加密學與 Java

`javax.crypto` 包在 Java 中提供了一組用於執行加密操作的類和方法，例如加密、解密、密鑰生成和消息驗證。以下是如何使用 `javax.crypto` 的全面指南，包括對常見用例（如對稱加密、非對稱加密和消息驗證）的說明和實際範例。

什麼是 javax.crypto？

`javax.crypto` 包是 Java 加密架構（JCA）的一部分，並提供通過加密實現安全通信的工具。它支持：- **對稱加密**：使用相同的密鑰進行加密和解密（例如，AES、DES）。- **非對稱加密**：使用公鑰/私鑰對（例如，RSA）。- **消息驗證**：確保數據完整性和真實性（例如，HMAC）。- **密鑰生成和管理**：創建和處理加密密鑰的工具。

要使用 `javax.crypto`，您需要：1. 選擇加密算法。2. 生成或獲取必要的密鑰。3. 使用提供的類（例如，`Cipher`、`KeyGenerator`、`Mac`）執行操作。

以下是常見情況的逐步範例。

---

1. 使用 AES 的對稱加密

對稱加密使用單個密鑰進行加密和解密。以下是如何使用 AES（高級加密標準）和 `Cipher` 類在 CBC 模式和 PKCS5 填充下加密和解密字符串的方法。

步驟

- 生成密鑰。
- 創建並初始化 `Cipher` 實例。
- 加密和解密數據。

範例代碼

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import java.security.SecureRandom;
import java.nio.charset.StandardCharsets;
```

```java
public class SymmetricEncryptionExample {
    public static void main(String[] args) throws Exception {
        // 步驟 1：為 AES 生成密鑰
        KeyGenerator keyGen = KeyGenerator.getInstance("AES");
        keyGen.init(128); // 128 位密鑰
        SecretKey secretKey = keyGen.generateKey();

        // 步驟 2：生成隨機初始化向量（IV）
        SecureRandom random = new SecureRandom();
        byte[] iv = new byte[16]; // AES 塊大小為 16 字節
        random.nextBytes(iv);
        IvParameterSpec ivSpec = new IvParameterSpec(iv);

        // 步驟 3：為加密創建並初始化 Cipher
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivSpec);

        // 步驟 4：加密數據
        String plaintext = "Hello, World!";
        byte[] plaintextBytes = plaintext.getBytes(StandardCharsets.UTF_8);
        byte[] ciphertext = cipher.doFinal(plaintextBytes);

        // 步驟 5：使用相同的 IV 初始化 Cipher 進行解密
        cipher.init(Cipher.DECRYPT_MODE, secretKey, ivSpec);
        byte[] decryptedBytes = cipher.doFinal(ciphertext);
        String decryptedText = new String(decryptedBytes, StandardCharsets.UTF_8);

        // 輸出結果
        System.out.println(" 原始: " + plaintext);
        System.out.println(" 解密: " + decryptedText);
    }
}
```

關鍵點

- **算法**："AES/CBC/PKCS5Padding" 指定 AES 使用 CBC 模式和填充來處理不是塊大小的倍數的數據。
- **IV**：初始化向量（IV）必須對加密是隨機的，並且在解密時重用。它通常預先附加到密文或單獨傳輸。
- **密鑰管理**：在實際應用中，安全地與接收方共享 secretKey。

2. 使用 RSA 的非對稱加密

非對稱加密使用公鑰進行加密，使用私鑰進行解密。以下是使用 RSA 的範例。

步驟

- 生成公鑰/私鑰對。
- 使用公鑰加密。
- 使用私鑰解密。

範例代碼

```
import javax.crypto.Cipher;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.nio.charset.StandardCharsets;

public class AsymmetricEncryptionExample {
    public static void main(String[] args) throws Exception {
        // 步驟 1：生成 RSA 鑰對
        KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("RSA");
        keyPairGen.initialize(2048); // 2048 位密鑰
        KeyPair keyPair = keyPairGen.generateKeyPair();
        PublicKey publicKey = keyPair.getPublic();
        PrivateKey privateKey = keyPair.getPrivate();

        // 步驟 2：使用公鑰加密
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);
        String plaintext = "Secret Message";
        byte[] plaintextBytes = plaintext.getBytes(StandardCharsets.UTF_8);
        byte[] ciphertext = cipher.doFinal(plaintextBytes);

        // 步驟 3：使用私鑰解密
        cipher.init(Cipher.DECRYPT_MODE, privateKey);
        byte[] decryptedBytes = cipher.doFinal(ciphertext);
        String decryptedText = new String(decryptedBytes, StandardCharsets.UTF_8);
```

```
        // 輸出結果
        System.out.println(" 原始: " + plaintext);
        System.out.println(" 解密: " + decryptedText);
    }
}
```

關鍵點

- **大小限制**：RSA 只能加密小於密鑰大小的數據（例如，對於 2048 位密鑰，約 245 字節）。對於較大的數據，使用混合加密（使用對稱密鑰加密數據，然後使用 RSA 加密該密鑰）。
- **密鑰分配**：公開公鑰；保密私鑰。

---

3. 使用 HMAC 的消息驗證

消息驗證碼（MAC）確保數據完整性和真實性。以下是如何使用 `Mac` 使用 HMAC-SHA256。

範例代碼

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.StandardCharsets;
import java.util.Base64;

public class MacExample {
    public static void main(String[] args) throws Exception {
        // 步驟 1：創建密鑰
        String secret = "mysecretkey";
        SecretKeySpec secretKey = new SecretKeySpec(secret.getBytes(StandardCharsets.UTF_8), "HmacSHA256");

        // 步驟 2：使用密鑰初始化 Mac
        Mac mac = Mac.getInstance("HmacSHA256");
        mac.init(secretKey);

        // 步驟 3：計算數據的 MAC
        String data = "Data to authenticate";
        byte[] macValue = mac.doFinal(data.getBytes(StandardCharsets.UTF_8));
        String macBase64 = Base64.getEncoder().encodeToString(macValue);
```

```
        // 輸出結果
        System.out.println("MAC: " + macBase64);
    }
}
```

關鍵點

- **驗證**：接收方使用相同的密鑰和數據重新計算 MAC；如果匹配，則數據是真實且未被篡改。
- **密鑰**：使用共享的密鑰，事先安全地分發。

---

4. 加密/解密流

對於大數據（例如，文件），使用 `CipherInputStream` 或 `CipherOutputStream`。

範例代碼（加密文件）

```
import javax.crypto.Cipher;
import javax.crypto.CipherOutputStream;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.security.SecureRandom;

public class StreamEncryptionExample {
    public static void main(String[] args) throws Exception {
        // 生成密鑰和 IV
        KeyGenerator keyGen = KeyGenerator.getInstance("AES");
        keyGen.init(128);
        SecretKey secretKey = keyGen.generateKey();
        byte[] iv = new byte[16];
        new SecureRandom().nextBytes(iv);
        IvParameterSpec ivSpec = new IvParameterSpec(iv);

        // 初始化 Cipher
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivSpec);
```

```
        // 加密文件
        try (FileInputStream fis = new FileInputStream("input.txt");
             FileOutputStream fos = new FileOutputStream("encrypted.txt");
             CipherOutputStream cos = new CipherOutputStream(fos, cipher)) {
            byte[] buffer = new byte[1024];
            int bytesRead;
            while ((bytesRead = fis.read(buffer)) != -1) {
                cos.write(buffer, 0, bytesRead);
            }
        }
    }
}
```

關鍵點

- **流**：使用 `CipherOutputStream` 進行加密和 `CipherInputStream` 進行解密以逐步處理數據。
- **IV 處理**：將 IV 存儲在加密文件中（例如，預先附加）。

---

5. 基於密碼的加密（PBE）

使用 `SecretKeyFactory` 從密碼派生密鑰。

範例代碼

```
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.SecretKeySpec;
import java.security.SecureRandom;
import java.nio.charset.StandardCharsets;

public class PBEExample {
    public static void main(String[] args) throws Exception {
        // 密碼和鹽
        char[] password = "mysecretpassword".toCharArray();
        byte[] salt = new byte[16];
```

```
        new SecureRandom().nextBytes(salt);

        // 從密碼派生密鑰
        PBEKeySpec pbeKeySpec = new PBEKeySpec(password, salt, 10000, 256); // 256 位密鑰
        SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
        SecretKey tempKey = factory.generateSecret(pbeKeySpec);
        SecretKey secretKey = new SecretKeySpec(tempKey.getEncoded(), "AES");

        // 使用派生的密鑰加密
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        String plaintext = "Hello from PBE";
        byte[] ciphertext = cipher.doFinal(plaintext.getBytes(StandardCharsets.UTF_8));

        System.out.println(" 加密: " + Base64.getEncoder().encodeToString(ciphertext));
    }
}
```

關鍵點

- **鹽**：隨機化密鑰派生；將其存儲在加密數據中。
- **迭代**：增加計算成本以抵禦暴力攻擊（例如，10,000）。

---

javax.crypto 中的關鍵類

- `Cipher`：執行加密和解密。
- `KeyGenerator`：生成對稱密鑰（例如，AES）。
- `KeyPairGenerator`：生成非對稱密鑰對（例如，RSA）。
- `Mac`：計算消息驗證碼。
- `SecretKeyFactory`：派生密鑰（例如，從密碼）。
- `SecureRandom`：生成加密安全的隨機數。

最佳實踐

- **異常處理**：使用 try-catch 塊處理異常（例如，`NoSuchAlgorithmException`、`InvalidKeyException` 等）。
- **密鑰管理**：安全地存儲密鑰（例如，在 `KeyStore` 中）並永遠不要硬編碼。

- **算法選擇**：使用安全算法（例如，AES-256、RSA-2048）和模式（例如，CBC 使用 IV）。
- **提供者**：默認的 SunJCE 提供者足夠，但您可以使用其他提供者（例如，BouncyCastle）通過 `Cipher.getInstance("AES", "BC")`。

結論

要使用 `javax.crypto`，選擇適合您需求的加密算法，生成或獲取密鑰，並利用類（例如，`Cipher`、`KeyGenerator` 和 `Mac`）執行操作。無論是使用 AES 對稱加密數據，使用 RSA 非對稱加密數據，還是使用 HMAC 確保完整性，`javax.crypto` 提供了工具，配合適當的初始化和安全密鑰管理，可以在 Java 中實現強大的加密。