AES CTR Encryption in C

by Guru | May 7, 2012 | Programming

Encryption is one of the best tools at protecting data when it comes to computer security. There are many forms of encryption as well. One of the forms that I encountered recently in my work is AES CTR encryption. I am sure you have heard of AES encryption, but what exactly is AES CTR?

AES CTR

CTR is a counter mode for AES encryption. It is also known as ICM and SIC.

In AES encryption you have what is called an Initializing Vector, or IV for short. This is a 128-bit input that is usually randomized. In CTR mode the IV has two parts. The first 8 bytes is the regular randomized IV. The last 8 bytes is a counter. This counter is a 0 index of the number of 128-bit blocks you are inside the encrypted information. For example. If you are encrypting 512 bits of information (64 bytes), the start position of 0 bytes into the information would have a counter of 0. 16 bytes in, you would have a counter of 1. 32 bytes in your counter would be up to 2. So on an so on until you are at the end if your information. Unlike normal AES encryption this encryption can be seek-able through the information. You don’t have to decrypt all of the bytes to get some information in the middle.

The way encryption works in AES CTR mode is that we generate some random bits with the encryption key provided and the IV. With these random bits we then XOR them with our string. This creates a randomized text. To decrypt them we just simply XOR the text with the same exact random bits that we generated with the encryption key and the IV.

Let’s look at this example. There is two people, person A and person B. They both share a random string of text that no one else has. Let’s have this random text be 10100011011011011. If person A wants to send person B a message all they have to do is to XOR their message with their random text.

Let’s have person A’s message be 10011010101010100.

If we XOR the message with the random text we get the following string. 00111001110001111

We then send this string to person B. If person B XORs the string with his random string he will get the original message from person A of 10011010101010100.

Encrypting a file and decrypting a file are the same steps. The only differences in our code example would be during decryption we set the IV from the file. During encryption we generate the IV randomly.

So everyone wants a code example right? What’s the point of knowing about the method without being able to implement it.

For this example we will be using OpenSSL’s AES implementation in their cryptography library.

Download the library:

Windows

For windows you can find the OpenSSL download link here: https://www.openssl.org/related/binaries.html

Linux

If you’re using a debain based version of linux you can download the library with this command: “sudo apt-get install libssl-dev”

OS X.

You already have this library installed. It can be found in /usr/include/openss/ and /usr/bin/openssl/

The language that we will be using will be C. The code is not platform specific. We will be writing the code in Linux using a text editor and the GCC compiler.

Demo: Encrypt/Decrypt files with OpenSSL AES CTR mode.

- Note: Code example uses partial code from: https://stackoverflow.com/questions/3141860/aes-ctr-256-encryption-mode-of-operation-on-openssl

Let’s start with our includes.

We will need to include 4 files for this example.

```
#include
#include
#include
#include
```

We will also need a structure to maintain our ivector, ecount, and num.

```
struct ctr_state
{
    unsigned char ivec[AES_BLOCK_SIZE];
    unsigned int num;
    unsigned char ecount[AES_BLOCK_SIZE];
};
```

The Ivector is the only piece of information used by our program that we care about. Num and ecount are variables that we have to pass into our encryption function that we don’t ever need to care about.

*Note AES_BLOCK_SIZE is defined to be the integer value of 16. This is the number of bytes in the 128-bit block for AES.

We will also need two file types declared for encrypting and decrypting.

```
FILE *readFile;
FILE *writeFile;
```

We will also need to set our encryption key.

```
AES_KEY key;
```

Some other inforation that we will need is to know how many bytes we read/wrote, the data that we read/wrote to the file, the IV that we read from the file, and our state for the encryption of a ctr_state struct.

```
int bytes_read, bytes_written;
unsigned char indata[AES_BLOCK_SIZE];
unsigned char outdata[AES_BLOCK_SIZE];
unsigned char iv[AES_BLOCK_SIZE];
struct ctr_state state;
```

It is helpful to have a function which initializes the IV setting all the values to be 0 except the first 8 which will be the random input.

```
int init_ctr(struct ctr_state *state, const unsigned char i
{
    /* aes_ctr128_encrypt requires 'num' and 'ecount' set t
    * first call. */
    state->num = 0;
    memset(state->ecount, 0, AES_BLOCK_SIZE);

    /* Initialise counter in 'ivec' to 0 */
    memset(state->ivec + 8, 0, 8);

    /* Copy IV into 'ivec' */
    memcpy(state->ivec, iv, 8);
}
```

Let’s work on our encryption function.

Our function can be described as this

```
void fencrypt(char* read, char* write, const unsigned char*
{
```

Read is the file name plus location that we are reading from to encrypt.

Write is the file name plus location that we are writing the encrypted information to.

enc_key is the encryption key used to encrypt the file. *Note, this must be 16 bytes long. This is the “password” to the file.

The first thing we need to do is to create an IV with random bytes. OpenSSL library has a function to generate random bytes into an array. It is found in rand.h.

```
    if(!RAND_bytes(iv, AES_BLOCK_SIZE))
    {
        fprintf(stderr, "Could not create random bytes.");
        exit(1);
    }
```

Now we need to open our reading/writing files and make sure that they can be used.

```
    readFile = fopen(read,"rb"); // The b is required in wi
    writeFile = fopen(write,"wb");
    if(readFile==NULL)
    {
        fprintf(stderr, "Read file is null.");
        exit(1);
    }

    if(writeFile==NULL)
    {
        fprintf(stderr, "Write file is null.");
        exit(1);
    }
```

Now that we have our file in place we need to write out our IV to the file. The IV is not suppose to be secure. Remember that only the first 8 bytes of the IV are even used in this mode of AES encryption.

```
     fwrite(iv, 1, 8, writeFile); // IV bytes 1 - 8
     fwrite("", 1, 8, writeFile); // Fill the last 8 with nu
```

The next step is to set our encryption key.

```
     //Initializing the encryption KEY
     if (AES_set_encrypt_key(enc_key, 128, &key) < 0)
     {
         fprintf(stderr, "Could not set encryption key.");
         exit(1);
     }
```

After we set our encryption key we need to initialize our state structure which holds our IV.

```
     init_ctr(&state, iv); //Counter call
```

Now the fun part. We will go into a continuous loop reading from the file encrypting all the data and writing it out.

```
     while(1)
     {
```

We then need to read 16 bytes from the file into our indata array.

```
         bytes_read = fread(indata, 1, AES_BLOCK_SIZE, readF
```

After we read the bytes we then encrypt them using our AES_ctr128_encrypt function. This is the 128-bit encryption function found in aes.h. Indata is the data we read from the file. Outdata is our array to which the encrypted bytes will be placed. bytes_read is the number of bytes in the indata array to be encrypted. Key is the encryption key that was set using our 16 byte password. State.ivec is the IV used for encryption. The last two variables are not used by us so we don’t need to know about them at all.

```
rypt(indata, outdata, bytes_read, &key, state.ivec, state.ec
```

Now that we encrypted our data into outdata it’s time to write them to a file.

```
        bytes_written = fwrite(outdata, 1, bytes_read, writ
```

If we read less than our block size it probably means we are at the end of the file. We can stop encrypting now.

```
        if (bytes_read < AES_BLOCK_SIZE)
        {
            break;
        }
    }
```

We now need to close our files to as they are no longer needed.

```
    fclose(writeFile);
    fclose(readFile);
}
```

So what changes in the decrypting function?

Basically nothing. The only changes that we do is we set our IV to be the first 16 bytes in the input file. The rest of the code is the exact same.

If you want a further example the code can be found here: https://www.gurutechnologies.net/uploads/martyj/aes_ctr_example.zip

Compile via command line with the following command. “gcc main.c -lm -lcrypto -lssl”

Search

Recent Posts

- Adding CI to your project
- 5 Reasons Why Custom Software is the Future of Business
- Custom Social Media App
- Good UX is All About Confidence
- Cross Platform iOS/Android Development

Categories

- Business Management
- Company News
- Industry News
- Network Administration
- Programming
- Sample Work by Guru
- Software Development
- Tech Support
- Uncategorized
- User Interface Design