정보보호개론(CSE441)
www.facebook.com/KoreaTech.Crypto

암호라이브러리 (Java, Python)

Note 08 addon

한국기술교육대학교 컴퓨터공학부 김상진

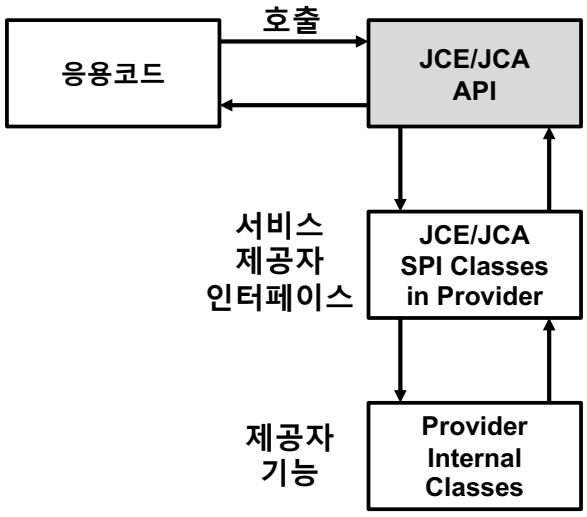
sangjin@koreatech.ac.kr
www.facebook.com/sangjin.kim.koreatech

교육목표

- Java 암호라이브러리
  - 대칭 암호, 해시함수, mac, 공개키 암호,
  - 전자서명, 인증서
  - 패스워드 기반 암호
  - bountycastle 라이브러리
- Python 암호라이브러리

JCA/JCE

- JCA(Java Cryptography Architecture)와 JCE(Java Cryptography Extension)은 프로그래머들이 응용을 개발할 때 사용할 수 있는 추상 계층(abstraction layer)을 제공함



JCA

- JCA는 제공자 기반 구조(provider-based architecture)라 하며, 다음과 같은 원리를 기반으로 설계되었음
  - 알고리즘 독립성
  - 알고리즘 확장성: 새 알고리즘을 쉽게 추가할 수 있는 구조
  - 구현 독립성: 프로그래머는 알고리즘을 구현한 제공자가 누구인지 알 필요가 없음
  - 구현 간 상호운영: 프로그래머가 서로 다른 제공자에 의해 제공되는구현을 사용하더라도 동작해야 함

Implementation Independence
Implementation Interoperability
Algorithm Extensibility

알고리즘 독립성 (1/2)

- 같은 종류의 여러 알고리즘을 동일한 메소드를 이용하여 암호연산을 수행할 수 있음
- 이를 위해 다음과 같이 암호연산을 분류함
  - 암호알고리즘
    - MessageDigest: 해시함수
    - Cipher: 암복호화 알고리즘
    - Signature: 전자서명
    - SecureRandom: 의사난수 비트 생성
    - Mac: Message Authentication Code
  - 키 관리 및 생성
    - KeyPairGenerator: 전자서명을 위한 키 쌍 생성
    - KeyGenerator: 알고리즘 전용 대칭키 생성
    - KeyFactory / SecretKeyFactory
    - KeyStore: 다양한 암호키 관리

알고리즘 독립성 (2/2)

  - 알고리즘 파라미터
    - AlgorithmParameters
    - AlgorithmParameterGenerator
  - PKI 관련
    - CertificateFactory: 인증서와 인증서폐지목록을 생성
    - CertPathBuilder
    - CertPathValidator
    - CertStore

알고리즘 독립성

```
try{
	MessageDigest hash = MessageDigest.getInstance("SHA-1");
	// MessageDigest hash = MessageDigest.getInstance("SHA-256");
	// MessageDigest hash = MessageDigest.getInstance("SHA3-256");
	hash.update(buffer);
	byte[] digest = md5.digest();
	for(int i = 0; i < digest.length; ++i) System.out.printf("%02X ", digest[i]);
}
catch(NoSuchAlgorithmException e){
	e.printStackTrace();
}
```

- JCA의 엔진(클래스)들은 java.security 패키지에 포함되어 있음
- 자바 라이브러리에서 사용할 수 있는 알고리즘 이름:
  https://docs.oracle.com/en/java/javase/20/docs/specs/security/standard-names.html
  - 자바 버전마다 차이가 있음

JCE

- JCE는 JCA를 확장한 것으로, 다음과 같은 엔진을 추가로 제공함
  - Cipher: 암호화/복호화
  - KeyGenerator: Cipher를 위한 비밀키 생성
  - SecretKeyFactory
  - KeyAgreement: 키 동의 프로토콜을 위한 클래스
  - Mac: 메시지 인증 코드
- JCE의 엔진은 javax.crypto 패키지에 포함되어 있음
- JDK 1.4부터 JCA/JCE 모두 기본적으로 포함되어 있음

제공자 인터페이스

- 이전 SUN은 자체적으로 암호라이브러리를 개발하지 않음
  - 따라서 암호알고리즘들은 제3의 제공자들이 개발하여 JDK와 함께 공급되거나 프로그래머들이 별도로 설치해야 함
- 프로그래머가 제공자 구현을 접근하는 두 가지 방법
  - 방법 1. (Opaque Algorithm Strategy): 프로그래머는 알고리즘의 이름을 통해 알고리즘을 요청하면 JCA가 사용할 제공자 구현을 선택함
  - 방법 2. (Transparent Algorithm Strategy): 프로그래머가 알고리즘의 이름과 제공자 이름까지 함께 지정함. 이 방법은 권장되지 않음

BouncyCastle (1/2)

- 다양한 암호알고리즘 라이브러리를 제공함
  - Java, C# 버전
- 자바 8이전
  - BouncyCastle을 사용하고 싶으면 www.bouncycastle.org를 방문하여 최신 provider 파일을 다운받아 java 디렉토리 jre/lib/ext에 jar 파일들을 복사하고, jre/lib/security/java.security 파일에 다음을 추가함
    security.provider.1x=org.bouncycastle.jce.provider.BouncyCastleProvider
    - bcprov-jdk18on-172.jar
    - bcprov-ext-jdk18on-172.jar
    - bcpkix.-jdk18on-172.jar
  - BouncyCastle 라이브러리를 사용하기 위해서는 java 사이트에서 unrestricted policy files를 다운받아 설치하여야 함

BouncyCastle (2/2)

- 자바 9 이후
  - BouncyCastle을 사용하고 싶으면 www.bouncycastle.org를 방문하여 최신 provider 파일을 다운받아 원하는 디렉토리에 저장
    - bcprov-jdk18on-172.jar
    - bcprov-ext-jdk18on-172.jar
    - bcpkix.-jdk18on-172.jar
  - Eclipse – Preference – Java – Build Path – User Libraries에 BC를 생성하고 위 3개 파일을 추가(add external jars)함
  - BC를 사용하고자 하는 프로젝트에서 Properties – Java Build Path – Libraries 탭에서 classpath에 위의 사용자 라이브러리 BC를 추가함
  - 코드에 Security.*addProvider*(new BouncyCastleProvider());
    문장을 추가하여 "BC" provider를 사용할 수 있도록 함

의사난수 비트 생성기

- 자바에서 난수는 보통 ThreadLocalRandom.current() 객체를 이용하거나 java.lang.Math에 정의된 random() 메소드나 java.util.Random 클래스를 사용함
  - 하지만 이들은 암호학적으로 안전하지 못함
- JCA에서는 java.security.SecureRandom 엔진을 사용함
  - 이 엔진은 java.util.Random 클래스를 상속받고 있어, nextInt()와 같은 매우 편리한 메소드들을 제공하고 있음
- 의사난수 비트 생성은 보통 실제 랜덤한 seed를 사용함
  - 이를 위해 사용자가 직접 setSeed 메소드를 사용할 수 있지만 일반적으로는 엔진이 안전한 seed를 자동으로 선택하도록 하는 것이 바람직함

의사난수 생성 예

- 예)

```
try{
        SecureRandom csprng = SecureRandom.getInstance("SHA1PRNG");
        boolean randBool = csprng.nextBoolean();
        int randInt = csprng.nextInt();
        byte[] randBytes = new byte[3];
        csprng.nextBytes(randomBytes);
}
catch(NoSuchAlgorithmException e){
        e.printStackTrace();
}
```


비밀키의 생성 (1/2)

- 비밀키는 알고리즘에 따라 키 길이가 다름
- 자바에서는 사용자가 키 길이를 별도로 제시하지 않아도 선택한 알고리즘에 필요한 비밀키를 생성하여 줌
  - 필요하면 특정한 알고리즘을 사용하여 키를 생성하도록 설정할 수 있음
    - init(int keySize)
    - init(SecureRandom random)
    - init(int keySize, SecureRandom random)
- 비밀키의 생성은 javax.crypto.KeyGenerator 엔진을 사용함
  - 예) KeyGenerator kg = KeyGenerator.getInstance("AES");
  - 키의 실제 생성은 generateKey 메소드를 이용하며, 그 결과는 SecretKey 클래스에 유지됨

비밀키의 생성 (2/2)

- 예)

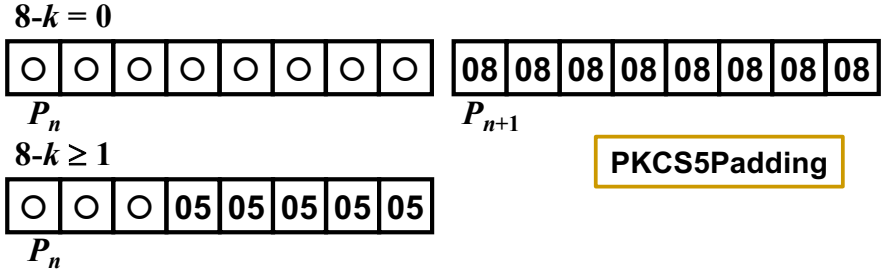
```
KeyGenerator kg  = KeyGenerator.getInstance("AES");
SecretKey key = kg.generateKey();
```

- generateKey() 메소드를 이용하여 생성된 키를 바로 알고리즘에 사용할 수 없음
- 이것을 key specification으로 변환해야 함. 이 변환은 생성된 키가 알고리즘에 사용하기에 보다 적합하도록 변환하는 것임
  - 이때 두 가지 방법이 있는데, 널리 사용하는 두 번째 방법만 소개함
  - 이 방법은 javax.crypto.spec.SecretKeySpec 엔진을 사용함
- 예)

```
KeyGenerator kg  = KeyGenerator.getInstance("AES");
SecretKey key = kg.generateKey();
SecretKeySpec keySpec = new SecretKeySpec(key.getEncoded(), "AES");
```


채우기

- 블록 암호 방식은 마지막 블록을 완전 블록으로 만들기 위해 채우기가 필요함
- JCE는 6가지 채우기 방법을 제공함
  - NoPadding: 사용자가 완전 블록임을 보장해 주어야 함
  - ISO10126Padding
  - PKCS1Padding, PKCS5Padding
  - SSL3Padding (예약만 되어 있고, 실제 구현되어 있지 않음)
  - OAEPWith<digest>And<mgf>Padding (RSA 관련)



암호화 모드

- JCE는 다음과 같은 암호 모드를 제공함
  - ECB
  - CBC
  - CCM: Counter/CBC Mode
  - CFB
  - CTR
  - CTS: Ciphertext Stealing Mode
  - GCM
  - OFB
  - KW, KWP
  - PCBC
  - NONE

Cipher 엔진 (1/2)

- Cipher 엔진은 비밀키 방식의 암호화/복호화를 하기 위해 사용됨
- 이 엔진을 사용하기 위해서는 먼저 사용할 알고리즘을 지정하여야 함
- 예)

```
Cipher AESCipher = Cipher.getInstance("AES");
```

- 이때 알고리즘 이름 뿐만 아니라 암호화 모드, 채우기 방식을 함께 지정 가능
- 예)

```
Cipher AESCipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
```

- 첫 번째 예처럼 암호화 모드와 채우기 방식을 지정하지 않으면 제공자에 의해 결정된 모드와 채우기 방식으로 암호화가 이루어짐
  - 하지만 이것에 의존하면 구현 상호운영이 보장되지 않을 수 있음

Cipher 엔진 (2/2)

- 알고리즘 지정을 통해 엔진 객체가 생성되면 이 객체를 초기화 해주어야 함
- 이때 init 메소드가 사용됨

```
public void init(int opmode, Key key)
```

  - opmode:
    - Cipher.ENCRYPT_MODE: 암호화 연산을 사용하고자 할 때
    - Cipher.DECRYPT_MODE: 복호화 연산을 사용하고자 할 때
- Ciper 엔진의 유용한 메소드
  - public final int getBlockSize();
    - 암호알고리즘의 블록 크기를 반환하여 줌
  - public final int getOutputSize(int inputLen);
    - 평문의 크기를 주면 필요한 암호문의 크기를 반환하여 줌

암호화 연산의 수행 (1/2)

- 방법 1. doFinal() 메소드만 호출
  - 전체 평문을 byte 배열로 확보하고 있는 경우

```
String plaintext = "This is a secret message!";
byte[] ciphertext = cipher.doFinal(plaintext.getBytes());
```

- 방법 2. 일련의 update() 메소드를 호출한 후에 doFinal() 메소드 호출
  - 한 번에 평문을 모두 제공할 수 없는 경우
  - update를 블록 단위로 하지 않아도 되며, 마지막 doFinal에 입력 데이터를 제공하고 있지 않고 마무리할 수 있음
  - 두 종류의 update

```
byte[] update(byte[] input, int inputOffset, int inputLen)
byte[] doFinal(byte[] input, int inputOffset, int inputLen)
byte[] doFinal()
```

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

암호화 연산의 수행 (2/2)

- 첫 번째 방법은 출력 byte 배열을 모두 결합해야 하기 때문에 번거로운 측면이 있음
- 예)

```
byte[] ciphertext = new byte[cipher.getOutputSize(input01.length+input02.length)];
byte[] output = cipher.update(input01, 0, input01.length);
int cipherLoc = output.length;
System.arraycopy(output, 0, ciphertext, 0, output.length);
output = cipher.update(input02, 0, input02.length);
System.arraycopy(output, 0, ciphertext, cipherLoc, output.length);
cipherLoc += output.length;
output = cipher.doFinal();
System.arraycopy(output, 0, ciphertext, cipherLoc, output.length);
```

```
byte[] ciphertext = new byte[cipher.getOutputSize(input01.length+input02.length)];
int ctLength = cipher.update(input01, 0, input01.length, ciphertext, 0);
ctLength += cipher.update(input02, 0, input02.length, ciphertext, ctLength);
ctLength += cipher.doFinal(ciphertext, ctLength);
```


AES ECB 모드

```
try{
    KeyGenerator kg  = KeyGenerator.getInstance("AES");
    SecretKey key = kg.generateKey();
    SecretKeySpec keySpec = new SecretKeySpec(key.getEncoded(), "AES");
    Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, keySpec);
    String plaintext = "This is a secret message!";
    byte[] ciphertext = cipher.doFinal(plaintext.getBytes());
    for(int i = 0; i < ciphertext.length; ++i)
        System.out.printf("%02X ", ciphertext[i]);
    System.out.println();
    cipher.init(Cipher.DECRYPT_MODE,keySpec);
    byte[] cleartext = cipher.doFinal(ciphertext);
    System.out.println(new String(cleartext));
}
catch(Exception e){
}
```

CBC/CTR 모드

- 두 모드는 모두 IV가 필요하며, IV가 필요할 경우에는 암호화와 복호화할 때 이것을 별도로 제공해 주어야 하며, 반드시 동일한 IV를 제공해 주어야 함
- JCE에서 IV는 javax.crypto.spce.IvParameterSpec을 사용하여 제공함
- IV를 생성하는 방법
  - 방법 1. 프로그래머가 직접 생성
  - 방법 2. cipher 객체 활용

```
SecureRandom csprng = SecureRandom.getInstance("SHA1PRNG");
byte[] ivBlock = new byte[cipher.getBlockSize()];
csprng.nextBytes(ivBlock);
IvParameterSpec ivSpec = new IvParameterSpec(ivBlock);
cipher.init(Cipher.ENCRYPT_MODE, keySpec, ivSpec);
```

```
cipher.init(Cipher.ENCRYPT_MODE, keySpec);
IvParameterSpec ivSpec = new IvParameterSpec(cipher.getIV());
```


AES CBC/CTR 모드

```
try{
    KeyGenerator kg  = KeyGenerator.getInstance("AES");
    SecretKey key = kg.generateKey();
    SecretKeySpec keySpec = new SecretKeySpec(key.getEncoded(), "AES");

    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    // Cipher cipher = Cipher.getInstance("AES/CTR/NoPadding");
    cipher.init(Cipher.ENCRYPT_MODE, keySpec);
    IvParameterSpec ivSpec = new IvParameterSpec(cipher.getIV());
    String plaintext = "This is a secret message!";
    byte[] ciphertext = cipher.doFinal(plaintext.getBytes());

    cipher.init(Cipher.DECRYPT_MODE, keySpec, ivSpec);
    byte[] cleartext = cipher.doFinal(ciphertext);
}
catch(Exception e){
}
```

AES GCM 모드

```
try{
    KeyGenerator kg  = KeyGenerator.getInstance("AES");
    SecretKey key = kg.generateKey();
    SecretKeySpec keySpec = new SecretKeySpec(key.getEncoded(), "AES");

    Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
    SecureRandom csprng = SecureRandom.getInstance("SHA1PRNG");
    byte[] N = new byte[12];
    csprng.nextBytes(N);
    GCMParameterSpec spec = new GCMParameterSpec(128, N);
    cipher.init(Cipher.ENCRYPT_MODE, spec);
    String plaintext = "This is a secret message!";
    byte[] aad = "associated data".getBytes();
    cipher.updateADD(add);
    byte[] ciphertext = cipher.doFinal(plaintext.getBytes());

    cipher.init(Cipher.DECRYPT_MODE, keySpec, spec);
    cipher.updateADD(add);
    byte[] cleartext = cipher.doFinal(ciphertext);
}
catch(Exception e){
}
```


MAC

```
try{
    KeyGenerator kg  = KeyGenerator.getInstance("HmacSHA1");
    SecretKey key = kg.generateKey();
    SecretKeySpec keySpec
        = new SecretKeySpec(key.getEncoded(), "HmacSHA1");

    Mac hmac = Mac.getInstance("HmacSHA1");
    hmac.init(keySpec);
    hmac.update("This is  a simple message.".getBytes());
    byte[] digest01 = hmac.doFinal();

    hmac.init(keySpec);
    hmac.update("This is  a simple message".getBytes());
    byte[] digest02 = hmac.doFinal();
    System.out.printf("Verified = %s%n",
        MessageDigest.isEqual(digest01, digest02));
}
catch(Exception e){
}
```

RSA

```
SecureRandom random = SecureRandom.getInstance("SHA1PRNG");
KeyPairGenerator kpg  = KeyPairGenerator.getInstance("RSA");
kpg.initialize(2048, random);
// kpg.initialize(new RSAKeyGenParameterSpec(
//         2048, RSAKeyGenParameterSpec.F0), random);
// F0: 3, F4: 65537
KeyPair keyPair = kpg.genKeyPair();

Cipher cipher = Cipher.getInstance("RSA/ECB/NoPadding");
// Cipher cipher = Cipher.getInstance(
//    "RSA/ECB/OAEPwithSHA-256andMGF1Padding");

cipher.init(Cipher.ENCRYPT_MODE, pubKey);
byte[] plaintext = {(byte)0x00, (byte)0xBE, (byte)0xBE};
byte[] ciphertext = cipher.doFinal(plaintext);

cipher.init(Cipher.DECRYPT_MODE, privKey);
byte[] cleartext = cipher.doFinal(ciphertext);
```


ElGamal

- 이산대수 기반 알고리즘 중 가장 유명한 알고리즘
  - 하지만 자바 라이브러리는 직접적으로 ElGamal 암호알고리즘을 제공하지 않음
    - BC provider를 사용할 수 있음
  - 이산대수 기반이기 때문에 DSA나 DiffieHellman 키 동의를 이용하여 필요한 키는 자바 라이브러리로 생성 가능

```
SecureRandom csprng = SecureRandom.getInstance("SHA1PRNG");
KeyPairGenerator kpg = KeyPairGenerator.getInstance("DH");
kpg.initialize(2048, csprng);
KeyPair keyPair = kpg.generateKeyPair();
DHPublicKey pubKey = (DHPublicKey)keyPair.getPublic();
DHPrivateKey privKey = (DHPrivateKey)keyPair.getPrivate();
```

    - 필요한 암호는 직접 BigInteger 연산을 이용하여 구현

ElGamal (1/2)

```
KeyPairGenerator kpg = KeyPairGenerator.getInstance("ElGamal", "BC");
final String modp2048 = (
      "FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1" +
      "29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD" +
      "EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245" +
      "E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED" +
      "EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D" +
      "C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F" +
      "83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D" +
      "670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B" +
      "E39E772C 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9" +
      "DE2BCBF6 95581718 3995497C EA956AE5 15D22618 98FA0510" +
      "15728E5A 8AACAA68 FFFFFFFF FFFFFFFF")
      .replaceAll("\\s", "");
BigInteger p = new BigInteger(modp2048, 16);
BigInteger g = BigInteger.valueOf(2L);
ElGamalParameterSpec params = new ElGamalParameterSpec(p, g);
kpg.initialize(params);
KeyPair keyPair = kpg.generateKeyPair();
```

RFC 3526
RFC 5114

```
KeyPairGenerator kpg = KeyPairGenerator.getInstance("ElGamal", "BC");
SecureRandom random = new SecureRandom();
kpg.initialize(512, random);
KeyPair keyPair = kpg.generateKeyPair();
```


ElGamal (2/2)

```
ElGamalPublicKey pubKey = (ElGamalPublicKey)keyPair.getPublic();

Cipher cipher = Cipher.getInstance(
      "ElGamal/None/OAEPWithSHA1AndMGF1Padding", "BC");
byte[] plaintext = {(byte)0x00, (byte)0xBE, (byte)0xBE};
SecureRandom random = new SecureRandom();
cipher.init(Cipher.ENCRYPT_MODE, pubKey, random);
```

```
ElGamalPrivateKey privKey = (ElGamalPrivateKey)keyPair.getPrivate();

Cipher cipher = Cipher.getInstance(
      "ElGamal/None/OAEPWithSHA1AndMGF1Padding", "BC");
cipher.init(Cipher.DECRYPT_MODE, privKey);
byte[] cleartext = cipher.doFinal(ciphertext);
```

Key Agreement

```
KeyPairGenerator kpg = KeyPairGenerator.getInstance("EC");
kpg.initialize(256);
KeyPair alicekp = kpg.generateKeyPair();
KeyPair bobkp = kpg.generateKeyPair();
```

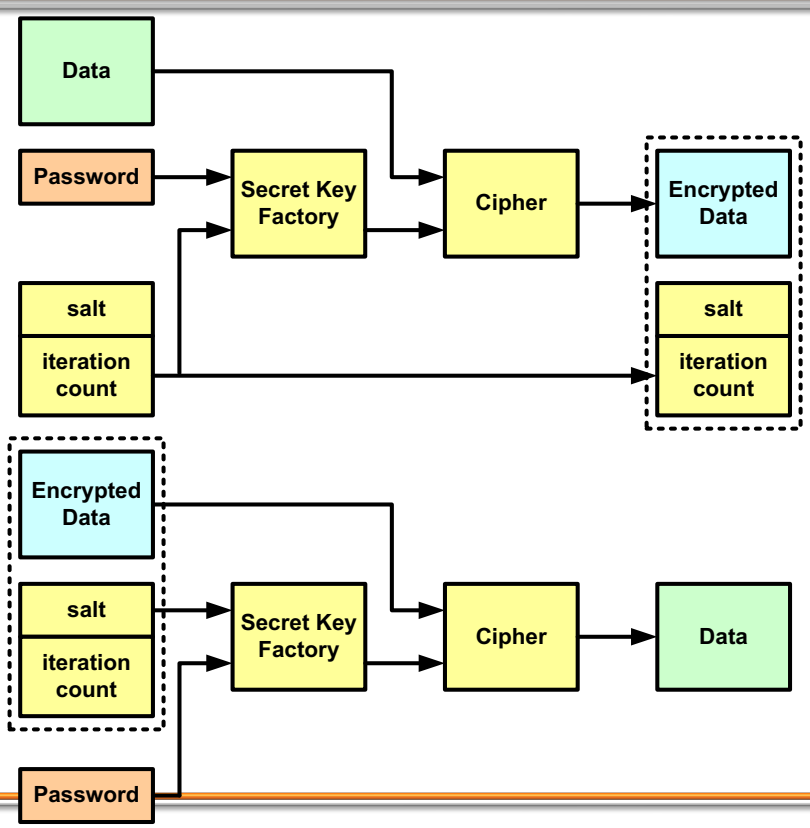
```
KeyAgreement ka = KeyAgreement.getInstance("ECDH");
ka.init(alicekp.getPrivate());
ka.doPhase(bobkp.getPublic(), true);
byte[] secret = ka.generateSecret();
byte[] key = HKDF(secret, "alice bob", 16);
SecretKeySpec keySpec = new SecretKeySpec(key, "AES");
```


패스워드 기반 암호화

- 기본적인 구조

Salt:
- 패스워드를 저장할 때
- 사용되는 salt와 동일 용도.
- 암호문과 함께 저장
- 동일 패스워드를 사용하여도
- 매번 다른 키를 사용함

Iteration Count:
- 전사공격을 어렵게 하기 위한 요소



PBE (1/2)

- 패스워드는 char 배열로 나타내야 하며, salt와 iteration 값을 제공해야 함
- salt와 iteration은 사전공격을 더욱 어렵도록 만들어 주는 역할을 하며, salt의 크기는 사용하는 암호알고리즘의 블록 크기와 같음

```
try{
    Cipher cipher = Cipher.getInstance("PBEWithHmac256AndAES_128");
    String password = "babo";
    char[] pwd = password.toCharArray();
    int iterations = 1000;
    SecureRandom csprng = SecureRandom.getInstance("SHA1PRNG");
    byte[] salt = new byte[cipher.getBlockSize()];
    csprng.nextBytes(salt);
    PBEParameterSpec pbeParamSpec = new PBEParameterSpec(salt, iterations);
    PBEKeySpec pbeKeySpec = new PBEKeySpec(pwd);
    ...
}
catch(){
}
```

```
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.PBEParameterSpec;
```


PBE (2/2)

- PBEKeySpec가 준비되면 실제 암호화에 사용될 암호키를 생성해야 한다. 이 키는 javax.crypto.SecretKeyFactory를 이용함

```
SecretKeyFactory factory
    = SecretKeyFactory.getInstance("PBEWithHmac256AndAES_128");
SecretKey key = factory.generateSecret(pbeKeySpec);
```

- 암/복호화는 기존과 유사하게 이루어짐

```
cipher.init(Cipher.ENCRYPT_MODE, key, pbeParamSpec);
String plaintext = "This is a secret message!";
byte[] ciphertext = cipher.doFinal(plaintext.getBytes());
for(byte b: ciphertext) System.out.printf("%02X ", b);
System.out.println();
cipher.init(Cipher.DECRYPT_MODE, key, pbeParamSpec);
byte[] cleartext = cipher.doFinal(ciphertext);
for(byte b: cleartext) System.out.print((char)b);
System.out.println();
```

Python Crypto Library

- PyCrypto
- PyCryptodome 3.18
  - PyCrypto 보다 더 많은 최신 모드와 알고리즘 제공
  - https://www.pycryptodome.org/en/latest/
  - 제공 알고리즘: https://www.pycryptodome.org/en/latest/src/features.html

AES – CBC MODE

```
key = Random.get_random_bytes(16)
Iv = Random.get_random_bytes(16)
```

```
cipher = AES.new(key, AES.MODE_CBC, iv)
plaintext = 'Kim Sangjin is Handsome'
pad_plaintext = Padding.pad(plaintext.encode(), 16)
ciphertext = cipher.encrypt(pad_plaintext)
```

```
cipher = AES.new(key, AES.MODE_CBC, iv)
decrypt_pad_text  = cipher.decrypt(ciphertext);
decrypted_text = Padding.unpad(decrypted_pad_text, 16)
```

AES – EAX MODE

```
key = Random.get_random_bytes(AES.key_size[0])
Iv = Random.get_random_bytes(AES.block_size)
```

```
cipher = AES.new(key, AES.MODE_EAX, iv)
plaintext = "Kim Sangjin is Handsome"
aad = "associated data"
cipher.update(aad.encode())
ciphertext, mactag = cipher.encrypt_and_digest(plaintext.encode())
```

```
cipher = AES.new(key, AES.MODE_EAX, iv)
cipher.update(aad.encode())
decrypted_text  = cipher.decrypt_and_verify(ciphertext, mactag);
```


Hash, MAC

```
hash = SHA256.new()
hash.update("Sangjin is Handsome".encode())
print(hash.hexdigest())
```

```
key = Random.new().read(16)
mac = HMAC.new(key, SHA256)
mac.update("Sangjin is Handsome".encode())
macval = mac.digest()
print(bytes(macval).hex())
```

```
mac = HMAC.new(key, SHA256)
mac.update('Kim Sangjin'.encode())
try:
    mac.verify(macval))
    print("The message is authentic")
except:
    print("The message or the key is wrong")
```

RSA

```
rng = Random.new().read
RSAKeyPair = RSA.generate(1024, rng)
```

```
cipher_rsa = PKCS1_OAEP.new(RSAKeyPair)
ctext = cipher_rsa.encrypt("Kim Sangjin",encode())
ptext = cipher_rsa.decrypt(ctext)
```

```
sigtext = "hello"
w = SHA256.new(sigtext,encode())
sigblock = PKCS1_PSS.new(RSAKeyPair).sign(w)
verifier = PKCS1_PSS.new(RSAKeyPair)
try:
    verifier.verify(w, sigblock)
    print("the signature is authentic")
except (ValueError, TypeError):
    print("the signature is not authentic")
```