[Letspic]Cryptography with Java 01

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 01

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.

4. 코드분석 정리

```
package chapter1;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
public class SimplePolicyTest {
        public static void main(String[] args) throws Exception {
                byte[] data = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07 };
                // create a 64 bit secret key from raw bytes
                SecretKey key64 = new SecretKeySpec( new byte[] {
                0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x05, 0x06, 0x07}, "Blowfish");
                // create a cipher and attempt to encrypt the data block with our key
                Cipher c = Cipher.getInstance("Blowfish/ECB/NoPadding");

                c.init(Cipher.ENCRYPT_MODE, key64);
                c.doFinal(data);
                System.out.println("64 bit test: passed");

                // create a 192 bit secret key from raw bytes
                SecretKey key192 = new SecretKeySpec( new byte[] {
                0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
                0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17 }, "Blowfish" );

                // now try encrypting with the larger key
                c.init(Cipher.ENCRYPT_MODE, key192);
                c.doFinal(data);
                System.out.println("192 bit test: passed");
                System.out.println("Tests completed");
        }
}
```

◆ Blowfish ? 1993년 브루스 슈나이어가 설계한 키(key) 방식의 대칭형 블록 암호, 특허가 없도록 고안된 디자인

◆ 확인하고자 하는 바

=> 64bit key를 가지고 암호화 하는 것은 문제가 없었다. 다만 192bit key를 가지고 암호화를 시도할 때 다음과 같은 예외가 발생하였다. ' java.security.InvalidKeyException: Illegal key size or default parameters '

◆ 해결 과정

ⅰ) %JAVA_HOME%/jre/lib/security 하에 policy 파일이 이미 있기에 코드에 문제가 있는지 확인하기 위해 Decomplier를 통해 source attachment를 하여 확인해보았으나 문제는 없었다.

ⅱ) UnlimitedJCEPolicyJDK6~8중 버전에 맞는 policy 파일로 교체하여 문제를 해결해야 한다.

ⅲ) android 에도 이러한 key size가 문제가 되는지 알아봐야 한다.

```java
package chapter1;
import java.security.Security;
public class SimpleProviderTest {
	public static void main(String[] args){
		String providerName ="BC";
		if( Security.getProvider(providerName) == null){
			System.out.println(providerName +" provider not installed");
		} else {
			System.out.println(providerName +" is installed.");
		}
	}
}
```

◆ 확인하고자 하는바

=> JCA는 프로바이더 기반으로 하여서 구현 독립성(Implementation Independence), 구현 호환성(Implementation interoperability), 알고리즘 확장성(Algorithm Extensibility)를 갖추고 있다.

이 프로바이더들은 JRE가 설치될 때 기본적으로 설치되며 (Sun) 추가로 설치할 수 있다.(Bouncy Castle)

해당 프로바이더들은 java.security에 다음과 같이 정의 된다.

```
#
# List of providers and their preference oreders (see above);
#
security.provider.1=sun.security.provider.Sun
security.provider.2=com.sun.net.ssl.internal.ssl.Provider
security.provider.3=com.sun.rsajca.Provider
security.provider.4=com.sun.crypto.provider.SunJCE
security.provider.5=org.bouncycastle.jce.provider.BouncyCastleProvider
```

◆ 해결 과정

ⅰ) java.security 파일에 명시 하지 않았다면 Bouncy Castle 프로바이더를 인식하지 못한다.

ⅱ) Bouncy Castle 설치 했다면, java.security에도 적용하자.

ⅲ) android 에서는 java.security 파일을 수정해야 하는가? 확인해야 한다.

```java
package chapter1;
import javax.crypto.Cipher;
public class PrecedenceTest {
	public static void main(String[] args) throws Exception {
		Cipher cipher = Cipher.getInstance("Blowfish/ECB/NoPadding");
		System.out.println(cipher.getProvider());
		cipher = Cipher.getInstance("Blowfish/ECB/NoPadding", "BC");
		System.out.println(cipher.getProvider());
	}
}
```

◆ 확인하고자 하는바

=> SunJCE 프로바이더와 BouncyCastle 프로바이더의 버전을 확인한다.

```java
package chapter2;
public class Utils {
	private static String digits ="0123456789abcdef";
	public static String toHex(byte[] data, intlength) {
		StringBuffer buf =new StringBuffer();
		for (int i =0; i !=length; i++) {
			int v = data[i] & 0xff;
			buf.append(digits.charAt(v >> 4));
			buf.append(digits.charAt(v & 0xf));
		}
		return buf.toString();
	}
	public staticString toHex(byte[] data) {
		return toHex(data, data.length);
	}
}
```

◆ 확인하고자 하는바

=> byte를 16진수로 변환하는 클래스

```java
package chapter2;
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
public class SimpleSymmetricExample {
        public static void main(String[] args) throws Exception {
                byte[] input =new byte[] { 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
                                           (byte)0x88, (byte)0x99, (byte)0xaa, (byte)0xbb,
                                           (byte)0xcc, (byte)0xdd, (byte)0xee, (byte)0xff};

                byte[] keyBytes =new byte[] {
                                0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                                0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
                                0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17 };

                SecretKeySpec key = new SecretKeySpec(keyBytes, "AES");
                Cipher cipher = Cipher.getInstance("AES/ECB/NoPadding", "BC");
                System.out.println("input text : "+ Utils.toHex(input));

                // encryption pass
                byte[] cipherText = new byte[input.length];
                cipher.init(Cipher.ENCRYPT_MODE, key);
                int ctLength = cipher.update(input,  0, input.length, cipherText, 0);
                ctLength += cipher.doFinal(cipherText, ctLength);
                System.out.println("cipher text: "+ Utils.toHex(cipherText) +" bytes: "+ctLength);

                // decryption pass
                byte[] plainText = new byte[ctLength];
                cipher.init(Cipher.DECRYPT_MODE, key);
                int ptLength = cipher.update(cipherText, 0, ctLength, plainText, 0);
                ptLength += cipher.doFinal(plainText, ptLength);
                System.out.println("plain text : "+ Utils.toHex(plainText) +" bytes: "+ ptLength );
        }
}
```

◆ 출력결과

```
input text : 00112233445566778899aabbccddeeff
cipher text: dda97ca4864cdfe06eaf70a0ec0d7191 bytes: 16
plain text : 00112233445566778899aabbccddeeff bytes: 16
```

◆ javax.crypto.Cipher : 암복호화 엔진

i) getInstance(String transformation, String provider)

=> “ 알고리즘이름/모드/패딩 ” , “ 프로바이더이름 ” 을 매개변수로 Cipher 객체화

ii) init(int opmode, Key key)

=> Cipher.ENCRYPT_MODE or Cipher.DECRYPT_MODE, “ key ” 을 매개변수로 초기화 진행

iii) update(byte[] input, int inputOffset, int inputLen, byte[] output, int outputOffset)

=> “ 입력바이트배열 ” , “ 입력 offset ” , “ 입력바이트배열의 길이 ” , “ 출력바이트배열 ” , “ 출력offset ” 으로 암복호화를 진행. (주의) 바이트배열의 인코딩을 명시하는 것이 바람직함

iv) doFinal(byte[] output, int outputOffset)

=> “ 출력바이트배열 ” , “ 출력 offset ” 으로 암복호화의 끝을 알림

◆ javax.crypto.SecretKeySpec : 지정된 바이트배열로부터 지정된 알고리즘의 key를 구축함

i) SecretKeySpec(byte[] key, String algorithm)

=> “ key바이트배열 ” , “ 알고리즘 ” 으로 key를 구축함. 다만 해당 바이트배열이 유효한지 검사하지는 않음

```java
package chapter2;
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
public class SimpleSymmetricPaddingExample {
	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,
				0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17 };
		byte[] keyBytes =new byte[] {
				0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
				0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
				0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17 };

		SecretKeySpec key = new SecretKeySpec(keyBytes, "AES");
		Cipher cipher = Cipher.getInstance("AES/ECB/PKCS7Padding", "BC");
		System.out.println("input : "+ Utils.toHex(input));

		// encryption pass
		cipher.init(Cipher.ENCRYPT_MODE, key);
		byte[] cipherText = new byte[cipher.getOutputSize(input.length)];
		int ctLength = cipher.update(input, 0, input.length, cipherText, 0);
		ctLength += cipher.doFinal(cipherText, ctLength);
		System.out.println("cipher: "+Utils.toHex(cipherText)+" bytes: "+ctLength);

		// decryption pass
		cipher.init(Cipher.DECRYPT_MODE, key);
		byte[] plainText = new byte[cipher.getOutputSize(ctLength)];
		int ptLength = cipher.update(cipherText, 0, ctLength, plainText, 0);
		ptLength += cipher.doFinal(plainText, ptLength);
		System.out.println("plain : "+Utils.toHex(plainText)+" bytes: "+ptLength);
	}
}
```

◆ 출력결과

```
input : 000102030405060708090a0b0c0d0e0f1011121314151617
cipher: 0060bffe46834bb8da5cf9a61ff220aefa46bbd3578579c0fd331874c7234233 bytes: 32
plain : 000102030405060708090a0b0c0d0e0f10111213141516170000000000000000 bytes: 24
```

◆ 확인하고자 하는바

=> block cipher 에서 input의 길이가 항상 block size로 나누어지지 않으므로 0 bit나 다른 bit로 채우는 것을 padding 이라함, plain text의 뒷부분이 0bit로 채워짐

```java
package chapter2;
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
public class SimpleECBExample {
        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, 0x07 };
                byte[] keyBytes = new byte[] {
                                0x01, 0x23, 0x45, 0x67,
                                (byte) 0x89, (byte) 0xab, (byte) 0xcd, (byte) 0xef };

                SecretKeySpec key = new SecretKeySpec(keyBytes, "DES");
                Cipher cipher = Cipher.getInstance("DES/ECB/PKCS7Padding", "BC");
                System.out.println("input : "+ Utils.toHex(input));

                // encryption pass
                cipher.init(Cipher.ENCRYPT_MODE, key);
                byte[] cipherText = new byte[cipher.getOutputSize(input.length)];
                int ctLength = cipher.update(input, 0, input.length, cipherText, 0);
                ctLength += cipher.doFinal(cipherText, ctLength);
                System.out.println("cipher : "+ Utils.toHex(cipherText, ctLength)+" bytes: "+ctLength);

                // decryption pass
                cipher.init(Cipher.DECRYPT_MODE, key);
                byte[] plainText = new byte[cipher.getOutputSize(ctLength)];
                int ptLength = cipher.update(cipherText, 0, ctLength, plainText, 0);
                ptLength += cipher.doFinal(plainText,  ptLength);
                System.out.println("plain : "+Utils.toHex(plainText, ptLength)+" bytes: "+ptLength);
        }
}
```

◆ 출력결과

```
input : 0001020304050607080090a0b0c0d0e0f0001020304050607
cipher : 3260266c2cf202e28325790654a444d93260266c2cf202e2086f9a1d74c94d4e bytes: 32
plain : 0001020304050607080090a0b0c0d0e0f0001020304050607 bytes: 24
```

◆ 확인하고자 하는 바

=> ECB 모드의 테스트

```java
package chapter2;
import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
public class SimpleCBCExample {
        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, 0x07 };
                byte[] keyBytes =new byte[] {
                                0x01, 0x23, 0x45, 0x67,
                                (byte) 0x89, (byte) 0xab, (byte) 0xcd, (byte) 0xef };
                byte[] ivBytes =new byte[] {
                                0x07, 0x06, 0x05, 0x04, 0x03, 0x02, 0x01, 0x00 };

                SecretKeySpec key = new SecretKeySpec(keyBytes, "DES");
                IvParameterSpec ivSpec = new IvParameterSpec(ivBytes);
                Cipher cipher = Cipher.getInstance("DES/CBC/PKCS7Padding", "BC");
                System.out.println("input : "+Utils.toHex(input));

                // encryption pass
                cipher.init(Cipher.ENCRYPT_MODE, key, ivSpec);
                byte[] cipherText = new byte[cipher.getOutputSize(input.length)];
                int ctLength = cipher.update(input, 0, input.length, cipherText, 0);
                ctLength += cipher.doFinal(cipherText, ctLength);
                System.out.println("cipher : "+Utils.toHex(cipherText, ctLength)+" bytes: "+ ctLength);

                // decryption pass
                cipher.init(Cipher.DECRYPT_MODE, key, ivSpec);
                byte[] plainText = new byte[cipher.getOutputSize(ctLength)];
                int ptLength = cipher.update(cipherText,  0, ctLength, plainText, 0);
                ptLength += cipher.doFinal(plainText, ptLength);
                System.out.println("plain : "+Utils.toHex(plainText, ptLength)+" bytes: "+ptLength);
        }
}
```

◆ 출력결과

```
input : 000102030405060708090a0b0c0d0e0f0001020304050607
cipher : 8a87d41c5d3caead0c21f1b3f12a6cd75424fa086e029e404c89d4c1b9457818 bytes: 32
plain : 000102030405060708090a0b0c0d0e0f0001020304050607 bytes: 24
```

◆ 확인하고자 하는 바

=> CBC 모드의 테스트

◆ IvParameterSpec : 초기화백터를 지정함

ⅰ) IvParameterSpec(byte[] iv)

=> iv 내의 바이트를 초기화 백터로서 사용함

◆ javax.crypto.Cipher : 암복호화 엔진

ⅰ) init(int opmode, Key key, AlgorithmParameterSpec params)

=> Cipher.ENCRYPT_MODE or Cipher.DECRYPTMODE, “key”, “Iv” 을 매개변수로 초기화 진행

```java
package chapter2;
import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
public class InlineIvCBCExample {
        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, 0x07 };
                byte[] keyBytes = new byte[] {
                                0x01, 0x23, 0x45, 0x67,
                                (byte) 0x89, (byte) 0xab, (byte) 0xcd, (byte) 0xef };
                byte[] ivBytes = new byte[] {
                                0x07, 0x06, 0x05, 0x04, 0x03, 0x02, 0x01, 0x00 };
                SecretKeySpec key = new SecretKeySpec(keyBytes, "DES");
                IvParameterSpec ivSpec = new IvParameterSpec(new byte[8]);
                Cipher cipher = Cipher.getInstance("DES/CBC/PKCS7Padding", "BC");
                System.out.println("input : "+ Utils.toHex(input));

                // encryption pass
                cipher.init(Cipher.ENCRYPT_MODE, key, ivSpec);
                byte[] cipherText = new byte[cipher.getOutputSize(ivBytes.length+input.length)];
                int ctLength = cipher.update(ivBytes, 0, ivBytes.length, cipherText, 0);
                ctLength += cipher.update(input, 0, input.length, cipherText, ctLength);
                ctLength += cipher.doFinal(cipherText, ctLength);
                System.out.println("cipher : "+Utils.toHex(cipherText, ctLength)+" bytes: "+ctLength);

                // decryption pass
                cipher.init(Cipher.DECRYPT_MODE, key, ivSpec);
                byte[] buf = new byte[cipher.getOutputSize(ctLength)];
                int bufLength = cipher.update(cipherText, 0, ctLength, buf, 0);
                bufLength += cipher.doFinal(buf, bufLength);

                // remove the iv from the start of the message
                byte[] plainText = new byte[bufLength - ivBytes.length];
                System.arraycopy(buf, ivBytes.length, plainText, 0, plainText.length);
                System.out.println("plain : "+ Utils.toHex(plainText, plainText.length)
                        +" bytes: "+plainText.length);
        }
}
```

◆ 출력결과

```
input : 000102030405060708090a0b0c0d0e0f0001020304050607
cipher : 159fc9af021f30024211a5d7bf88fd0b9e2a82facabb493f39c5a9febe6a659e85039332be56f6a4 bytes: 40
plain : 000102030405060708090a0b0c0d0e0f0001020304050607 bytes: 24
```

◆ 확인하고자 하는 바

=> 초기화 벡터를 포함한 암호화, 초기화 벡터를 제외한 복호화 CBC 모드 테스트

```java
package chapter2;
import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
public class NonceIvCBCExample {
        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, 0x07 };
                byte[] keyBytes = new byte[] {
                                0x01, 0x23, 0x45, 0x67,
                                (byte) 0x89, (byte) 0xab, (byte) 0xcd, (byte) 0xef };
                byte[] msgNumber = new byte[] {
                                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
                IvParameterSpec zeroIv = new IvParameterSpec(newbyte[8]);
                SecretKeySpec key = new SecretKeySpec(keyBytes, "DES");
                Cipher cipher = Cipher.getInstance("DES/CBC/PKCS7Padding", "BC");
                System.out.println("input : "+Utils.toHex(input));

                // encryption pass
                // generate IV
                cipher.init(Cipher.ENCRYPT_MODE, key, zeroIv);
                IvParameterSpec encryptionIv = new IvParameterSpec( cipher.doFinal(msgNumber), 0, 8);

                // encrypt message
                cipher.init(Cipher.ENCRYPT_MODE, key, encryptionIv);
                byte[] cipherText = new byte[cipher.getOutputSize(input.length)];
                int ctLength = cipher.update(input, 0, input.length, cipherText, 0);
                ctLength += cipher.doFinal(cipherText, ctLength);
                System.out.println("cipher : "+Utils.toHex(cipherText, ctLength)+" bytes : "+ctLength);

                // decryption pass
                // generate IV
                cipher.init(Cipher.ENCRYPT_MODE, key, zeroIv);
                IvParameterSpec decryptionIv =new IvParameterSpec(cipher.doFinal(msgNumber), 0, 8);

                // decrypt message
                cipher.init(Cipher.DECRYPT_MODE, key, decryptionIv);
                byte[] plainText = new byte[cipher.getOutputSize(ctLength)];
                int ptLength = cipher.update(cipherText, 0, ctLength, plainText, 0);
                ptLength += cipher.doFinal(plainText, ptLength);
                System.out.println("plain : "+Utils.toHex(plainText, ptLength)+" bytes : "+ ptLength );
        }
}
```

◆ 출력결과

```
input : 000102030405060708090a0b0c0d0e0f0001020304050607
cipher : eb913126049ccdea00f2d86fda94a02fd72e0914fd361400d909f45f73058fc3 bytes : 32
plain : 000102030405060708090a0b0c0d0e0f0001020304050607 bytes : 24
```

◆ 확인하고자 하는 바

=> Iv를 Nonce 로 만들기 위해 암호화 과정을 거쳐 사용, 현 코드는 Iv 생성과 메시지 암복호화 key가 같음

```java
package chapter2;
import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
public class SimpleCTRExample {
        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 };
                byte[] keyBytes = new byte[] {
                                0x01, 0x23, 0x45, 0x67,
                                (byte) 0x89, (byte) 0xab, (byte) 0xcd, (byte) 0xef };
                byte[] ivBytes = new byte[] {
                                0x00, 0x01, 0x02, 0x03, 0x00, 0x00, 0x00, 0x01 };

                SecretKeySpec key = new SecretKeySpec(keyBytes, "DES");
                IvParameterSpec ivSpec = new IvParameterSpec(ivBytes);
                Cipher cipher = Cipher.getInstance("DES/CTR/NoPadding", "BC");
                System.out.println("input : "+Utils.toHex(input));

                // encryption pass
                cipher.init(Cipher.ENCRYPT_MODE, key, ivSpec);
                byte[] cipherText =newbyte[cipher.getOutputSize(input.length)];
                int ctLength = cipher.update(input, 0, input.length, cipherText, 0);
                ctLength += cipher.doFinal(cipherText, ctLength);
                System.out.println("cipher : "+Utils.toHex(cipherText, ctLength)+" bytes: "+ctLength);

                // decryption pass
                cipher.init(Cipher.DECRYPT_MODE, key, ivSpec);
                byte[] plainText =newbyte[cipher.getOutputSize(ctLength)];
                int ptLength = cipher.update(cipherText, 0, ctLength, plainText, 0);
                ptLength += cipher.doFinal(plainText, ptLength);
                System.out.println("plain : "+Utils.toHex(plainText, ptLength)+" bytes: "+ptLength);
        }
}
```

◆ 출력결과

```
input : 000102030405060708090a0b0c0d0e0f00010203040506
cipher : 61a1f886ff9bc709dd37cd9ce33adc6ff9ab110e46f387 bytes: 23
plain : 000102030405060708090a0b0c0d0e0f00010203040506 bytes: 23
```

◆ 확인하고자 하는 바

=> CTR 모드의 테스트

```java
package chapter2;
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
public class SimpleStreamExample {
        public static void main(String[] args) throws Exception{
                byte[] input = new byte[] {
                                0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
                                (byte) 0x88, (byte) 0x99, (byte) 0xaa, (byte) 0xbb,
                                (byte) 0xcc, (byte) 0xdd, (byte) 0xee, (byte) 0xff };
                byte[] keyBytes = new byte[] {
                                0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                                0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f };

                SecretKeySpec key = new SecretKeySpec(keyBytes, "ARC4");
                Cipher cipher = Cipher.getInstance("ARC4", "BC");
                System.out.println("input text : "+ Utils.toHex(input));

                // encryption pass
                byte[] cipherText = new byte[input.length];
                cipher.init(Cipher.ENCRYPT_MODE, key);
                int ctLength = cipher.update(input, 0, input.length, cipherText, 0);
                ctLength += cipher.doFinal(cipherText, ctLength);
                System.out.println("cipher text : "+ Utils.toHex(cipherText)+" bytes: "+ctLength);

                // decryption pass
                byte[] plainText = new byte[ctLength];
                cipher.init(Cipher.DECRYPT_MODE, key);
                int ptLength = cipher.update(cipherText, 0, ctLength, plainText, 0);
                ptLength += cipher.doFinal(plainText, ptLength);
                System.out.println("plain text : "+ Utils.toHex(plainText)+" bytes: "+ptLength);
        }
}
```

◆ 출력결과

```
input text : 00112233445566778899aabbccddeeff
cipher text : e98d62ca03b77fbb8e423d7dc200c4b0 bytes: 16
plain text : 00112233445566778899aabbccddeeff bytes: 16
```

◆ 확인하고자 하는 바

=> ARC4 스트림 알고리즘 테스트, 다른 block cipher와 비슷하나 모드나 패딩이 필요 없음

```java
package chapter2;
import java.security.Key;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
public class KeyGeneratorExample {
        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, 0x07 };
                byte[] ivBytes = new byte[] {
                                0x00, 0x00, 0x00, 0x01, 0x04, 0x05, 0x06, 0x07,
                                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 };

                Cipher cipher = Cipher.getInstance("AES/CTR/NoPadding", "BC");
                KeyGenerator generator = KeyGenerator.getInstance("AES", "BC");
                generator.init(192);
                Key encryptionKey = generator.generateKey();
                System.out.println("key : "+ Utils.toHex(encryptionKey.getEncoded()));
                System.out.println("input : "+ Utils.toHex(input));

                // encryption pass
                cipher.init(Cipher.ENCRYPT_MODE, encryptionKey, new IvParameterSpec(ivBytes));
                byte[] cipherText = new byte[cipher.getOutputSize(input.length)];
                int ctLength = cipher.update(input, 0, input.length, cipherText, 0);
                ctLength += cipher.doFinal(cipherText, ctLength);

                // create our decryption key using information
                // extracted from the encryption Key
                Key decryptionkey = new SecretKeySpec(encryptionKey.getEncoded(),
                        encryptionKey.getAlgorithm());
                cipher.init(Cipher.DECRYPT_MODE, decryptionkey, new IvParameterSpec(ivBytes));
                byte[] plainText = new byte[cipher.getOutputSize(ctLength)];
                int ptLength = cipher.update(cipherText, 0, ctLength, plainText, 0);
                ptLength += cipher.doFinal(plainText, ptLength);
                System.out.println("plain : "+ Utils.toHex(plainText, ptLength) +" bytes: "+ptLength);
        }
}
```

◆ 출력결과

```
key : 1b33f26cc4f88a0538f0633a4561f09ecaf3a705181a1ad1
input : 000102030405060708090a0b0c0d0e0f0001020304050607
plain : 000102030405060708090a0b0c0d0e0f0001020304050607 bytes: 24
```

◆ 확인하고자 하는 바

=> KeyGenerator를 통해 AES key 를 생성 후 암복호화

◆ javax.crypto.KeyGenerator : 대칭 key 를 생성

i) getInstanc(String algorithm, String provider)

=> “ 알고리즘 ” , “ 프로바이더 ” 를 매개변수로 KeyGenerator를 생성

ii) init(int keysize)

=> “ key 사이즈 ” 를 매개변수로 key 사이즈를 명시

iii) generateKey()

=> Cipher 암복호화 엔진에 쓰일 Key를 생성

◆ java.security.Key

i) getEncoded()

=> key를 바이트배열로 반환

ii) getAlgorithm()

=> key생성시 명시하였던 알고리즘 반환

```java
package chapter2;
import java.security.Key;
import javax.crypto.Cipher;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.PBEParameterSpec;
import javax.crypto.spec.SecretKeySpec;
public class PBEWithparamsExample {
	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, 0x07 };
		byte[] KeyBytes = new byte[] {
				0x73, 0x2f, 0x2d, 0x33, (byte) 0xc8, 0x01, 0x73,
				0x2b, 0x72, 0x06, 0x75, 0x6c, (byte) 0xbd, 0x44,
				(byte) 0xf9, (byte) 0xc1, (byte) 0xc1, 0x03, (byte) 0xdd,
				(byte) 0xd9, 0x7c, 0x7c, (byte) 0xbe, (byte) 0x8e };
		byte[] ivBytes = new byte[] {
				(byte) 0xb0, 0x7b, (byte) 0xf5, 0x22, (byte) 0xc8,
				(byte) 0xd6, 0x08, (byte) 0xb8 };

		// encrypt the data using precalculated keys
		Cipher cEnc = Cipher.getInstance("DESede/CBC/PKCS7Padding", "BC");
		cEnc.init(Cipher.ENCRYPT_MODE, new SecretKeySpec(KeyBytes, "DESede"),
			new IvParameterSpec(ivBytes));
		byte[] out= cEnc.doFinal(input);

		// decrypt the data using PBE
		char[] password ="password".toCharArray();
		byte[] salt = new byte[] {
				0x7d, 0x60, 0x43, 0x5f,
				0x02, (byte) 0xe9, (byte) 0xe0, (byte) 0xae };
		int iterationCount =2048;
		PBEKeySpec pbeSpec =new PBEKeySpec(password);
		SecretKeyFactory keyFact = SecretKeyFactory
			.getInstance("PBEWithSHAAnd3KeyTripleDES", "BC");
		Cipher cDec = Cipher.getInstance("PBEWithSHAAnd3KeyTripleDES", "BC");
		Key sKey = keyFact.generateSecret(pbeSpec);
		cDec.init(Cipher.DECRYPT_MODE, sKey, new PBEParameterSpec(salt, iterationCount));
		System.out.println("cipher : "+Utils.toHex(out));
		System.out.println("gen key : "+Utils.toHex(sKey.getEncoded()));
		System.out.println("gen iv : "+Utils.toHex(cDec.getIV()));
		System.out.println("plain : "+Utils.toHex(cDec.doFinal(out)));
	}
}
```

◆ 출력결과

```
cipher : a7b955896f750665ba71eb50ac3071d9832a8b02760c600bf619a75a0697c87c
gen key : 732f2c32c801732a7307756dbc45f8c1c102dcd97c7cbf8f
gen iv : b07bf522c8d608b8
plain : 000102030405060708090a0b0c0d0e0f0001020304050607
```

◆ 확인하고자 하는 바

=> PassWord-Based Encryption 암복호화를 진행, 해당 KeyBytes 는 PBEKeySpec의 바이트배열 값.

문제는

```
byte[] KeyBytes = new byte[] {
                                0x73, 0x2f, 0x2c, 0x32, (byte) 0xc8, 0x01, 0x73,
                                0x2a, 0x73, 0x07, 0x75, 0x6d, (byte) 0xbc, 0x45,
                                (byte) 0xf8, (byte) 0xc1, (byte) 0xc1, 0x02, (byte) 0xdc,
                                (byte) 0xd9, 0x7c, 0x7c, (byte) 0xbf, (byte) 0x8f };
```

이것으로 암호화한 정보와

```
byte[] KeyBytes = new byte[] {
                                0x73, 0x2f, 0x2d, 0x33, (byte) 0xc8, 0x01, 0x73,
                                0x2b, 0x72, 0x06, 0x75, 0x6c, (byte) 0xbd, 0x44,
                                (byte) 0xf9, (byte) 0xc1, (byte) 0xc1, 0x03, (byte) 0xdd,
                                (byte) 0xd9, 0x7c, 0x7c, (byte) 0xbe, (byte) 0x8e };
```

이것으로 암호화한 정보가 같다.

why ?

```java
package chapter2;
import java.security.Key;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
public class SimpleWrapExample {
        public static void main(String[] args) throws Exception {
                // create a key to wrap
                KeyGenerator generator = KeyGenerator.getInstance("AES", "BC");
                generator.init(128);
                Key keyToBeWrapped = generator.generateKey();
                System.out.println("input : "+Utils.toHex(keyToBeWrapped.getEncoded()));

                // create a wrapper and do the wrapping
                Cipher cipher = Cipher.getInstance("AESWrap", "BC");
                KeyGenerator keyGen = KeyGenerator.getInstance("AES", "BC");
                keyGen.init(256);
                Key wrapKey = keyGen.generateKey();
                cipher.init(Cipher.WRAP_MODE, wrapKey);
                byte[] wrappedKey = cipher.wrap(keyToBeWrapped);
                System.out.println("wrapped : "+Utils.toHex(wrappedKey));

                // unwrap the wrapped key
                cipher.init(Cipher.UNWRAP_MODE, wrapKey);
                Key key = cipher.unwrap(wrappedKey, "AES", Cipher.SECRET_KEY);
                System.out.println("unwrapped: "+Utils.toHex(key.getEncoded()));
        }
}
```

◆ 출력결과

```
input : dcee75d652a22f04fe4ab89f4d7a440b
wrapped : 7babb10243894338c01d48b7ef9f5be35d3453ed33303cbe
unwrapped: dcee75d652a22f04fe4ab89f4d7a440b
```

◆ 확인하고자 하는 바

=> key를 wrapping 하고 unwrapping 함

◆ javax.crypto.Cipher

i) getInstance(String transformation, String provider)

=> “ 알고리즘이름/모드/패딩 ” , “ 프로바이더이름 ” 을 매개변수로 Cipher 객체화

ii) init(int opmode, Key key)

=> Cipher.WRAP_MODE or Cipher.UNWRAP_MODE, “ key ” 을 매개변수로 초기화 진행

iii) wrap(Key key)

=> “ wrap 하고자 하는 key ” 를 매개변수로 바이트배열의 wrapkey를 생성

iv) unwrap(byte[] wrappedKey, String wrappedKeyAlgorithm, int wrappedKeyType)

=> “ 바이트배열의 wrapkey ” , “ wrapkey의 알고리즘 ” , “ wrapkey의 타입 ” 을 매개변수 Key의 unwrapkey 생성

```java
package chapter2;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import javax.crypto.Cipher;
import javax.crypto.CipherInputStream;
import javax.crypto.CipherOutputStream;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
public class SimpleIOExample {
        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 };
                byte[] keyBytes = new byte[] {
                                0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                                0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
                                0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17 };
                byte[] ivBytes = new byte[] {
                                0x00, 0x01, 0x02, 0x03, 0x00, 0x01, 0x02, 0x03,
                                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 };
                SecretKeySpec key = new SecretKeySpec(keyBytes, "AES");
                IvParameterSpec ivSpec = new IvParameterSpec(ivBytes);
                Cipher cipher = Cipher.getInstance("AES/CTR/NoPadding", "BC");
                System.out.println("input : "+Utils.toHex(input));

                // encryption pass
                cipher.init(Cipher.ENCRYPT_MODE, key, ivSpec);
                ByteArrayInputStream bIn =new ByteArrayInputStream(input);
                CipherInputStream cIn =new CipherInputStream(bIn, cipher);
                ByteArrayOutputStream bOut =new ByteArrayOutputStream();
                int ch;
                while( (ch = cIn.read()) >=0 ){
                        bOut.write(ch);
                }
                byte[] cipherText = bOut.toByteArray();
                System.out.println("cipher : "+Utils.toHex(cipherText));

                // decryption pass
                cipher.init(Cipher.DECRYPT_MODE, key, ivSpec);
                bOut =new ByteArrayOutputStream();
                CipherOutputStream cOut =new CipherOutputStream(bOut, cipher);
                cOut.write(cipherText);
                cOut.close();
                System.out.println("plain : "+Utils.toHex(bOut.toByteArray()));
        }
}
```

◆ 출력결과

```
input : 000102030405060708090a0b0c0d0e0f00010203040506
cipher : bbfe17383cc002047c11be5dfc524e4ead5f2a887d197b
plain : 000102030405060708090a0b0c0d0e0f00010203040506
```

◆ 확인하고자 하는 바

=> I/O Stream을 통해 암복호화

◆ javax.crypto.CipherInputStream

i) CipherInputStream(InputStream is, Cipher c)

=> “ Input스트림 ”, Cipher 암복호화 엔진을 매개변수로 암호화된 InputStream 객체화

ii) read()

=> stream의 next byte를 읽음

◆ javax.crypto.CipherOutputStream

i) CipherOutputStream(OutputStream os, Cipher c)

=> “ Output스트림 ”, Cipher 암복호화 엔진을 매개변수로 암호화된 OutputStream 객체화

ii) write(byte[] b)

=> 바이트배열 b에 stream의 next byte를 씀

iii) close()

=> Cipher 암복호화 엔진의 doFinal() 과 같은 맥락, 누락 시 메시지가 잘리게 됨