[Letspic]Cryptography with Java 02

보고서 및 논문 윤리 서약

1. 나는 보고서 및 논문의 내용을 조작하지 않겠습니다.
2. 나는 다른 사람의 보고서 및 논문의 내용을 내 것처럼 무단으로 복사하지 않겠습니다.
3. 나는 다른 사람의 보고서 및 논문의 내용을 참고하거나 인용할 시 참고 및 인용 형식을 갖추고 출처를 반드시 밝히겠습니다.
4. 나는 보고서 및 논문을 대신하여 작성하도록 청탁하지도 청탁받지도 않겠습니다.

나는 보고서 및 논문 작성 시 위법 행위를 하지 않고, 명지인으로서 또한 공학인으로서 나의 양심과 명예를 지킬 것을 약속합니다.

보고서명 : [Letspic]Cryptography with Java 02

학　　과 : 컴퓨터공학과
담당교수 : 신민호 교수님
학　　번 : 60122480
이　　름 : 전영민　　(서명)

1. 목적

Letpics 안드로이드 어플 사용자 그룹 간 데이터 공유에서 발생하는 보안 문제와 parse.com에 저장되는 사진 메타데이터들의 관리 문제에 대해 구체적인 보안 솔루션을 적용하기 전에 android에서 사용할 수 있는 Java 기반의 cryptography를 확인한다.

2. 배경

안드로이드 기반 기기에서는 SUN JCE나 BC JCE의 보안 프로바이더가 제공하는 API를 사용한다. 따라서 Java 기반으로 해당 기능들을 테스트 해보고 소스코드 분석을 통해 제공되는 API를 이해할 필요가 있다.

3. 방법론

‘Beginning Cryptography with Java™’, David Hook 의 chapter 순대로 예제를 통해 API를 숙지한다.
https://www.bouncycastle.org/specifications.html 을 통해 알고리즘 등의 스펙을 확인한다.

4. 코드분석 정리

```
package chpater3;
import java.security.SecureRandom;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
public class Utils extends chapter2.Utils{
	public static SecretKey createKeyForAES(int bitLength, SecureRandom random) throws Exception{
		KeyGenerator generator = KeyGenerator.getInstance("AES", "BC");
		generator.init(bitLength, random);
		return generator.generateKey();
	}
	public static IvParameterSpec createCtrIvForAES(int messageNumber, SecureRandom random){
		byte[] ivBytes =new byte[16];
		// initially randomize
		random.nextBytes(ivBytes);
		// set the message number bytes
		ivBytes[0] = (byte) (messageNumber >>24);
		ivBytes[1] = (byte) (messageNumber >>16);
		ivBytes[2] = (byte) (messageNumber >>8);
		ivBytes[3] = (byte) (messageNumber >>0);
		// set the counter bytes to 1
		for(int i =0; i !=7; i++ ){
			ivBytes[8+i] =0;
		}
		ivBytes[15] =1;
		return new IvParameterSpec(ivBytes);
	}

	public static String toString(byte[] bytes, intlength){
		char[] chars = new char[length];
		for(int i =0; i != chars.length; i++ ){
			chars[i] = (char) (bytes[i] & 0xff);
		}
		return new String(chars);
	}

	public static String toString(byte[] bytes){
		return toString(bytes, bytes.length);
	}

	public static byte[] toByteArray(String string){
		byte[] bytes = new byte[string.length()];
		char[] chars = string.toCharArray();
		for(int i =0 ; i != chars.length; i++ ){
			bytes[i] = (byte)chars[i];
		}
		return bytes;
	}
}
```

◆ 주의할 점

Java에서 기본으로 제공되는 String 클래스에 getBytes() 메소드를 호출하거나 생성자를 통해 byte array를 얻을 수 있다. 하지만 기본 String 클래스는 JVM이 사용하는 default charset에 영향을 받으면서 string-to-byte, byte-to-string 변환을 하게 된다. 모든 JVM이 예를 들어 String.getBytes(“ UTF8 ”)와 같이 “ UTF-8 ” 등 특정한 charset을 지원하고 있지 않다. 따라서 toByteArray(String string), toString(byte[] bytes, intlength)의 구현이 필요하다.

```java
package chpater3;
import java.security.Key;
import java.security.SecureRandom;
import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
public class TamperedExample {
        public static void main(String[] args) throws Exception {
                SecureRandom random = new SecureRandom();
                IvParameterSpec ivSpec = Utils.createCtrIvForAES(1, random);
                Key key = Utils.createKeyForAES(256, random);
                Cipher cipher = Cipher.getInstance("AES/CTR/NoPadding", "BC");
                String input ="Transfer 0000100 to AC 1234-5678";
                System.out.println("input : "+ input);

                // encryption pass
                cipher.init(Cipher.ENCRYPT_MODE, key, ivSpec);
                byte[] cipherText = cipher.doFinal(Utils.toByteArray(input));

                // tampering step
                cipherText[9] ^='0' ^ '9';

                // decryptiobn step
                cipher.init(Cipher.DECRYPT_MODE, key, ivSpec);
                byte[] plainText = cipher.doFinal(cipherText);
                System.out.println("plain : "+Utils.toString(plainText));
        }
}
```

◆ 출력결과

```
input : Transfer 0000100 to AC 1234-5678
plain : Transfer 9000100 to AC 1234-5678
```

◆ 확인하고자 하는 바

AES, CTR, NoPadding 으로 전달되는 암호화문이 변경되는 것을 확인

```java
package chpater3;
import java.security.Key;
import java.security.MessageDigest;
import java.security.SecureRandom;
import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
public class TamperedWithDigestExample {
        public static void main(String[] args) throws Exception{
                SecureRandom random = new SecureRandom();
                IvParameterSpec ivSpec = Utils.createCtrIvForAES(1, random);
                Key key = Utils.createKeyForAES(256, random);
                Cipher cipher = Cipher.getInstance("AES/CTR/NoPadding", "BC");
                String input ="Transfer 0000100 to AC 1234-5678";
                MessageDigest hash = MessageDigest.getInstance("SHA-1", "BC");
                System.out.println("input : "+input);

                // encryption step
                cipher.init(Cipher.ENCRYPT_MODE, key, ivSpec);
                byte[] cipherText = new byte[cipher.getOutputSize(input.length()
                        + hash.getDigestLength())];
                int ctLength = cipher.update(Utils.toByteArray(input),0, input.length(), cipherText, 0);
                hash.update(Utils.toByteArray(input));
                ctLength += cipher.doFinal(hash.digest(),0,hash.getDigestLength(),cipherText, ctLength);

                // tampering step
                cipherText[9] ^='0' ^ '9';

                // decryption step
                cipher.init(Cipher.DECRYPT_MODE, key, ivSpec);
                byte[] plainText = cipher.doFinal(cipherText, 0, ctLength);
                int messageLength = plainText.length- hash.getDigestLength();
                hash.update(plainText, 0, messageLength);

                byte[] messageHash =newbyte[hash.getDigestLength()];
                System.arraycopy(plainText, messageLength, messageHash, 0, messageHash.length);
                System.out.println("plain : "+Utils.toString(plainText, messageLength)+
                                " verified : "+MessageDigest.isEqual(hash.digest(), messageHash));
        }
}
```

◆ 출력결과

```
input : Transfer 0000100 to AC 1234-5678
plain : Transfer 9000100 to AC 1234-5678 verified : false
```

◆ 확인하고자 하는 바

기존의 단계에서 MessageDigest(SHA-1)를 추가하여 암호화문을 전송하고,
암호문의 일부분을 변경하였을 때,
복호화 과정에서 나온 평문을 hash한 결과와 전송된 hash 값을 비교하여 검증할 수 있다.

◆ MessageDigest

i) getInstance(String algorithm, String provider)
ii) update(byte[] input)
iii) digest()
iv) getDigestLength()
v) isEqual(byte[] digesta, byte[] digestb)

```java
package chpater3;
import java.security.Key;
import java.security.MessageDigest;
import java.security.SecureRandom;
import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
public class TamperedDigestExample {
        public static void main(String[] args) throws Exception {
                SecureRandom random = new SecureRandom();
                IvParameterSpec ivSpec = Utils.createCtrIvForAES(256, random);
                Key key = Utils.createKeyForAES(256, random);
                Cipher cipher = Cipher.getInstance("AES/CTR/NoPadding", "BC");
                String input ="Transfer 0000100 to AC 1234-5678";
                MessageDigest hash = MessageDigest.getInstance("SHA-1", "BC");
                System.out.println("input : "+ input);

                // encryption pass
                cipher.init(Cipher.ENCRYPT_MODE, key, ivSpec);
                byte[] cipherText =newbyte[cipher.getOutputSize(input.length()
                        + hash.getDigestLength())];
                int ctLength = cipher.update(Utils.toByteArray(input),0,input.length(), cipherText, 0);
                hash.update(Utils.toByteArray(input));
                ctLength += cipher.doFinal(hash.digest(),0,hash.getDigestLength(),cipherText, ctLength);

                // tampering step
                cipherText[9] ^='0' ^ '9';

                // replace digest
                byte[] originalHash = hash.digest(Utils.toByteArray(input));
                byte[] tamperedHash = hash.digest(Utils.toByteArray(
                        "Transfer 9000100 to AC 1234-5678"));
                for(int i = ctLength - hash.getDigestLength(), j =0; i != ctLength; i++, j++ ){
                        cipherText[i] ^= originalHash[j] ^ tamperedHash[j];
                }

                // decryption step
                cipher.init(Cipher.DECRYPT_MODE, key, ivSpec);
                byte[] plainText = cipher.doFinal(cipherText, 0, ctLength);
                int messageLength = plainText.length- hash.getDigestLength();
                hash.update(plainText, 0, messageLength);
                byte[] messageHash =newbyte[hash.getDigestLength()];
                System.arraycopy(plainText, messageLength, messageHash, 0, messageHash.length);

                System.out.println("plain : "+Utils.toString(plainText, messageLength)
                                +" verified : "+ MessageDigest.isEqual(hash.digest(), messageHash));
        }
}
```

◆ 출력결과

```
input : Transfer 0000100 to AC 1234-5678
plain : Transfer 9000100 to AC 1234-5678 verified : true
```

◆ 확인하고자 하는 바

이전 단계에서 MessageDigest(SHA-1)을 변경된 것을 검증해낼 수 있었지만,
Hash 전부를 변경해버리면 복호화 과정에서의 검증이 무의미 해진다.

```java
package chpater3;
import java.security.Key;
import java.security.MessageDigest;
import java.security.SecureRandom;
import javax.crypto.Cipher;
import javax.crypto.Mac;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
public class TamperedWithHMacExample {
        public static void main(String[] args) throws Exception {
                SecureRandom random = new SecureRandom();
                IvParameterSpec ivSpec = Utils.createCtrIvForAES(1, random);
                Key key = Utils.createKeyForAES(256, random);
                Cipher cipher = Cipher.getInstance("AES/CTR/NoPadding", "BC");
                String input ="Transfer 0000100 to AC 1234-5678";
                Mac hMac = Mac.getInstance("HmacSHA1", "BC");
                Key hMacKey =new SecretKeySpec(key.getEncoded(), "HmacSHA1");
                System.out.println("input : "+ input);

                // encryption step
                cipher.init(Cipher.ENCRYPT_MODE, key, ivSpec);
                byte[] cipherText =newbyte[cipher.getOutputSize(input.length()+hMac.getMacLength())];
                int ctLength = cipher.update(Utils.toByteArray(input),0, input.length(), cipherText, 0);
                hMac.init(hMacKey);
                hMac.update(Utils.toByteArray(input));
                ctLength += cipher.doFinal(hMac.doFinal(), 0,hMac.getMacLength(), cipherText, ctLength);

                // tampering step
                cipherText[9] ^='0' ^ '9';

                // replace digest
                // ?

                // decryption pass
                cipher.init(Cipher.DECRYPT_MODE, key, ivSpec);
                byte[] plainText = cipher.doFinal(cipherText, 0, ctLength);
                int messageLength = plainText.length- hMac.getMacLength();
                hMac.init(hMacKey);
                hMac.update(plainText,0,messageLength);
                byte[] messageHash =newbyte[hMac.getMacLength()];
                System.arraycopy(plainText, messageLength, messageHash, 0, messageHash.length);
                System.out.println("plain : "+Utils.toString(plainText, messageLength)
                                +" verfied : "+ MessageDigest.isEqual(hMac.doFinal(), messageHash));

        }
}
```

◆ 출력결과

```
input : Transfer 0000100 to AC 1234-5678
plain : Transfer 9000100 to AC 1234-5678 verfied : false
```

◆ 확인하고자 하는 바

이전 단계의 MessageDigest로 메시지 검증을 하는 것이 아닌 hMac(HmacSHA1)을 통해 검증을 하려 한다.
결과적으로 인증과 메시지 검증을 할 수 있다.

◆ MAC

i) getInstance(String algorithm, String provider)
ii) init(Key key)
iii) update(byte[] input)
iv) doFinal()
v) getMacLength()

```java
package chpater3;
import java.security.Key;
import java.security.MessageDigest;
import java.security.SecureRandom;
import javax.crypto.Cipher;
import javax.crypto.Mac;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
public class CipherMacExample {
        public static void main(String[] args) throws Exception {
                SecureRandom random = new SecureRandom();
                IvParameterSpec ivSpec = Utils.createCtrIvForAES(1, random);
                Key key = Utils.createKeyForAES(256, random);
                Cipher cipher = Cipher.getInstance("AES/CTR/NoPadding", "BC");
                String input ="Transfer 0000100 to AC 1234-5678";
                Mac mac = Mac.getInstance("DES", "BC");
                byte[] macKeyBytes = new byte[] {
                                0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08 };
                Key macKey =new SecretKeySpec(macKeyBytes, "DES");
                System.out.println("input : "+input);

                // encryption step
                cipher.init(Cipher.ENCRYPT_MODE, key, ivSpec);
                byte[] cipherText = new byte[cipher.getOutputSize(input.length() + mac.getMacLength())];
                int ctLength = cipher.update(Utils.toByteArray(input),0, input.length(), cipherText, 0);
                mac.init(macKey);
                mac.update(Utils.toByteArray(input));
                ctLength += cipher.doFinal(mac.doFinal(), 0, mac.getMacLength(), cipherText, ctLength);

                // decryption step
                cipher.init(Cipher.DECRYPT_MODE, key, ivSpec);
                byte[] plainText = cipher.doFinal(cipherText, 0, ctLength);
                int messageLength = plainText.length- mac.getMacLength();
                mac.init(macKey);
                mac.update(plainText, 0, messageLength);
                byte[] messageHash =new byte[mac.getMacLength()];
                System.arraycopy(plainText, messageLength, messageHash, 0, messageHash.length);

                System.out.println("plain : "+Utils.toString(plainText, messageLength)
                                +" verified : "+MessageDigest.isEqual(mac.doFinal(), messageHash));
        }
}
```

◆ 출력결과

```
input : Transfer 0000100 to AC 1234-5678
plain : Transfer 0000100 to AC 1234-5678 verified : true
```

◆ 확인하고자 하는 바

이전단계와 같이 메시지 검증과 인증을 동시에 하고자 하는데,
hash MAC이 아닌 고전적인 MAC을 암호화하여 이를 하고자 한다.

```java
package chpater3;
import java.security.MessageDigest;
public class PKCS5Scheme1 {
	private MessageDigest digest;
	public PKCS5Scheme1(MessageDigest digest){
		this.digest = digest;
	}
	public byte[] generateDeriveKey(char[] password, byte[] salt, int iterationCount){
		for(int i =0 ; i  != password.length; i++ ){
			digest.update((byte)password[i]);
		}
		digest.update(salt);
		byte[] digestBytes = digest.digest();
		for(int i =1; i < iterationCount; i++ ){
			digest.update(digestBytes);
			digestBytes = digest.digest();
		}
		return digestBytes;
	}
}
```

```java
package chpater3;
import java.security.MessageDigest;
import javax.crypto.Cipher;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.SecretKeySpec;
public class PKCS5Scheme1Test {
	public static void main(String[] args) throws Exception {
		char[] password ="hello".toCharArray();
		byte[] salt = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };
		byte[] input = new byte[] { 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f };
		int iterationCount =100;

		System.out.println("input : "+Utils.toHex(input));

		// encryption step using regular PBE
		Cipher cipher = Cipher.getInstance("PBEWithSHA1AndDES", "BC");
		SecretKeyFactory fact = SecretKeyFactory.getInstance("PBEWithSHA1AndDES", "BC");
		PBEKeySpec pbeKeySpec =new PBEKeySpec(password, salt, iterationCount);
		cipher.init(Cipher.ENCRYPT_MODE, fact.generateSecret(pbeKeySpec));
		byte[] enc = cipher.doFinal(input);
		System.out.println("encrypt: "+ Utils.toHex(enc));

		// decryption step - using the local implementation
		cipher = Cipher.getInstance("DES/CBC/PKCS5Padding");
		PKCS5Scheme1 pkcs5s1 =new PKCS5Scheme1( MessageDigest.getInstance("SHA-1", "BC"));
		byte[] derivedKey = pkcs5s1.generateDeriveKey(password, salt, iterationCount);
		cipher.init(Cipher.DECRYPT_MODE, new SecretKeySpec(derivedKey, 0, 8, "DES"),
			new IvParameterSpec(derivedKey, 8, 8));
		byte[] dec = cipher.doFinal(enc);
		System.out.println("decrypt : "+Utils.toHex(dec));
	}
}
```

◆ 출력결과

```
input : 0a0b0c0d0e0f
encrypt: af1c264c0946104d
decrypt : 0a0b0c0d0e0f
```

◆ 확인하고자 하는 바

패스워드 기반의 (SHA1, DES)암복호화를 진행한다.

```java
package chpater3;
import java.security.MessageDigest;
public class MGF1 {
	private MessageDigest digest;
	public MGF1(MessageDigest digest){
		this.digest = digest;
	}
	private void ItoOSP(int i, byte[] sp){
		sp[0] = (byte)(i >>>24);
		sp[1] = (byte)(i >>>16);
		sp[2] = (byte)(i >>>8);
		sp[3] = (byte)(i >>>0);
	}
	public byte[] generateMask(byte[] seed, intlength){
		byte[] mask =new byte[length];
		byte[] C =new byte[4];
		int counter =0;
		int hLen = digest.getDigestLength();
		while( counter < (length/ hLen) ){
			ItoOSP(counter, C);
			digest.update(seed);
			digest.update(C);
			System.arraycopy(digest.digest(),0,mask,counter * hLen, hLen);
			counter++;
		}
		if( (counter * hLen) < length ){
			ItoOSP(counter, C);
			digest.update(seed);
			digest.update(C);
			System.arraycopy(digest.digest(), 0, mask, counter * hLen,
				mask.length- (counter * hLen));
		}
		return mask;
	}
	public static void main(String[] args) throws Exception {
		MGF1 mgf1 =new MGF1(MessageDigest.getInstance("SHA-1", "BC"));
		byte[] source =new byte[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
		System.out.println(Utils.toHex(mgf1.generateMask(source, 20)));
	}
}
```

◆ 출력결과

```
e0a63d7c8c4d81cd1c0567bfab6c22ed2022977f
```

◆ 확인하고자 하는 바

PKCS #1 V2.1 에 존재하는 Mask Generation Function의 예제이다.

```java
package chpater3;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.security.DigestInputStream;
import java.security.DigestOutputStream;
import java.security.MessageDigest;
public class DigestIOExample {
        public static void main(String[] args) throws Exception {
                byte[] input = new byte[] {
                                0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                                0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
                                0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06 };
                MessageDigest hash = MessageDigest.getInstance("SHA1");
                System.out.println("input : "+Utils.toHex(input));

                // input pass
                ByteArrayInputStream bIn = new ByteArrayInputStream(input);
                DigestInputStream dIn = new DigestInputStream(bIn, hash);
                ByteArrayOutputStream bOut = new ByteArrayOutputStream();
                int ch;
                while( (ch = dIn.read()) >= 0 ){
                        bOut.write(ch);
                }
                byte[] newInput = bOut.toByteArray();
                System.out.println("in digest : "+Utils.toHex(dIn.getMessageDigest().digest()));

                // output pass
                bOut = new ByteArrayOutputStream();
                DigestOutputStream dOut =new DigestOutputStream(bOut, hash);
                dOut.write(newInput);
                dOut.close();
                System.out.println("out digest: "+Utils.toHex(dOut.getMessageDigest().digest()));
        }
}
```

◆ 출력결과

```
input : 000102030405060708090a0b0c0d0e0f00010203040506
in digest : 864ea9ee65d6f86847ba302ded2da77ad6c64722
out digest: 864ea9ee65d6f86847ba302ded2da77ad6c64722
```

◆ 확인하고자 하는 바

I/O Stream을 통한 암복호화 및 메시지 검증