	2e
89	07	7d	2c
72	5f	94	b5

e9	cb	3d	af
31	32	2e	09
7d	2c	89	07
b5	72	5f	94

⊕

d0	c9	e1	b6
14	ee	3f	63
f9	25	0c	0c
a8	89	c8	a6

=

output

39	02	dc	19
25	dc	11	6a
84	09	85	0b
1d	fb	97	32

Based on the pdf, my expected input and output are correct for round 10.

```
Round 9

eb 59 8b 1b
40 2e a1 c3
f2 38 13 42
1e 84 e7 d2

Round 10

39 2 dc 19
25 dc 11 6a
84 9 85 b
1d fb 97 32
```

James Le
11/19/2012
100068794

AES Project

Test Case #2

This test is similar to test case 1 in that the website lists the steps and expected inputs and outputs for each round. *Provided by the TA*

```
	/*######################TEST CASE 2 Start##############################*/
	//Used the website http://people.eku.edu/styere/Encrypt/JS-AES.html to
see whats happening.
printf("Plain Text\n");
	const unsigned char Plaintext[] = "THIS IS A TEST";
	printf("%s\n", Plaintext);
	PlainTextSize = sizeof(Plaintext);
	const unsigned char EnterKey[] = {0xf6, 0xcc, 0x34, 0xcd, 0xc5, 0x55,
0xc5, 0x41, 0x82, 0x54, 0x26, 0x02, 0x03, 0xad, 0x3e, 0xcd };
	for(int i = 0; i < KeySize; i++){
		key[i] = EnterKey[i];
	}
/*############################TEST CASE 2 End################################*/
```