Implementing Elliptic Curve Integrated Encryption Scheme on Android Platforms

Liang Xia

Abstract— **Elliptic curve cryptography (ECC) is one of the strongest cryptography in terms of security level. A 160-bit ECC key is roughly equivalent to a 1024-bit RSA key (NIST).**

Elliptic Curve Integrated Encryption Scheme is a hybrid encrytion scheme that works like static Diffie-Hellman followed by symmetric encryption.

Bouncy Castle is a provider for the Java/C# Cryptography Extension and Java Cryptography Architecture. Spongy Castle is the Android version for Bouncy Castle.

In this paper, we show an implementation of Elliptic Curve Integrated Encryption Scheme (ECIES) using Spongy Castle on Android platforms. The implementation includes elliptic curve key generation, message encryption and decryption.

We run our application on three platforms. The first platform is Android virtual device simulating Samsung Nexus S. The second is Samsung Nexus S having an ARM Cortex-A8 CPU which runs at 1GHz with 512MB RAM. The third platform is Android Mini TV Dual-Core A9 Processor which runs at 1GHz with 1GB RAM.

I. ECIES

ECC is based on the operation of a chosen elliptic curve and a point P on it. We use Weierstrass form as our elliptic curve, and it contains parameters a, b. The curve is over prime field p and has order n. A point P on the curve is chosen as base point.

The scheme ECIES is composed of three algorithms: key generation, encryption and decryption.

A. Key Generation

For key generation, we need to choose a shared secret d as a private key. Then, a public key Q, which is a point on the curve, is generated using Q = [d]P. Therefore, the key generator returns a key pair (Q, d) to the following encryption algorithm and decryption algorithm.

Assume Alice wants to send Bob a message. Bob has a public key Q. Alice and Bob both know the private key d.

B. ECIES Encryption

ECIES encryption has the following steps to encrypt a message.

- Alice generates a random number $k \in [1, p]$.
- Alice calculates U = [k]P.
- Alice calculates T = [k]Q.
- Alice uses a key derivation function (KDF) to compute two keys k1 and k2 from T. Since T is a 192 bit key, a hash function is needed to hash the key into 256 bit. Then Alice can choose the first half of the 256-bit hashed key as k1 and second half as k2.
- Alice then uses 128-bit AES encryption algorithm to encrypt her message with key k1, and obtain cipher text c.
- Alice then chooses HMAC-SHA256 to calculate a message authentication code (MAC) r with k2.
- Alice sends the pair (U, c, r) as an encrypted message e to Bob.

C. ECIES Decryption

At Bob's side, ECIES decryption is shown in the following steps.

- Bob parses encrypted message e into (U, c, r).
- Bob has the secret private key d, so he can computes T = [d]U.
- Like what Alice has done with the T, Bob uses the same key derivation function to obtain k1, and k2.
- Bob needs to make sure that the message c he receives is authentic, so he computes MAC using k2 and gets a result. He compares the result with r that he receives from Alice. If they are equal, then process to the next step; otherwise, the message is invalid and discarded.
- Bob uses the 128-bit AES decryption algorithm to decrypt c. He obtains the original message m.

If any of the checks fails: reject the message as forged.

II. Bouncy Castle & Spongy Castle

Bouncy Castle (BC) is a set of easy-to-use cryptography APIs. It is implemented in both Java and C#. It is a provider for the Java Cryptography Extension and the Java Cryptography Architecture. It also contains a lightweight cryptography API.

The Android platform unfortunately ships with a cut-down version of Bouncy Castle - as well as being crippled, it also makes installing an updated version of the libraries difficult due to classloader conflicts. [1]

Spongy Castle contains some small changes to the stock Bouncy Castle in order to make it work on Android.

III. Implementation

This ECIES implementation uses JAVA SE 6 JDK 1.6 and Spongy Castle package. The development environment is Eclipse SDK V4.2.2.

Spongy Castle artifacts are published on Maven Central. The following libraries needs to be downloaded and be set in java build path.

- sc-light-jdk15on (jar) - Core lightweight API
- scprov-jdk15on (jar) - JCE provider (requires sc-light-jdk15on)

Authors are with the Department of Computer Science, University of California, Santa Barbara, CA 93106. E-mail: liangxia@cs.ucsb.edu

• scpkix-jdk15on (jar) - PKIX, CMS, EAC, TSP, PKCS, OCSP, CMP, and CRMF APIs (requires scprov-jdk15on)

Java BigInteger library is very useful in calculating big numbers of arbitrary length. Our private key is BigInteger type, and public key, which is point Q, is composed of two BigInteger variables.

Below is my java class structure.

A. public class ECIES_Engine

We define class ECIES_Engine as our ECIES engine. We choose NIST 192-bit curve. Please refer to ECIES_Engine.java for more details.

B. public class ECIESDemoActivity

We define class ECIESDemoActivity to build an application on top of ECIES MyEngine class. It is an Android activity class as user GUI. Please refer to ECIESDemoActivity.java for details.

IV. Compilation

This section tells you how to compile, run and test our program.

If you only want to install the program, then unzip the source code and install bin/ECIES_Demo.apk on your Android device. Or download binary file from http://www.cs.ucsb.edu/~liangxia/ECIES_Demo.apk and install it.

The following steps show how to build the program in Eclipse.

• Make sure that you have JAVA SE 6 JDK 1.6 installed on your computer. The operating system is Windows 7 Professional. The developing environment is Eclipse SDK.
• Make sure Android 4.2.2 API Level 17 is installed on your computer.
• Create a new Android project in Eclipse.
• Unzip the source code ECIES_Demo.zip and import them into this Android project.
• Add Spongy Castle library to the project. Go to project properties and find java build path. Add the three Spongy Castle APIs mentioned in the above section to the project.
• Connect your Android device (for example, Samsung Nexus S) to your computer via a USB cable. Make sure the debugging mode on Android device is on.
• Build or refresh the project, and run it as an Android application.
• You can see from the Eclipse output window the log of running this program. You will be able to see the program running on the Samsung Nexus S.

V. Evaluation

This demo runs on three devices. The first device is Android virtual device simulating Nexus S on Eclipse. Since it is a virtual device, it is also the slowest device. The second device is a physical Samsung Nexus S. It runs at 1GHz and has 512 MB memory. The third device is Android Mini TV Stick. It is based on ARM Cortex A9 Processor at 1GHz and 1GB memory. The following table shows the timing information for encryption and decryption on all three devices.

Table 1: ECIES Encryption and Decryption Time on Different Devices.

Device	Enc (ms)	Dec (ms)
Android Virtual Device	4536	1670
Samsung Nexus S	566	225
Android Mini TV Stick	337	181

VI. Conclusion

My project is an implementation of ECIES algorithms on Android devices. Fortunately, with Java BigInteger library and Spongy Castle Android Java Cryptography APIs, the implementation goes very smoothly. The only thing that needs to take care is the two schemes: encryption and decryption. The 128-bit AES algorithm shows a secure symmetric encryption.

Our original implementation of AES can only deal with a message that is a multiple of 16 characters. With the Cipher Block Chaining (CBC) technique, we can encrypt arbitrary long messages, such as the message the demo shows.

For future work, we need to take care of side channel attacks. For example, when calculating point multiplication, an aggressive attacker could leverage the timing information to launch timing attacks. Encryption process could also be attacked by some malicious process which fills cache beforehand and computes cache usage on Android device to do cache timing attack. Point multiplication is also vulnerable to the simple power analysis or differential power analysis when the device is obtained by a malicious person.

References

[1] Spongy Castle, *http://rtyley.github.io/spongycastle/* March 2013.