AES Challenge – Extra Credit

Due: Friday, October 12

This handout describes the extra-credit challenge assignment that involves implementing AES. What you need to do to complete this assignment is to implement AES encryption with a 128-bit key. The following amounts of extra credit are available, as points added to your total cumulative assignment points:

A working implementation:	25 points
The most elegant implementation:	15 points
The fastest implementation:	20 points

Email me your solution before midnight on October 12 in order to be considered. I will first test all submissions by running test vectors through in order to see that they give the correct answers. Then I will look at the code of the working submissions to see which I think is the most cleanly or elegantly implemented (and comments count too!). Finally, I'll run a "speed test" of all working implementations, where I encrypt around 100,000 blocks of plaintext with each implementation. The implementation that runs fastest gets those extra credit points.

In order to be able to test these, it is important that everyone use the same interface. To implement this, write a class called `AESCipher` that has two methods (at least – if you want to add more, feel free, as long as they make sense). The first method will set the key for the block cipher (call this `setKey`), and the second will encrypt a block of plaintext (call this `encrypt`). You can call `encrypt` multiple times without having to call `setKey` again, so if you can do any pre-computations with the key in `setKey` then the subsequent encryptions will run faster (hint, hint...).

There is sample code on the following page showing how these methods should be defined. The actual methods will be replaced, of course, but the method signatures (parameters, etc.) in your code should be *exactly* as shown in the example. You can download this code from the class web page, so that you can replace the method bodies while leaving the interfaces the same.

The class web page will have a few "test vectors" that you can use to see if your code works properly. Make sure you use these to make sure you did things right!

```
class AESCipher {
public:
    static const int KEYLEN = 16;
    static const int BLOCKLEN = 16;

private:
    unsigned char keyCopy[KEYLEN];

public:
    void setKey(unsigned char key[]) {
        for (int i=0; i<KEYLEN; i++) {
            keyCopy[i] = key[i];
        }
    }

    void encrypt(unsigned char inBytes[], unsigned char outBytes[]) {
        for (int i=0; i<BLOCKLEN; i++)
            outBytes[i] = inBytes[i] ^ keyCopy[i];
    }
};


int main()
{
    unsigned char key[AESCipher::KEYLEN] =
        {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};
    unsigned char plain[AESCipher::BLOCKLEN] =
        {2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17};
    unsigned char cipher[AESCipher::BLOCKLEN];

    AESCipher aesObject;

    aesObject.setKey(key);
    aesObject.encrypt(plain, cipher);

    for (int i=0; i<AESCipher::BLOCKLEN; i++)
        printf("%02x ", (unsigned int)cipher[i]);
    putchar('\n');

    return 0;
}
```