Data Encryption Application Whitepaper

Christopher Bird, Software Applications Engineers

Application Origination:
Intel SSG

Introduction

Encryption is important because it allows you to securely protect data that you don't want anyone else to have access to. Encryption has been a trending topic in the security community. As more mobile devices store valuable information than ever before, encryption has become crucial to ensure information security.

This paper introduces data encryption APIs that are available through either Java* or OpenSSL*. Both solutions work on the Android* OS.

We recommend that you try out the features and compile the code as you read through the paper.

Data Encryption Code and Explanations

If you want to encrypt data on Android, you have two options: Java Crypto API and OpenSSL API. We will show you how to encrypt data using both ways.

Java Crypto API

Using Java Crypto API on Android is very straightforward. First, you will need to generate a key for the encryption. There is a KeyGenerator class in the javax.crypto package that can do this for you.

```
        mKey = null;
        try {
                kgen = KeyGenerator.getInstance("AES");
                mKey = kgen.generateKey();

        } catch (NoSuchAlgorithmException e) {
                e.printStackTrace();
        }
```

Then you can use the generated key to encrypt the data file. This can be done by feeding chunks of bytes to an AES Cipher created by javax.crypto.

```
// open stream to read origFilepath. We are going to save encrypted contents
to outfile
        InputStream fis = new FileInputStream(origFilepath);
        File outfile = new File(encFilepath);
        int read = 0;
        if (!outfile.exists())
                outfile.createNewFile();

        FileOutputStream encfos = new FileOutputStream(outfile);
        // Create Cipher using "AES" provider
        Cipher encipher = Cipher.getInstance("AES");
        encipher.init(Cipher.ENCRYPT_MODE, mKey);
        CipherOutputStream cos = new CipherOutputStream(encfos, encipher);

        // capture time it takes to encrypt file
        start = System.nanoTime();
        Log.d(TAG, String.valueOf(start));

        byte[] block = new byte[mBlocksize];

        while ((read = fis.read(block,0,mBlocksize)) != -1) {
                cos.write(block,0, read);
        }
        cos.close();
        stop = System.nanoTime();

        Log.d(TAG, String.valueOf(stop));
        seconds = (stop - start) / 1000000;// for milliseconds
        Log.d(TAG, String.valueOf(seconds));


        fis.close();
```

OpenSSL API

Encrypting data in OpenSSL on Android requires writing native C code that can be accessed in Java through JNI calls. It requires more work, but you will get better performance in return.

First, let's generate the key and the iv.

```
unsigned char cKeyBuffer[KEYSIZE/sizeof(unsigned char)];
unsigned char iv[] = "01234567890123456";

int opensslIsSeeded = 0;
if (!opensslIsSeeded) {
      if (!RAND_load_file("/dev/urandom", seedbytes)) {
            return -1;
      }
      opensslIsSeeded = 1;
}

if (!RAND_bytes((unsigned char *)cKeyBuffer, KEYSIZE )) {
}
```

Then, we can use the generated key (cKeyBuffer) to encrypt a file. Initialize EVP by feeding it your key and iv. Then feed chunks of bytes to the EVP_EncryptUpdate function. The final chunk of bytes from your file will need to be fed to the EVP_EncryptFinal_ex function.

```
if (!(EVP_EncryptInit_ex(e_ctx, EVP_aes_256_cbc(), NULL, cKeyBuffer, iv ))) {
      ret = -1;
      printf( "ERROR: EVP_ENCRYPTINIT_EX\n");
}

// go through file, and encrypt
if ( orig_file != NULL ) {
      origData = new unsigned char[aes_blocksize];
      encData = new unsigned
char[aes_blocksize+EVP_CIPHER_CTX_block_size(e_ctx)]; // potential for
encryption to be 16 bytes longer than original

      printf( "Encoding file: %s\n", filename);

      bytesread = fread(origData, 1, aes_blocksize, orig_file);
      // read bytes from file, then send to cipher
      while ( bytesread ) {


            if (!(EVP_EncryptUpdate(e_ctx, encData, &len, origData,
bytesread))) {
                  ret = -1;
                  printf( "ERROR: EVP_ENCRYPTUPDATE\n");
            }
            encData_len = len;

            fwrite(encData, 1, encData_len, enc_file );
            // read more bytes
            bytesread = fread(origData, 1, aes_blocksize, orig_file);
      }
      // last step encryption
      if (!(EVP_EncryptFinal_ex(e_ctx, encData, &len))) {
            ret = -1;
            printf( "ERROR: EVP_ENCRYPTFINAL_EX\n");
      }
      encData_len = len;

      fwrite(encData, 1, encData_len, enc_file );

      // free cipher
      EVP_CIPHER_CTX_free(e_ctx);
```

Conclusion

By implementing code like the samples described in this paper, you can quickly learn how to use both Java Crypto API and OpenSSL API to encrypt data on Intel® processor-based platforms running Android.

About the author

Christopher Bird is a member of the Intel Software and Solutions Group (SSG), Developer Relations Division, Intel® Atom™ Processor Innovative Technologies Engineering team.