Encryption in Java

AES encryption/decryption (ECB mode)

```java
public static byte[] encryptMessageECB(byte[] plaintext, byte[] key)
throws Exception
{
    /** Create a Key object from key bytes **/
    SecretKeySpec aesKey = new SecretKeySpec(key, "AES");

    /** Create a cipher object **/
    Cipher myAES = Cipher.getInstance("AES/ECB/NoPadding");

    /** Initialize cipher object for encryption **/
    myAES.init(Cipher.ENCRYPT_MODE, aesKey);

    /** Create ciphertext buffer **/
    byte[] ciphertext = new byte[myAES.getOutputSize(plaintext.length)];

    /** Encrypt message **/
    myAES.doFinal(plaintext, 0, plaintext.length, ciphertext);

    /** Return ciphertext **/
    return ciphertext;
}
```

```java
public static byte[] decryptMessageECB(byte[] ciphertext, byte[] key)
throws Exception
{
    /** Create a Key object from key bytes **/
    SecretKeySpec aesKey = new SecretKeySpec(key, "AES");

    /** Create a cipher object **/
    Cipher myAES = Cipher.getInstance("AES/ECB/NoPadding");

    /** Initialize cipher object for decryption **/
    myAES.init(Cipher.DECRYPT_MODE, aesKey);

    /** Create plaintext buffer **/
    byte[] plaintext = new byte[myAES.getOutputSize(ciphertext.length)];

    /** Decrypt message **/
    myAES.doFinal(ciphertext, 0, ciphertext.length, plaintext);

    /** Return plaintext **/
    return plaintext;
}
```

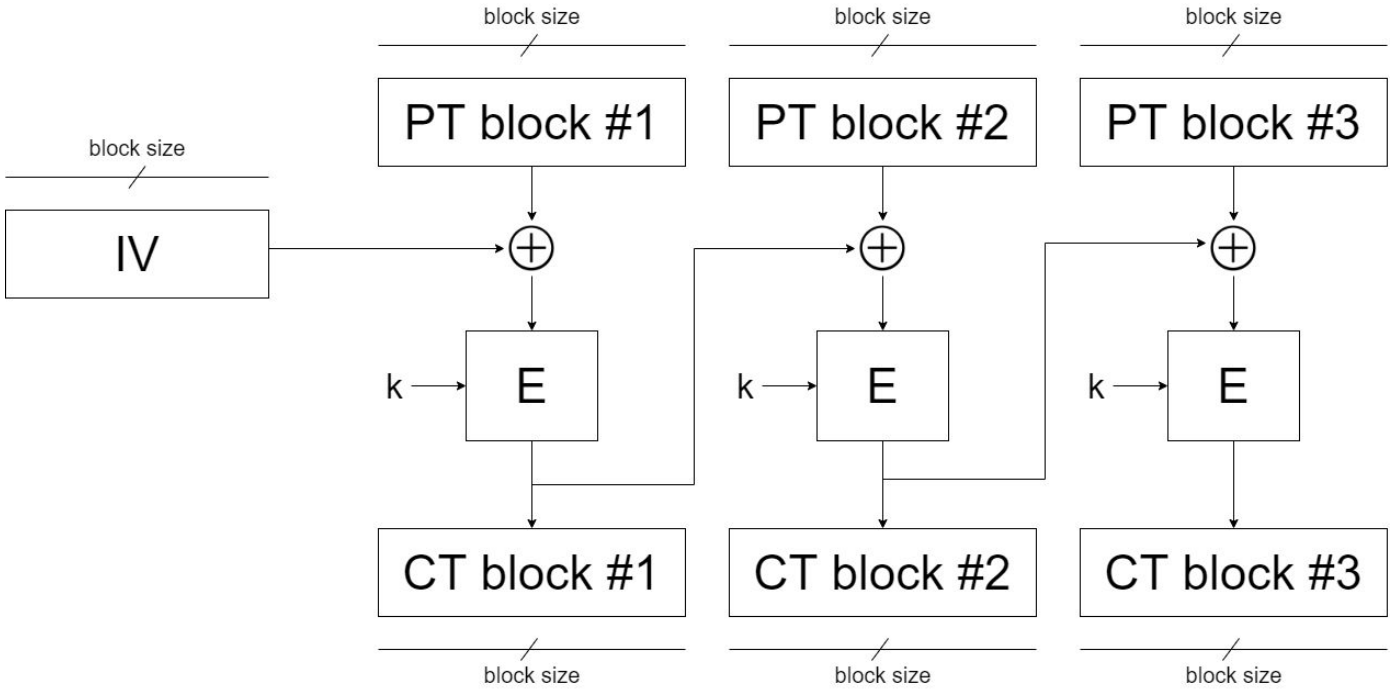
AES encryption/decryption (CBC mode)

```java
public static byte[] encryptMessageCBC(byte[] plaintext, byte[] key, byte[] iv)
throws Exception
{
    /** Create a Key object from key bytes **/
    SecretKeySpec aesKey = new SecretKeySpec(key, "AES");

    /** Create an IvParameterSpec object from iv bytes **/
    IvParameterSpec aesIv = new IvParameterSpec(iv);

    /** Create a cipher object **/
    Cipher myAES = Cipher.getInstance("AES/CBC/NoPadding");

    /** Initialize cipher object for encryption **/
    myAES.init(Cipher.ENCRYPT_MODE, aesKey, aesIv);

    /** Create ciphertext buffer **/
    byte[] ciphertext = new byte[myAES.getOutputSize(plaintext.length)];

    /** Encrypt message **/
    myAES.doFinal(plaintext, 0, plaintext.length, ciphertext);

    /** Return ciphertext **/
    return ciphertext;
}
```

```java
public static byte[] decryptMessageCBC(byte[] ciphertext, byte[] key, byte[] iv)
throws Exception
{
    /** Create a Key object from key bytes **/
    SecretKeySpec aesKey = new SecretKeySpec(key, "AES");

    /** Create an IvParameterSpec object from iv bytes **/
    IvParameterSpec aesIv = new IvParameterSpec(iv);

    /** Create a cipher object **/
    Cipher myAES = Cipher.getInstance("AES/CBC/NoPadding");

    /** Initialize cipher object for decryption **/
    myAES.init(Cipher.DECRYPT_MODE, aesKey, aesIv);

    /** Create plaintext buffer **/
    byte[] plaintext = new byte[myAES.getOutputSize(ciphertext.length)];

    /** Decrypt message **/
    myAES.doFinal(ciphertext, 0, ciphertext.length, plaintext);

    /** Return plaintext **/
    return plaintext;
}
```

Recall CBC encryption mode

Cipher block chaining (CBC) w/ random IV - guarantees all CT blocks are different



RSA key generation

```
public static Key[] generateRsaKeys() throws Exception
{
    SecureRandom myPRNG = new SecureRandom();

    /** Create and initialize KeyPairGenerator object */
    KeyPairGenerator myRSAKeyGen = KeyPairGenerator.getInstance("RSA");
    myRSAKeyGen.initialize(2048, myPRNG);

    /** Generate key pair **/
    KeyPair RsaKeyPair = myRSAKeyGen.generateKeyPair();

    /** Return keys **/
    Key[] keys = new Key[2];
    keys[0] = RsaKeyPair.getPublic();
    keys[1] = RsaKeyPair.getPrivate();

    return keys;
}
```

RSA encryption/decryption

```
public static byte[] encryptMessageRSA(byte[] plaintext, Key publicKey)
throws Exception
{
    SecureRandom myPRNG = new SecureRandom();

    /** Create a cipher object **/
    Cipher myRSA = Cipher.getInstance("RSA/ECB/PKCS1Padding");

    /** Initialize cipher object for encryption **/
    myRSA.init(Cipher.ENCRYPT_MODE, publicKey, myPRNG);

    /** Create ciphertext buffer **/
    byte[] ciphertext = new byte[myRSA.getOutputSize(plaintext.length)];

    /** Encrypt message **/
    myRSA.doFinal(plaintext, 0, plaintext.length, ciphertext);

    /** Return ciphertext **/
    return ciphertext;
}
```

```
public static byte[] decryptMessageRSA(byte[] ciphertext, Key privateKey)
throws Exception
{
    /** Create a cipher object **/
    Cipher myRSA = Cipher.getInstance("RSA/ECB/PKCS1Padding");

    /** Initialize cipher object for decryption **/
    myRSA.init(Cipher.DECRYPT_MODE, privateKey);

    /** Create plaintext buffer **/
    byte[] plaintext = new byte[myRSA.getOutputSize(ciphertext.length)];

    /** Decrypt message **/
    myRSA.doFinal(ciphertext, 0, ciphertext.length, plaintext);

    /** Return plaintext **/
    return plaintext;
}
```

Password-based Key Derivation Functions (PBKDF)

Goal: build a function that securely derives cryptographic keys from user passwords

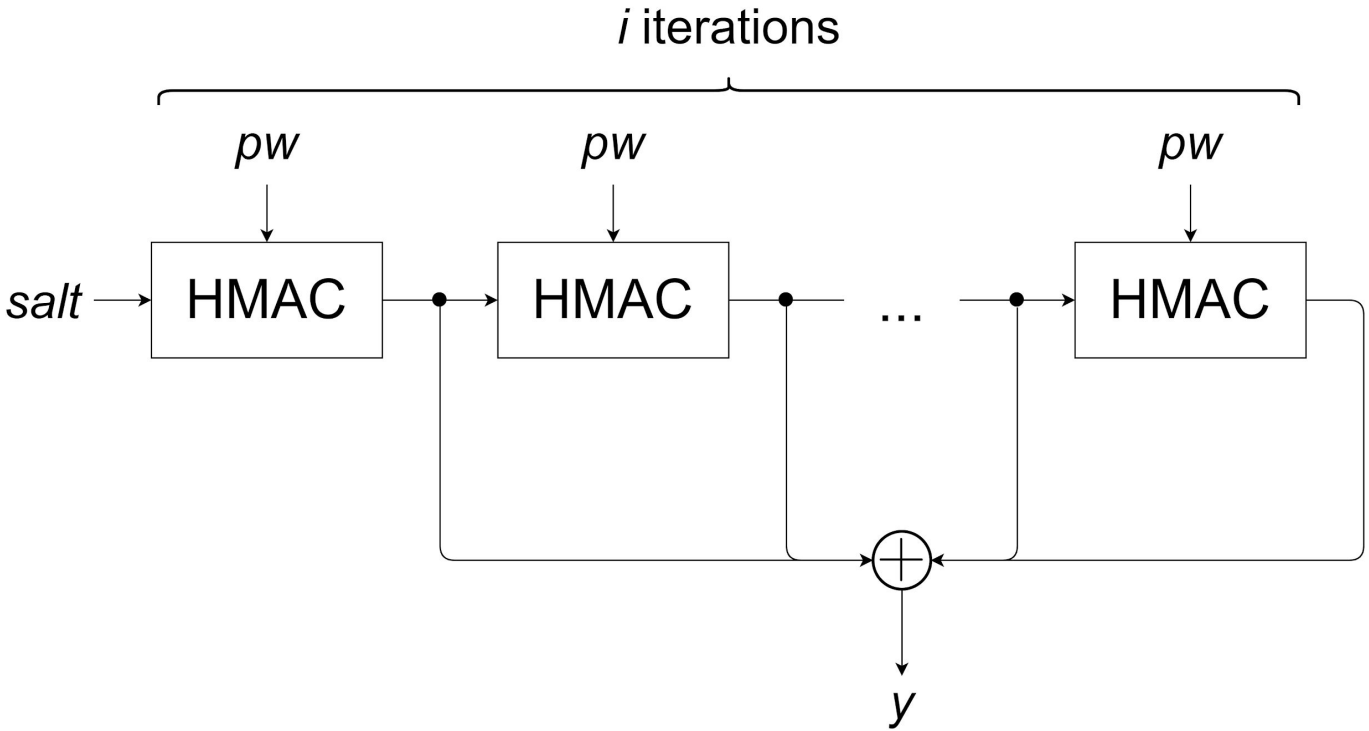
Example of use: encrypting files using winRAR (recall L2)

Security requirements

- the generated keys must have high entropy..
  ***uses HMAC to generate the output*
- the function must be resistant to offline attacks (e.g. pre-computed dictionaries)
  ***uses salt values*
- the function must be slow to make brute forcing hard
  ***runs the HMAC alg. multiple times, e.g., 10000 times*

***PBKDF2 w. HMAC*

PBKDF2 with HMAC



Next week: 1st evaluation

Example of questions/exercises:

1. What's the purpose of *salts* in UNIX/Linux password authentication systems?
2. Confidentiality/integrity/authenticity → definitions/exercises
3. Symmetric encryption w. block ciphers → CBC vs. ECB (possibly with implementation)
4. Write a C# **console application** that encrypts/decrypts/signs/verifies some data using (a)symmetric cryptography
5. Authenticated encryption → implementation
6. Write a Java program that decrypts the ciphertext *c,* knowing that it was encrypted using the symmetric scheme *X* and the key *k* derived from password *p*