Java面向对象程序设计课程设计实验报告十

实验人：王维

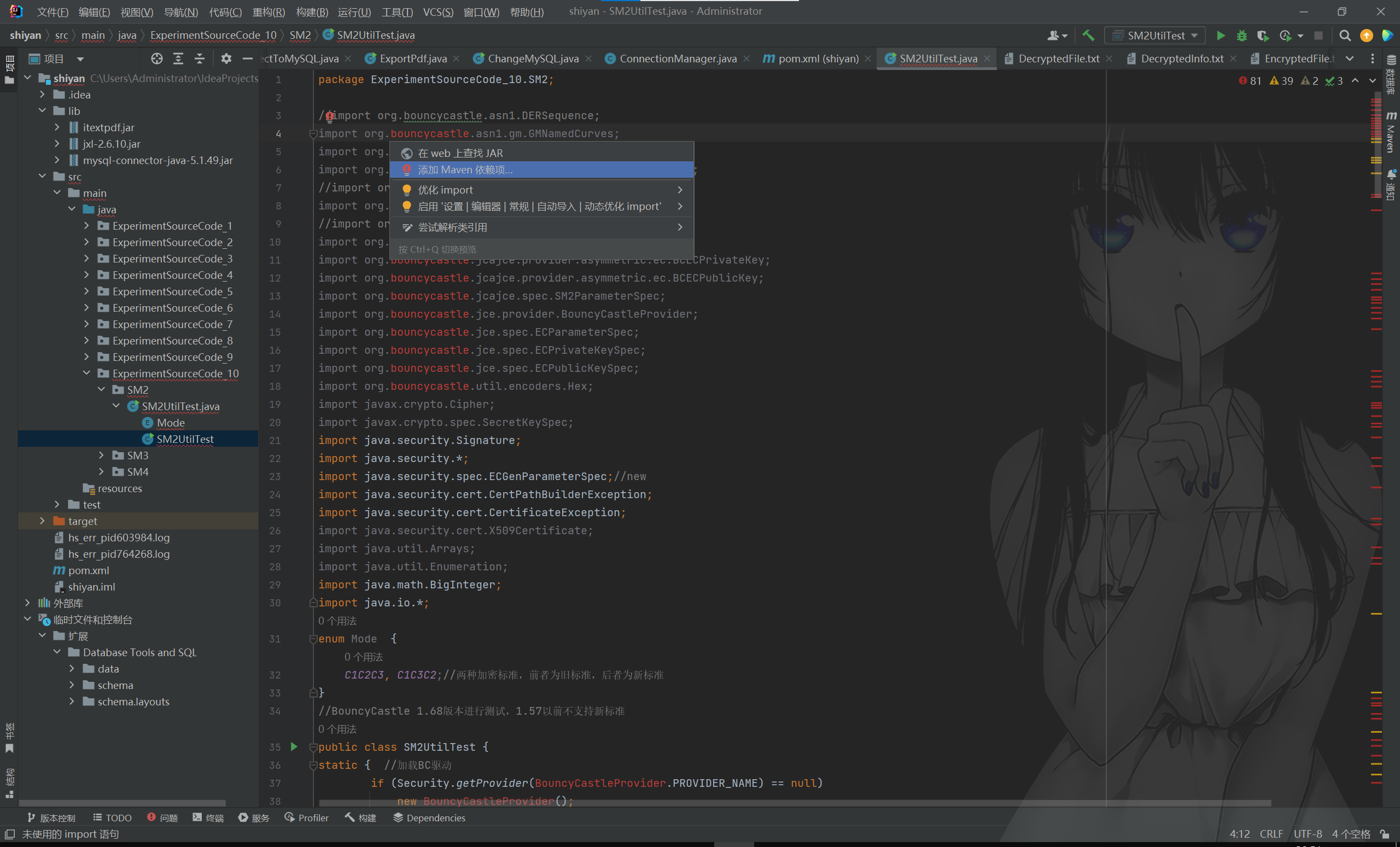
学号：320210942931

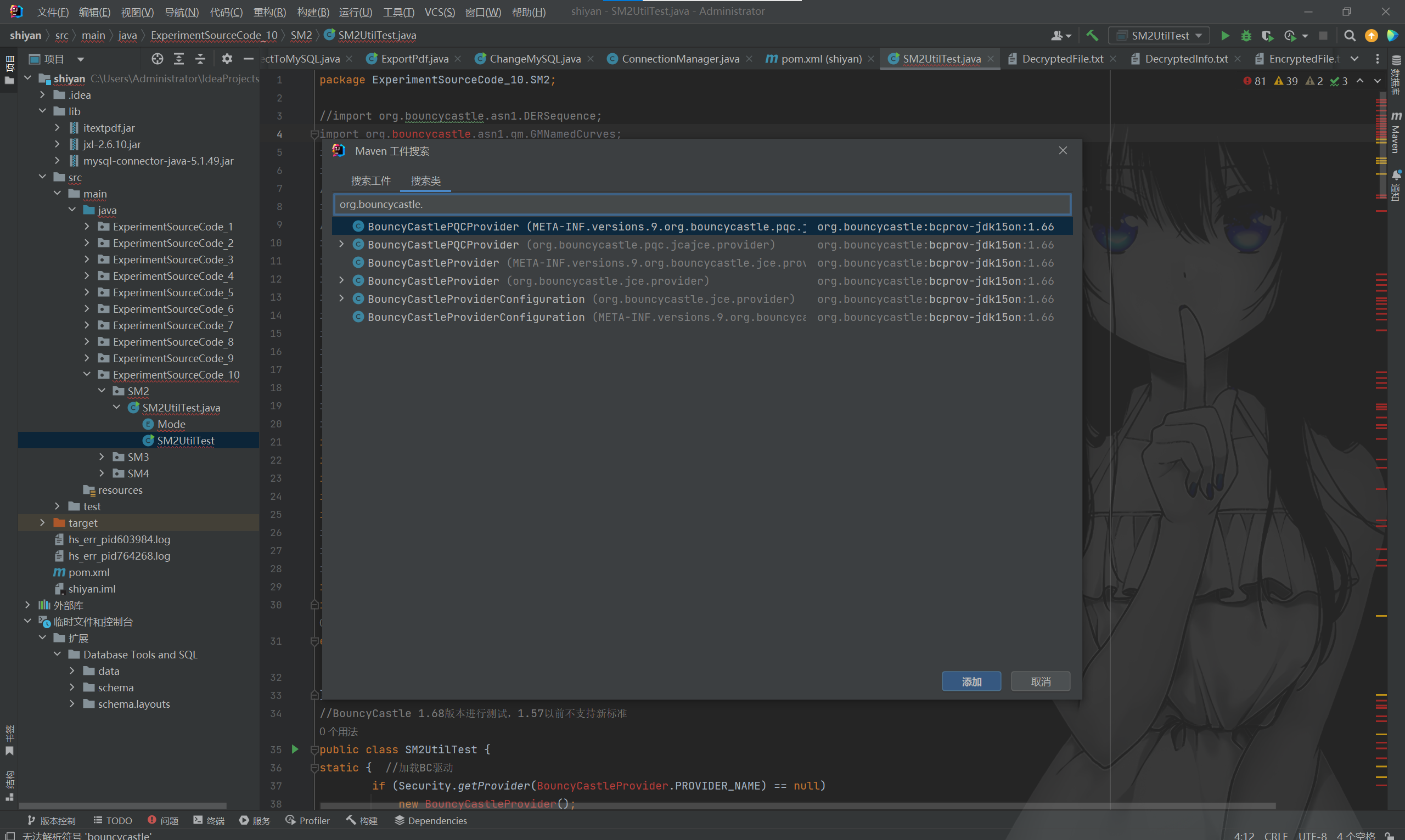
班级：2021级计算机科学与技术二班

一：商用密码算法相关依赖导入（maven）

1. 因为在idea上配置好了MAVEN项目，所以相关缺失的包可以直接通过

pom.xml文件导入。





2.pom.xml文件配置如下：



二：用SM4算法实现对一个Java源程序文件的加解密操作。

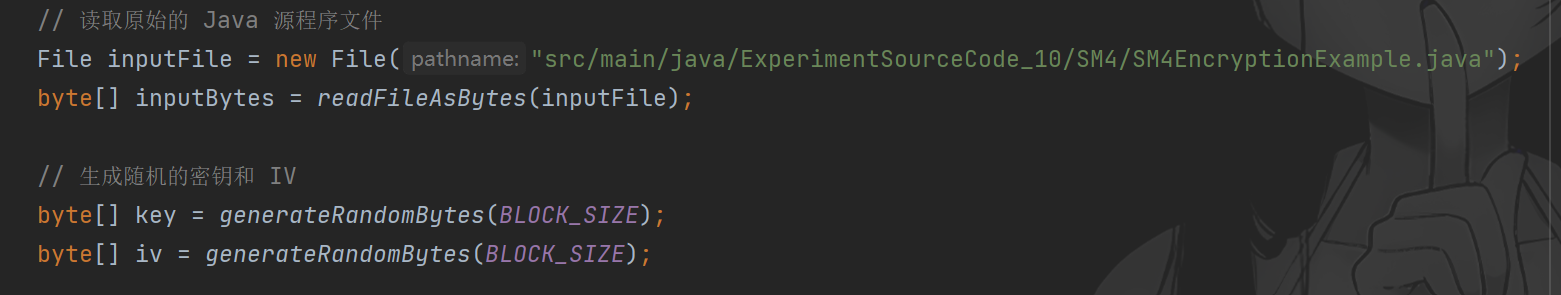
1. 思路分析

其实就是调用SM4算法的一个加密再解密的过程，对于加密玩的密钥、密文、IV、内容写入文件，对于解密后的内容、密钥、IV也写入文件。

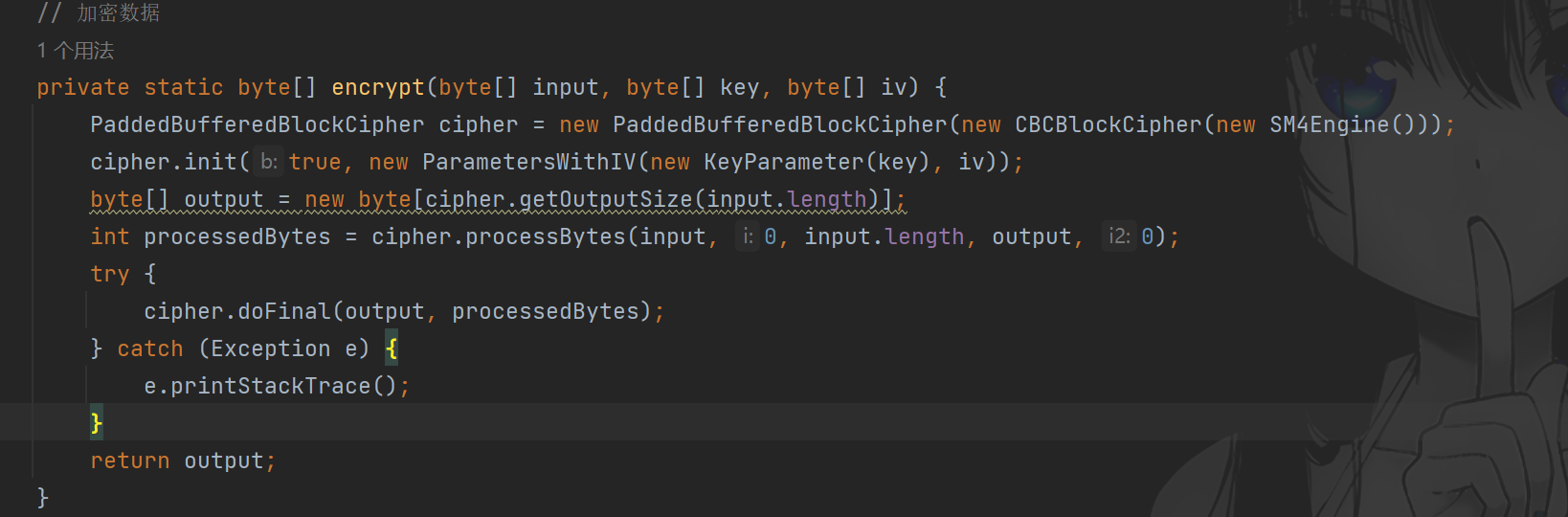
* 加密：读取Java源文件，转化为字节数组，生成随机密钥和初始化向量，采用encrypt方法对输入字节数组进行加密；
* 解密：采用decrypt方法对加密后的字节数组进行解密，得到解密后的字节数组

1. 具体实现：

* 读取源文件，生成随机密钥和IV



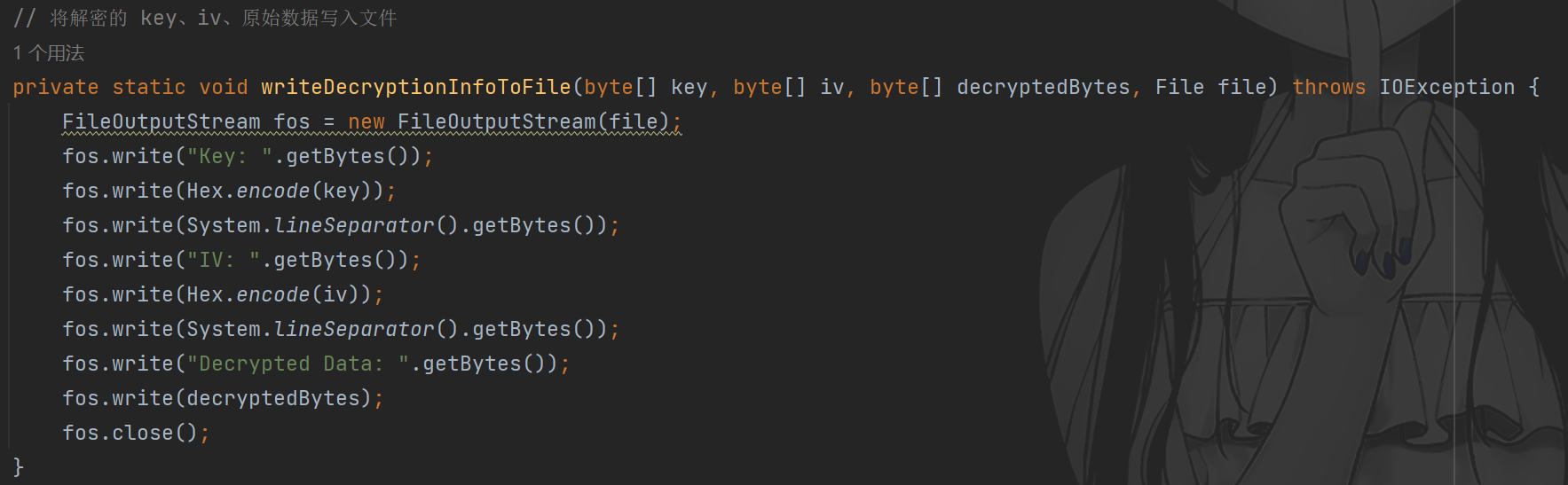
* 加密



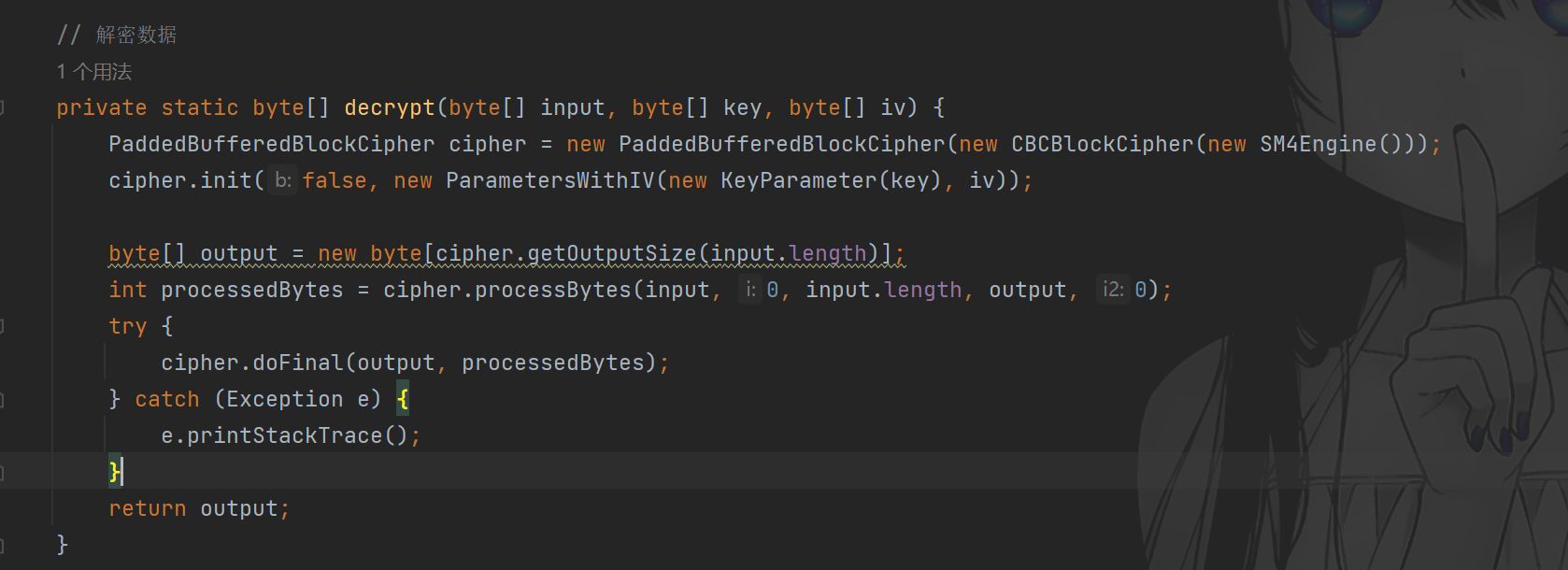
* 加密后的密文、数据、密钥、IV写入文件



* 解密后的数据、密钥、IV写入文件



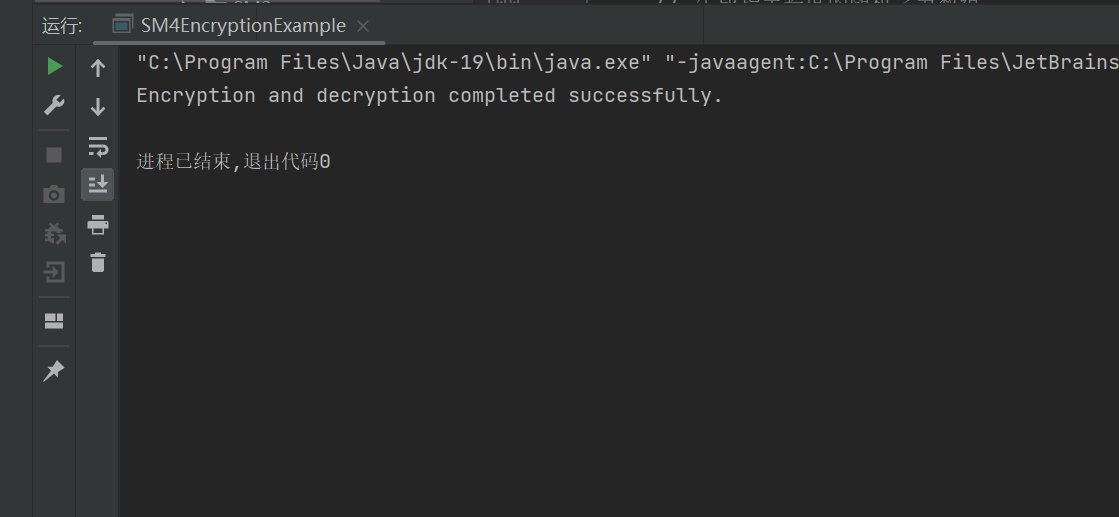
* 解密：



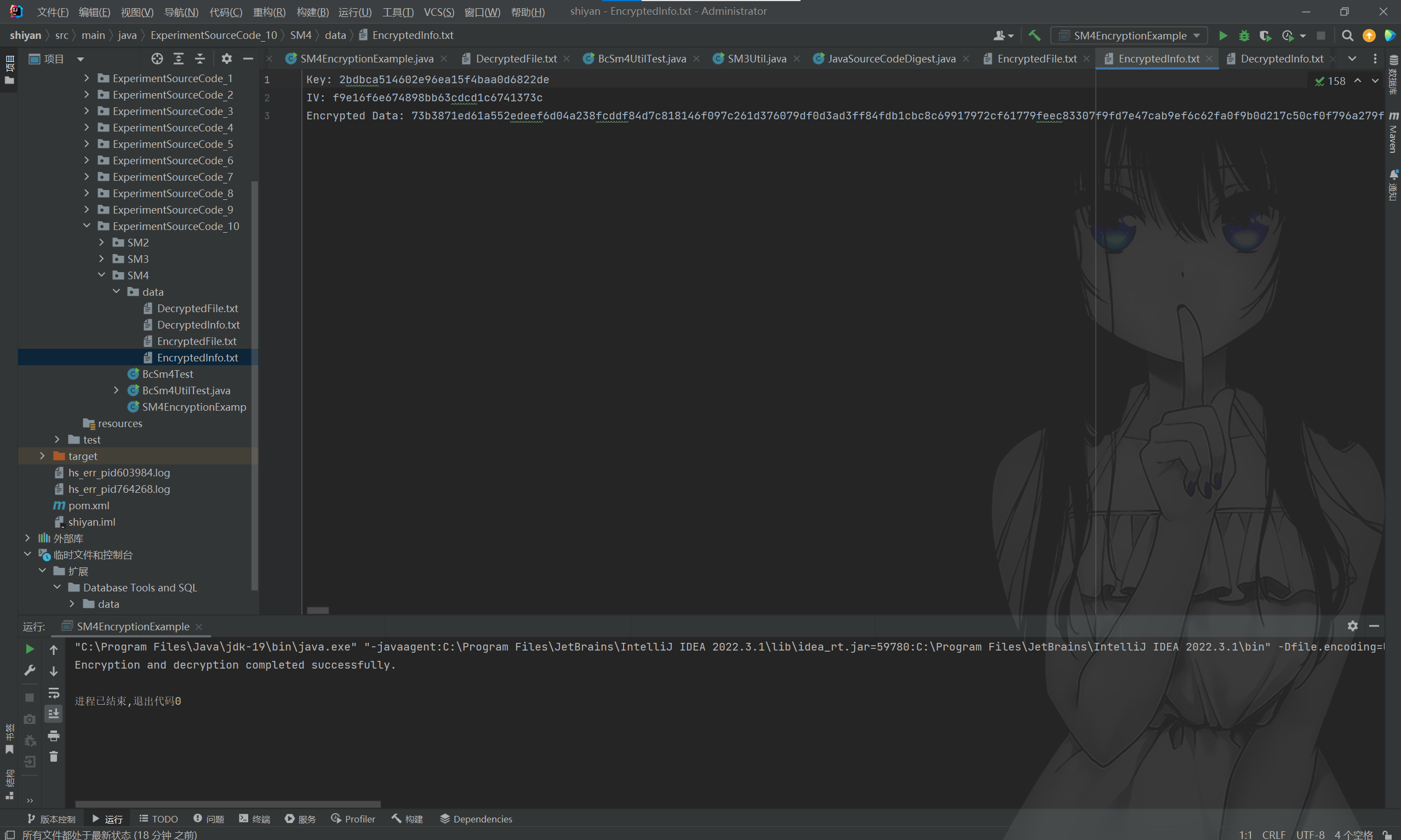
1. 源代码：

|  |
| --- |
| package ExperimentSourceCode\_10**.**SM4**;**  **import** java**.**io**.\*;**  **import** org**.**bouncycastle**.**crypto**.**engines**.**SM4Engine**;**  **import** org**.**bouncycastle**.**crypto**.**modes**.**CBCBlockCipher**;**  **import** org**.**bouncycastle**.**crypto**.**paddings**.**PaddedBufferedBlockCipher**;**  **import** org**.**bouncycastle**.**crypto**.**params**.**KeyParameter**;**  **import** org**.**bouncycastle**.**crypto**.**params**.**ParametersWithIV**;**  **import** org**.**bouncycastle**.**util**.**encoders**.**Hex**;**  public class SM4EncryptionExample **{**  private static final int BLOCK\_SIZE **=** 16**;** // SM4 分组大小为 16 字节  public static void main**(**String**[]** args**)** **{**  **try** **{**  // 读取原始的 Java 源程序文件  File inputFile **=** **new** File**(**"src/main/java/ExperimentSourceCode\_10/SM4/SM4EncryptionExample.java"**);**  byte**[]** inputBytes **=** readFileAsBytes**(**inputFile**);**  // 生成随机的密钥和 IV  byte**[]** key **=** generateRandomBytes**(**BLOCK\_SIZE**);**  byte**[]** iv **=** generateRandomBytes**(**BLOCK\_SIZE**);**  // 加密文件内容  byte**[]** encryptedBytes **=** encrypt**(**inputBytes**,** key**,** iv**);**  // 将加密后的内容写入文件  File encryptedFile **=** **new** File**(**"src/main/java/ExperimentSourceCode\_10/SM4/data/EncryptedFile.txt"**);**  writeBytesToFile**(**encryptedBytes**,** encryptedFile**);**  // 将加密后的 key、iv、密文写入文件  File encryptedInfoFile **=** **new** File**(**"src/main/java/ExperimentSourceCode\_10/SM4/data/EncryptedInfo.txt"**);**  writeEncryptionInfoToFile**(**key**,** iv**,** encryptedBytes**,** encryptedInfoFile**);**  // 解密文件内容  byte**[]** decryptedBytes **=** decrypt**(**encryptedBytes**,** key**,** iv**);**  // 将解密后的内容写入文件  File decryptedFile **=** **new** File**(**"src/main/java/ExperimentSourceCode\_10/SM4/data/DecryptedFile.txt"**);**  writeBytesToFile**(**decryptedBytes**,** decryptedFile**);**  // 将解密后的 key、iv、原始数据写入文件  File decryptedInfoFile **=** **new** File**(**"src/main/java/ExperimentSourceCode\_10/SM4/data/DecryptedInfo.txt"**);**  writeDecryptionInfoToFile**(**key**,** iv**,** inputBytes**,** decryptedInfoFile**);**  System**.**out**.**println**(**"Encryption and decryption completed successfully."**);**  **}** **catch** **(**IOException e**)** **{**  e**.**printStackTrace**();**  **}**  **}**  // 读取文件内容为字节数组  private static byte**[]** readFileAsBytes**(**File file**)** **throws** IOException **{**  ByteArrayOutputStream bos **=** **new** ByteArrayOutputStream**();**  FileInputStream fis **=** **new** FileInputStream**(**file**);**  byte**[]** buffer **=** **new** byte**[**1024**];**  int length**;**  **while** **((**length **=** fis**.**read**(**buffer**))** **!=** **-**1**)** **{**  bos**.**write**(**buffer**,** 0**,** length**);**  **}**  fis**.**close**();**  **return** bos**.**toByteArray**();**  **}**  // 将字节数组写入文件  private static void writeBytesToFile**(**byte**[]** bytes**,** File file**)** **throws** IOException **{**  FileOutputStream fos **=** **new** FileOutputStream**(**file**);**  fos**.**write**(**bytes**);**  fos**.**close**();**  **}**  // 将加密的 key、iv、密文写入文件  private static void writeEncryptionInfoToFile**(**byte**[]** key**,** byte**[]** iv**,** byte**[]** encryptedBytes**,** File file**)** **throws** IOException **{**  FileOutputStream fos **=** **new** FileOutputStream**(**file**);**  fos**.**write**(**"Key: "**.**getBytes**());**  fos**.**write**(**Hex**.**encode**(**key**));**  fos**.**write**(**System**.**lineSeparator**().**getBytes**());**  fos**.**write**(**"IV: "**.**getBytes**());**  fos**.**write**(**Hex**.**encode**(**iv**));**  fos**.**write**(**System**.**lineSeparator**().**getBytes**());**  fos**.**write**(**"Encrypted Data: "**.**getBytes**());**  fos**.**write**(**Hex**.**encode**(**encryptedBytes**));**  fos**.**close**();**  **}**  // 将解密的 key、iv、原始数据写入文件  private static void writeDecryptionInfoToFile**(**byte**[]** key**,** byte**[]** iv**,** byte**[]** decryptedBytes**,** File file**)** **throws** IOException **{**  FileOutputStream fos **=** **new** FileOutputStream**(**file**);**  fos**.**write**(**"Key: "**.**getBytes**());**  fos**.**write**(**Hex**.**encode**(**key**));**  fos**.**write**(**System**.**lineSeparator**().**getBytes**());**  fos**.**write**(**"IV: "**.**getBytes**());**  fos**.**write**(**Hex**.**encode**(**iv**));**  fos**.**write**(**System**.**lineSeparator**().**getBytes**());**  fos**.**write**(**"Decrypted Data: "**.**getBytes**());**  fos**.**write**(**decryptedBytes**);**  fos**.**close**();**  **}**  // 生成指定长度的随机字节数组  private static byte**[]** generateRandomBytes**(**int length**)** **{**  byte**[]** randomBytes **=** **new** byte**[**length**];**  **new** java**.**security**.**SecureRandom**().**nextBytes**(**randomBytes**);**  **return** randomBytes**;**  **}**  // 加密数据  private static byte**[]** encrypt**(**byte**[]** input**,** byte**[]** key**,** byte**[]** iv**)** **{**  PaddedBufferedBlockCipher cipher **=** **new** PaddedBufferedBlockCipher**(new** CBCBlockCipher**(new** SM4Engine**()));**  cipher**.**init**(true,** **new** ParametersWithIV**(new** KeyParameter**(**key**),** iv**));**  byte**[]** output **=** **new** byte**[**cipher**.**getOutputSize**(**input**.**length**)];**  int processedBytes **=** cipher**.**processBytes**(**input**,** 0**,** input**.**length**,** output**,** 0**);**  **try** **{**  cipher**.**doFinal**(**output**,** processedBytes**);**  **}** **catch** **(**Exception e**)** **{**  e**.**printStackTrace**();**  **}**  **return** output**;**  **}**  // 解密数据  private static byte**[]** decrypt**(**byte**[]** input**,** byte**[]** key**,** byte**[]** iv**)** **{**  PaddedBufferedBlockCipher cipher **=** **new** PaddedBufferedBlockCipher**(new** CBCBlockCipher**(new** SM4Engine**()));**  cipher**.**init**(false,** **new** ParametersWithIV**(new** KeyParameter**(**key**),** iv**));**  byte**[]** output **=** **new** byte**[**cipher**.**getOutputSize**(**input**.**length**)];**  int processedBytes **=** cipher**.**processBytes**(**input**,** 0**,** input**.**length**,** output**,** 0**);**  **try** **{**  cipher**.**doFinal**(**output**,** processedBytes**);**  **}** **catch** **(**Exception e**)** **{**  e**.**printStackTrace**();**  **}**  **return** output**;**  **}**  **}** |

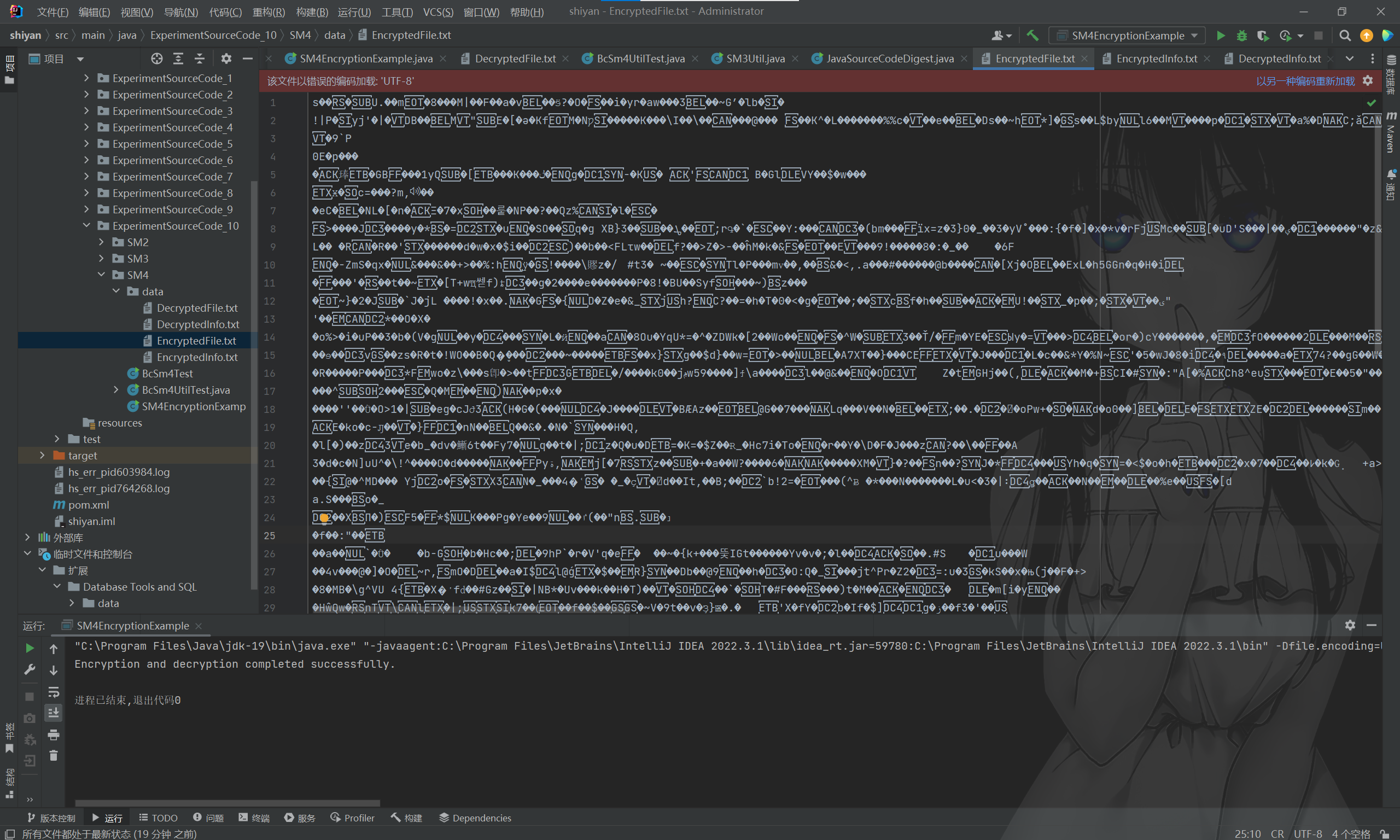
1. 运行结果：



* 加密后的密钥、IV和密文



* 加密后的内容（可见加密后是乱码，加密成功）



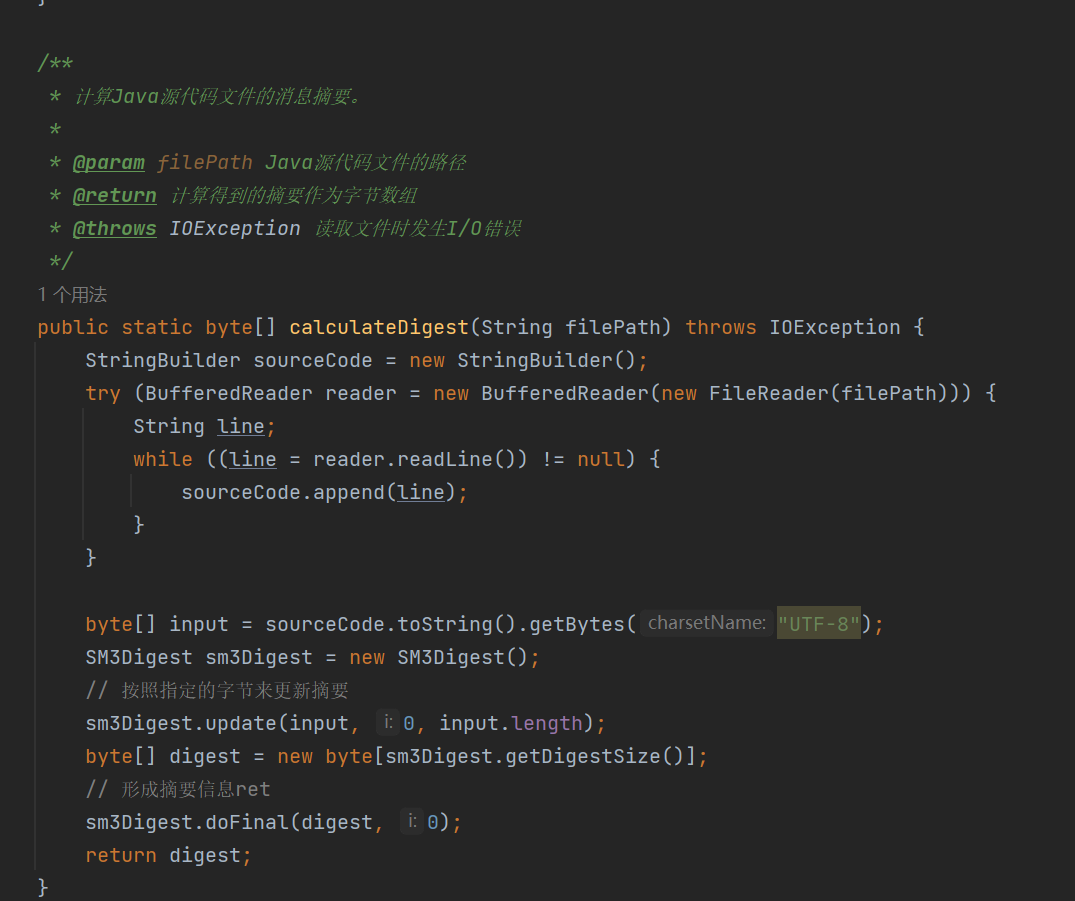
* 解密后的数据、密钥和IV：



三． 用SM3算法计算一个Java源程序的消息摘要。

1.思路分析及具体过程：

* 采用SM3算法计算Java源程序的消息摘要：



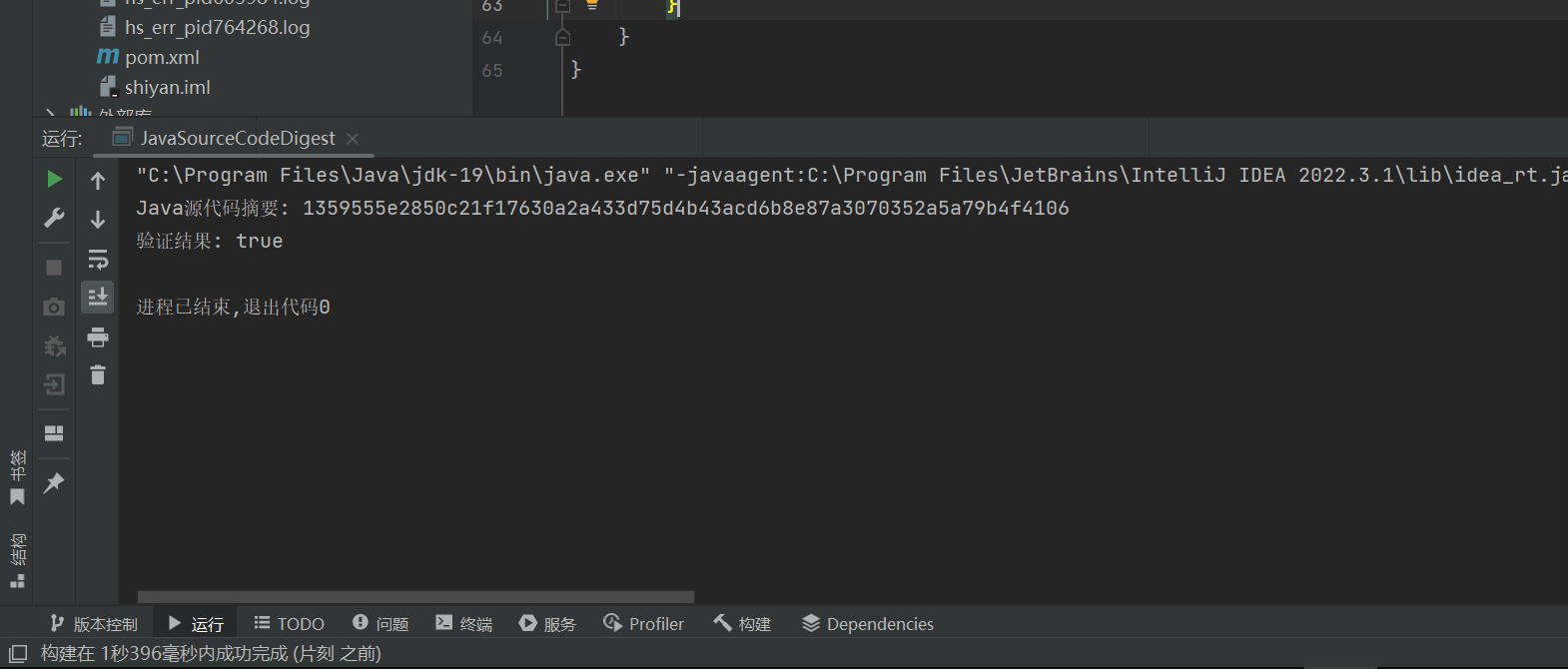
* 调用方法并验证消息摘要是否正确



1. 源代码：

|  |
| --- |
| package ExperimentSourceCode\_10**.**SM3**;**  **import** java**.**io**.**BufferedReader**;**  **import** java**.**io**.**FileReader**;**  **import** java**.**io**.**IOException**;**  **import** java**.**security**.**Security**;**  **import** org**.**bouncycastle**.**jce**.**provider**.**BouncyCastleProvider**;**  **import** org**.**bouncycastle**.**crypto**.**digests**.**SM3Digest**;**  **import** org**.**bouncycastle**.**util**.**encoders**.**Hex**;**  /\*\*  \* 该类使用SM3算法计算Java源代码的消息摘要。  \*/  public class JavaSourceCodeDigest **{**  static **{**  // 如果尚未添加Bouncy Castle提供者，则添加  **if** **(**Security**.**getProvider**(**BouncyCastleProvider**.**PROVIDER\_NAME**)** **==** **null)** **{**  Security**.**addProvider**(new** BouncyCastleProvider**());**  **}**  **}**  /\*\*  \* 计算Java源代码文件的消息摘要。  \*  \* **@param** filePath Java源代码文件的路径  \* **@return** 计算得到的摘要作为字节数组  \* **@throws** IOException 读取文件时发生I/O错误  \*/  public static byte**[]** calculateDigest**(**String filePath**)** **throws** IOException **{**  StringBuilder sourceCode **=** **new** StringBuilder**();**  **try** **(**BufferedReader reader **=** **new** BufferedReader**(new** FileReader**(**filePath**)))** **{**  String line**;**  **while** **((**line **=** reader**.**readLine**())** **!=** **null)** **{**  sourceCode**.**append**(**line**);**  **}**  **}**  byte**[]** input **=** sourceCode**.**toString**().**getBytes**(**"UTF-8"**);**  SM3Digest sm3Digest **=** **new** SM3Digest**();**  // 按照指定的字节来更新摘要  sm3Digest**.**update**(**input**,** 0**,** input**.**length**);**  byte**[]** digest **=** **new** byte**[**sm3Digest**.**getDigestSize**()];**  // 形成摘要信息ret  sm3Digest**.**doFinal**(**digest**,** 0**);**  **return** digest**;**  **}**  /\*\*  \* 主方法，用于计算Java源代码文件的摘要并进行验证。  \*  \* **@param** args 命令行参数（未使用）  \*/  public static void main**(**String**[]** args**)** **{**  **try** **{**  String filePath **=** "src/main/java/ExperimentSourceCode\_10/SM3/SM3Util.java"**;**  String expectedDigest **=** "1359555e2850c21f17630a2a433d75d4b43acd6b8e87a3070352a5a79b4f4106"**;**  byte**[]** digest **=** calculateDigest**(**filePath**);**  System**.**out**.**println**(**"Java源代码摘要: " **+** Hex**.**toHexString**(**digest**));**  // 验证计算得到的摘要是否与预期的摘要值相同  System**.**out**.**println**(**"验证结果: " **+** Hex**.**toHexString**(**digest**).**equals**(**expectedDigest**));**  **}** **catch** **(**IOException e**)** **{**  e**.**printStackTrace**();**  **}**  **}**  **}** |

1. 运行结果：

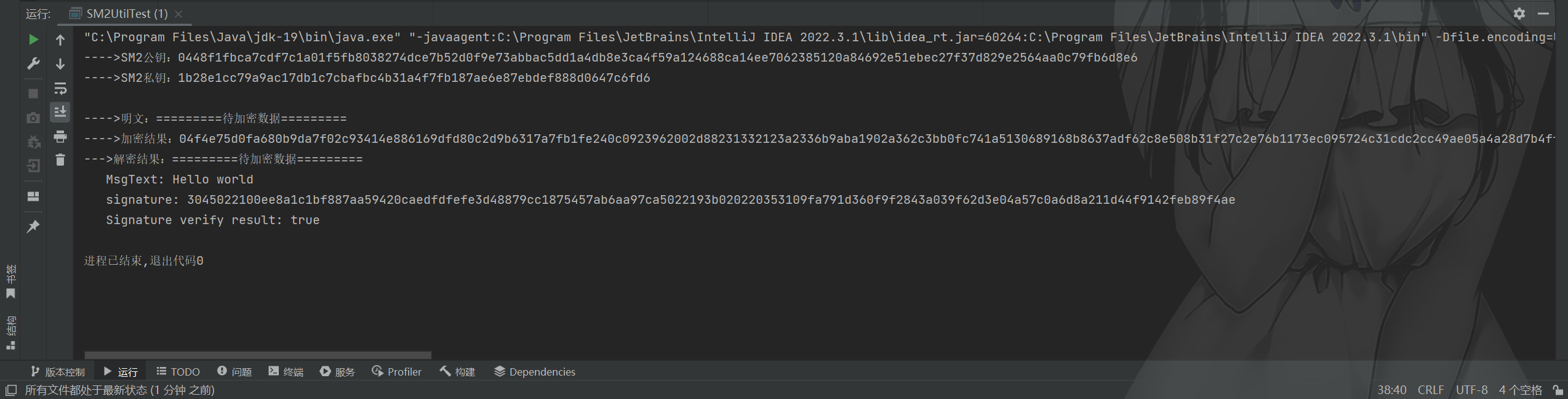


四：SM2算法的程序应用分析。分析理解示例程序，编译运行后回答问题

1. 源代码：

|  |
| --- |
| package ExperimentSourceCode\_10**.**SM2**;**  //import org.bouncycastle.asn1.DERSequence;  **import** org**.**bouncycastle**.**asn1**.**gm**.**GMNamedCurves**;**  **import** org**.**bouncycastle**.**asn1**.**x9**.**X9ECParameters**;**  **import** org**.**bouncycastle**.**crypto**.**InvalidCipherTextException**;**  //import org.bouncycastle.crypto.digests.SM3Digest;  **import** org**.**bouncycastle**.**crypto**.**engines**.**SM2Engine**;**  //import org.bouncycastle.crypto.engines.SM4Engine;  **import** org**.**bouncycastle**.**crypto**.**params**.\*;**  **import** org**.**bouncycastle**.**jcajce**.**provider**.**asymmetric**.**ec**.**BCECPrivateKey**;**  **import** org**.**bouncycastle**.**jcajce**.**provider**.**asymmetric**.**ec**.**BCECPublicKey**;**  **import** org**.**bouncycastle**.**jcajce**.**spec**.**SM2ParameterSpec**;**  **import** org**.**bouncycastle**.**jce**.**provider**.**BouncyCastleProvider**;**  **import** org**.**bouncycastle**.**jce**.**spec**.**ECParameterSpec**;**  **import** org**.**bouncycastle**.**jce**.**spec**.**ECPrivateKeySpec**;**  **import** org**.**bouncycastle**.**jce**.**spec**.**ECPublicKeySpec**;**  **import** org**.**bouncycastle**.**util**.**encoders**.**Hex**;**  **import** javax**.**crypto**.**Cipher**;**  **import** javax**.**crypto**.**spec**.**SecretKeySpec**;**  **import** java**.**security**.**Signature**;**  **import** java**.**security**.\*;**  **import** java**.**security**.**spec**.**ECGenParameterSpec**;**//new  **import** java**.**security**.**cert**.**CertPathBuilderException**;**  **import** java**.**security**.**cert**.**CertificateException**;**  **import** java**.**security**.**cert**.**X509Certificate**;**  **import** java**.**util**.**Arrays**;**  **import** java**.**util**.**Enumeration**;**  **import** java**.**math**.**BigInteger**;**  **import** java**.**io**.\*;**  enum Mode **{**  C1C2C3**,** C1C3C2**;**//两种加密标准，前者为旧标准，后者为新标准  **}**  //BouncyCastle 1.68版本进行测试，1.57以前不支持新标准  public class SM2UtilTest **{**  static **{** //加载BC驱动  **if** **(**Security**.**getProvider**(**BouncyCastleProvider**.**PROVIDER\_NAME**)** **==** **null)**  **new** BouncyCastleProvider**();**  **}**  // 生成sm2密钥对  static KeyPair createECKeyPair**()** **{**  Security**.**addProvider**(new** BouncyCastleProvider**());** //需要添加，否则会报错  //使用标准名称创建EC参数生成的参数规范  final ECGenParameterSpec sm2Spec **=** **new** ECGenParameterSpec**(**"sm2p256v1"**);**  // 获取一个椭圆曲线类型的密钥对生成器  final KeyPairGenerator kpg**;**  **try** **{**  kpg **=** KeyPairGenerator**.**getInstance**(**"EC"**,** "BC"**);**  // 使用SM2的算法域参数集和指定的随机源初始化密钥生成器  kpg**.**initialize**(**sm2Spec**,** **new** SecureRandom**());**  // 通过密钥生成器生成密钥对  **return** kpg**.**generateKeyPair**();**  **}** **catch** **(**Exception e**)** **{**  e**.**printStackTrace**();**  **return** **null;**  **}**  **}**  /\*  \*公钥加密  \* @param publicKey SM2公钥  \* @param data 明文数据  \* @param modeType 加密模式  \* @return 密文数据  \*/  public static byte**[]** encrypt**(**BCECPublicKey publicKey**,** byte**[]** inputBytes**,** int modeType**)** **{**  //加密模式  SM2Engine**.**Mode mode**;**  **if** **(**modeType **==** 1**)** **{**//采用新模式加密标准  mode **=** SM2Engine**.**Mode**.**C1C3C2**;**  **}** **else** **{**//采用旧模式加密标准  mode **=** SM2Engine**.**Mode**.**C1C2C3**;**  **}**  //通过公钥对象获取公钥的基本域参数。  ECParameterSpec ecParameterSpec **=** publicKey**.**getParameters**();**  ECDomainParameters ecDomainParameters **=** **new** ECDomainParameters**(**ecParameterSpec**.**getCurve**(),**  ecParameterSpec**.**getG**(),** ecParameterSpec**.**getN**());**  //通过公钥值和公钥基本参数创建公钥参数对象  ECPublicKeyParameters ecPublicKeyParameters **=** **new** ECPublicKeyParameters**(**publicKey**.**getQ**(),** ecDomainParameters**);**  //根据加密模式实例化SM2公钥加密引擎  SM2Engine sm2Engine **=** **new** SM2Engine**(**mode**);**  //初始化加密引擎  sm2Engine**.**init**(true,** **new** ParametersWithRandom**(**ecPublicKeyParameters**,** **new** SecureRandom**()));**  byte**[]** arrayOfBytes **=** **null;**  **try** **{**  //通过加密引擎对字节数串行加密  arrayOfBytes **=** sm2Engine**.**processBlock**(**inputBytes**,** 0**,** inputBytes**.**length**);**  **}** **catch** **(**Exception e**)** **{**  System**.**out**.**println**(**"SM2加密时出现异常:" **+** e**.**getMessage**());**  e**.**printStackTrace**();**  **}**  **return** arrayOfBytes**;**  **}**  /\*\*  \* 私钥解密  \*  \* **@param** privateKey SM私钥  \* **@param** cipherBytes 密文数据  \* **@param** modeType 加密模式  \* **@return** 解密后的明文  \*/  public static byte**[]** decrypt**(**BCECPrivateKey privateKey**,** byte**[]** cipherBytes**,** int modeType**)** **{**  //解密模式  SM2Engine**.**Mode mode**;**  **if** **(**modeType **==** 1**)** **{**  mode **=** SM2Engine**.**Mode**.**C1C3C2**;**  **}** **else** **{**  mode **=** SM2Engine**.**Mode**.**C1C2C3**;**  **}**  //将十六进制字符串密文转换为字节数组（需要与加密一致，加密是：加密后的字节数组转换为了十六进制字符串）  //byte[] cipherDataByte = Hex.decode(cipherData);  //通过私钥对象获取私钥的基本域参数。  ECParameterSpec ecParameterSpec **=** privateKey**.**getParameters**();**  ECDomainParameters ecDomainParameters **=** **new** ECDomainParameters**(**ecParameterSpec**.**getCurve**(),**  ecParameterSpec**.**getG**(),** ecParameterSpec**.**getN**());**  //通过私钥值和私钥基本参数创建私钥参数对象  ECPrivateKeyParameters ecPrivateKeyParameters **=** **new** ECPrivateKeyParameters**(**privateKey**.**getD**(),**  ecDomainParameters**);**  //通过解密模式创建解密引擎并初始化  SM2Engine sm2Engine **=** **new** SM2Engine**(**SM2Engine**.**Mode**.**C1C3C2**);**  sm2Engine**.**init**(false,** ecPrivateKeyParameters**);**  byte**[]** arrayOfBytes **=** **null;**  **try** **{**  //通过解密引擎对密文字节串进行解密  arrayOfBytes **=** sm2Engine**.**processBlock**(**cipherBytes**,** 0**,** cipherBytes**.**length**);**  **}** **catch** **(**Exception e**)** **{**  System**.**out**.**println**(**"SM2解密时出现异常" **+** e**.**getMessage**());**  **}**  **return** arrayOfBytes**;**  **}**  //椭圆曲线ECParameters ASN.1 结构  private static X9ECParameters x9ECParameters **=** GMNamedCurves**.**getByName**(**"sm2p256v1"**);**  //椭圆曲线公钥或私钥的基本域参数。  private static ECParameterSpec ecDomainParameters **=** **new** ECParameterSpec**(**x9ECParameters**.**getCurve**(),** x9ECParameters**.**getG**(),** x9ECParameters**.**getN**());**  /\*\*  \* 公钥字符串转换为 BCECPublicKey 公钥对象  \*  \* **@param** pubKeyHex 64字节十六进制公钥字符串(如果公钥字符串为65字节首个字节为0x04：表示该公钥为非压缩格式，操作时需要删除)  \* **@return** BCECPublicKey SM2公钥对象  \*/  public static BCECPublicKey getECPublicKeyByPublicKeyHex**(**String pubKeyHex**)** **{**  //截取64字节有效的SM2公钥（如果公钥首个字节为0x04）  **if** **(**pubKeyHex**.**length**()** **>** 128**)** **{**  pubKeyHex **=** pubKeyHex**.**substring**(**pubKeyHex**.**length**()** **-** 128**);**  **}**  //将公钥拆分为x,y分量（各32字节）  String stringX **=** pubKeyHex**.**substring**(**0**,** 64**);**  String stringY **=** pubKeyHex**.**substring**(**stringX**.**length**());**  //将公钥x、y分量转换为BigInteger类型  BigInteger x **=** **new** BigInteger**(**stringX**,** 16**);**  BigInteger y **=** **new** BigInteger**(**stringY**,** 16**);**  //通过公钥x、y分量创建椭圆曲线公钥规范  ECPublicKeySpec ecPublicKeySpec **=** **new** ECPublicKeySpec**(**x9ECParameters**.**getCurve**().**createPoint**(**x**,** y**),** ecDomainParameters**);**  //通过椭圆曲线公钥规范，创建出椭圆曲线公钥对象（可用于SM2加密及验签）  **return** **new** BCECPublicKey**(**"EC"**,** ecPublicKeySpec**,** BouncyCastleProvider**.**CONFIGURATION**);**  **}**  /\*\*  \* 私钥字符串转换为 BCECPrivateKey 私钥对象  \*  \* **@param** privateKeyHex: 32字节十六进制私钥字符串  \* **@return** BCECPrivateKey:SM2私钥对象  \*/  public static BCECPrivateKey getBCECPrivateKeyByPrivateKeyHex**(**String privateKeyHex**)** **{**  //将十六进制私钥字符串转换为BigInteger对象  BigInteger d **=** **new** BigInteger**(**privateKeyHex**,** 16**);**  //通过私钥和私钥域参数集创建椭圆曲线私钥规范  ECPrivateKeySpec ecPrivateKeySpec **=** **new** ECPrivateKeySpec**(**d**,** ecDomainParameters**);**  //通过椭圆曲线私钥规范，创建出椭圆曲线私钥对象（可用于SM2解密和签名）  **return** **new** BCECPrivateKey**(**"EC"**,** ecPrivateKeySpec**,** BouncyCastleProvider**.**CONFIGURATION**);**  **}**  /\*\*  \* 明文签证  \*  \* **@param** msg: 32字节十六进制明数组  \* **@param** privateKey:SM2私钥对象  \* **@return** 十六进制字符数组签名  \*/  public static byte**[]** Sm2Sign**(**byte**[]** msg**,**PrivateKey privateKey**)throws** IOException**,** NoSuchAlgorithmException**,** NoSuchProviderException**,** InvalidAlgorithmParameterException**,**  InvalidKeyException**,**CertPathBuilderException**,** SignatureException**,** CertificateException **{**  // 生成SM2sign with sm3 签名验签算法实例  Signature signature **=** Signature**.**getInstance**(**"SM3withSm2"**,** "BC"**);**  // 签名 签名需要使用私钥初始化签名实例  signature**.**initSign**(**privateKey**);**  // 签名原文  signature**.**update**(**msg**);**  // 计算签名值，作为返回值  byte**[]** signatureValue **=** signature**.**sign**();**  **return** signatureValue**;**  **}**  /\*\*  \* 签名验证  \*  \* **@param** msg: 十六进制明文字符数组  \* **@param** fingerprint: 提供的十六进制字节数组签名  \* **@param** publicKey:SM2公钥对象  \* **@return** 签名是否正确  \*/  public static boolean Sm2Verify**(**byte**[]** msg**,**byte**[]** fingerprint**,**PublicKey publicKey**)throws** IOException**,**  NoSuchAlgorithmException**,**CertPathBuilderException**,** NoSuchProviderException**,** InvalidAlgorithmParameterException**,**  InvalidKeyException**,** SignatureException **{**  // 生成SM2sign with sm3 签名验签算法实例  Signature signature **=** Signature**.**getInstance**(**"SM3withSm2"**,** "BC"**);**  //验签 签名需要使用公钥初始化签名实例  signature**.**initVerify**(**publicKey**);**  // 写入待验签的签名原文  signature**.**update**(**msg**);**  // 验签  **return** signature**.**verify**(**fingerprint**);**  **}**  public static void main**(**String**[]** args**)** **throws** IOException**,** NoSuchAlgorithmException**,** NoSuchProviderException**,**  InvalidAlgorithmParameterException**,** CertPathBuilderException**,**InvalidKeyException**,** SignatureException**,** CertificateException **{**  String publicKeyHex **=** **null;**  String privateKeyHex **=** **null;**  //产生公钥、私钥密钥对  KeyPair keyPair **=** createECKeyPair**();**  PrivateKey privateKey **=** keyPair**.**getPrivate**();**  PublicKey publicKey **=** keyPair**.**getPublic**();**  **if** **(**publicKey **instanceof** BCECPublicKey**)** **{**  //获取65字节非压缩的十六进制公钥串(0x04)  byte**[]** publicKeyBytes **=** **((**BCECPublicKey**)** publicKey**).**getQ**().**getEncoded**(false);**  publicKeyHex **=** Hex**.**toHexString**(**publicKeyBytes**);**  System**.**out**.**println**(**"---->SM2公钥：" **+** publicKeyHex**);**  **}**  **if** **(**privateKey **instanceof** BCECPrivateKey**)** **{**  //获取32字节十六进制私钥串  privateKeyHex **=** **((**BCECPrivateKey**)** privateKey**).**getD**().**toString**(**16**);**  System**.**out**.**println**(**"---->SM2私钥：" **+** privateKeyHex**);**  **}**  // 公钥加密  String plainText **=** "=========待加密数据========="**;**  byte**[]** plainBytes **=** plainText**.**getBytes**(**"utf-8"**);**  System**.**out**.**println**(**"\n---->明文：" **+** plainText**);**  //将十六进制公钥串转换为 BCECPublicKey 公钥对象  BCECPublicKey bcecPublicKey **=** getECPublicKeyByPublicKeyHex**(**publicKeyHex**);**  byte**[]** encryptBytes **=** encrypt**(**bcecPublicKey**,** plainBytes**,** 1**);**  String encryptData **=** Hex**.**toHexString**(**encryptBytes**);**  System**.**out**.**println**(**"---->加密结果：" **+** encryptData**);**  // 私钥解密  //将十六进制私钥串转换为 BCECPrivateKey 私钥对象  BCECPrivateKey bcecPrivateKey **=** getBCECPrivateKeyByPrivateKeyHex**(**privateKeyHex**);**  byte**[]** decryBytes **=** decrypt**(**bcecPrivateKey**,** encryptBytes**,** 1**);**  //将解密后的字节串转换为utf8字符编码的字符串（需要与明文加密时字符串转换成字节串所指定的字符编码保持一致）  String data **=** **new** String**(**decryBytes**,**"utf-8"**);**  System**.**out**.**println**(**"--->解密结果：" **+** data**);**  //签名与验证  String msgText **=** "Hello world"**;**  byte**[]** msgBytes **=** msgText**.**getBytes**(**"UTF-8"**);**  byte**[]** signatureValue **=** Sm2Sign**(**msgBytes**,**privateKey**);**  System**.**out**.**println**(**" MsgText: " **+** msgText**);**  System**.**out**.**println**(**" signature: " **+** Hex**.**toHexString**(**signatureValue**));**  System**.**out**.**println**(**" Signature verify result: " **+** Sm2Verify**(**msgBytes**,**signatureValue**,**publicKey**));**  **}**  **}** |

1. 运行结果



1. 程序理解分析：

这段程序利用BouncyCastle库提供的功能实现了SM2加密和解密的操作，程序中提供了一套完整的SM2加密和解密功能，以及与密钥生成、密钥转换和签名验证相关的辅助方法。它可以用于安全地进行数据的加密、解密和签名操作。它包括以下主要步骤和功能：

* 定义了加密模式的枚举类型。
* 创建了一个SM2UtilTest类，包含各种加密、解密和辅助方法。
* 在静态代码块中加载BouncyCastle的提供者。
* 实现了生成SM2密钥对的方法。
* 实现了公钥加密和私钥解密的方法。
* 实现了将公钥和私钥的十六进制字符串转换为相应对象的方法。
* 实现了明文签名和签名验证的方法。
* 在main方法中，生成了SM2密钥对，进行了公钥加密和私钥解密的操作，并输出结果。

1. 问题回答：

* 程序中SM2算法的密钥对是如何生成的?
* 在静态代码块中加载BouncyCastle提供者Security.addProvider(new BouncyCastleProvider())。这是为了确保BouncyCastle提供者已被正确加载，以便使用SM2算法。
* 在generateKeyPair()方法中，创建一个KeyPairGenerator对象，指定算法为"EC"（椭圆曲线）。
* 为KeyPairGenerator对象设置参数，使用ECNamedCurveTable.getParameterSpec("sm2p256v1")指定椭圆曲线参数为SM2的标准参数。
* 调用KeyPairGenerator对象的generateKeyPair()方法生成密钥对。
* 通过KeyPair对象的getPrivate()和getPublic()方法分别获取私钥和公钥。将私钥和公钥转换为十六进制字符串表示。

通过以上步骤，程序使用BouncyCastle库生成了SM2算法所需的密钥对。私钥用于解密和签名操作，公钥用于加密和验证签名操作

* SM2Engine类的init()、processBlock()方法的功能是什么?
* init()方法的作用是初始化SM2引擎，根据操作模式设置相应的内部状态和参数。在加密模式下，它会设置加密操作所需的公钥；在解密模式下，它会设置解密操作所需的私钥。
* processBlock()方法用于处理输入数据块并产生输出数据块。它接受输入数据块，并将其加密（或解密）处理后输出到指定的输出缓冲区。这个方法会根据当前的操作模式和设置的密钥对输入数据进行相应的处理。处理完成后，它返回处理的数据块长度，通常与输入数据块长度相同。
* Signature类的getInstance()、initSign()、update()、initVerify()、verify()方法的能是什么?
* getInstance()方法用于获取指定算法的Signature对象实例，以便后续的签名或验证操作。
* initSign()方法用于初始化签名对象，并设置私钥以便进行签名操作。
* update()方法用于向签名对象添加要签名的数据，可以多次调用以添加不同的数据块。
* initVerify()方法用于初始化签名对象，并设置公钥以便进行验证操作。
* verify()方法用于对之前更新的数据进行验证签名，传入签名数据的字节数组进行验证，并返回验证结果。