Case Study: Mật mã học với Java

Message Digest

Thông điệp có kích thước thay đổi được đưa qua hàm băm bảo mật để tạo ra một trị băm có kích thước cố định, đặc trưng cho thông điệp đầu vào. Trị băm này gọi là Message Digest (tóm tắt thông điệp). Các thuật toán băm bảo mật phổ biến là:

+ SHA - Secure Hash Algorithm, NIST FIPS 180-1.
+ MD2 - RFC 1319.
+ MD5 - RFC 1321.

Lớp MessageDigest

java.security.MessageDigest là lớp trừu tượng đại diện cho các lớp cài đặt các thuật toán băm bảo mật cụ thể dùng tạo Message Digest cho một thông điệp đầu vào. Một số phương thức chính:

getInstance()	Trả về đối tượng MessageDigest với thuật toán băm bảo mật chỉ định.
digest()	Trả về dữ liệu của MessageDigest, chính là kết quả của việc áp dụng thuật toán băm.
getAlgorithm()	Trả về tên của thuật toán băm được chỉ định.
getProvider()	Trả về nhà cung cấp đối tượng MessageDigest, mặc định là Sun.

Chương trình sau sẽ đọc tập tin input lần lượt vào một mảng byte, đối tượng MessageDigest dùng phương thức update() để cập nhật dữ liệu từ mảng byte này. Sau đó, phương thức digest() được gọi để sinh ra Message Digest từ dữ liệu chứa trong đối tượng MessageDigest, theo thuật toán băm bảo mật được chỉ định.

```
import java.io.*;
import java.security.*;
class JcaMessageDigest {
  public static void main( String[] args ) {
    if ( args.length < 3 ) {
      System.out.println( "Usage: java JcaMessageDigest input output algorithm" );
      return;
    }
    String input = args[0];
    String output = args[1];
    String algorithm = args[2];  // SHA, MD5
    try {
      digest( input, output, algorithm );
    } catch ( Exception e ) { e.printStackTrace(); }
  }

  private static void digest( String input, String output, String algorithm ) throws Exception {
    MessageDigest md = MessageDigest.getInstance( algorithm );
    System.out.println( "\nMessageDigest Object Info: " );
    System.out.println( "Algorithm = " + md.getAlgorithm() );
    System.out.println( "Provider = " + md.getProvider() );
    System.out.println( "Digest Length = " + md.getDigestLength() );

    FileInputStream in = new FileInputStream( input );
    int bufSize = 1024;
    byte[] buffer = new byte[bufSize];
    int n = in.read( buffer, 0, bufSize );
    int count = 0;
    while ( n != -1 ) {
      count += n;
      md.update( buffer, 0, n );
      n = in.read( buffer, 0, bufSize );
    }
    in.close();

    FileOutputStream out = new FileOutputStream( output );
    byte[] digest = md.digest();
    out.write( digest );
    out.close();
    System.out.println( "\nMessage Digest Processing Info: " );
    System.out.println( "Number of input bytes = " + count );
    System.out.println( "Number of output bytes = " + digest.length );
  }
}
```

Chương trình xuất Message Digest của tập tin JcaMessageDigest.class. Kết quả chạy chương trình:

```
javac -classpath . JcaMessageDigest.java
java -cp . JcaMessageDigest JcaMessageDigest.class digest.sha SHA

MessageDigest Object Info:
Algorithm = SHA
Provider = SUN version 1.6
Digest Length = 20
```

```
Message Digest Processing Info:
Number of input bytes = 2157
Number of output bytes = 20
```

Với các thông điệp đầu vào khác nhau ta sẽ thu được các Message Digest khác nhau nhưng có kích thước cố định như nhau.

Khóa private và khóa public

Cặp khóa private và public cần được sinh ra khi áp dụng mật mã bất đối xứng. Các thuật toán sinh khóa thường dùng là:

+ RSA – Mã hóa bất đối xứng (Ronald L. Rivest, Adi Shamir, và Leonard M. Adleman, 1977).
+ DSA – Thuật toán sinh chữ ký số (Digital Signature Algorithm).
+ DiffieHellman – Thuật toán trao đổi khóa.

Các giao diện Key

Có 3 giao diện thể hiện các khóa private và pubic:

`java.security.Key` mô tả các phương thức chung cho cả hai loại khóa private và public. Một số phương thức chính:

getAlgorithm()	Trả về tên thuật toán dùng sinh khóa.
getEncoded()	Trả về khóa như một mảng byte trong định dạng mã hóa, hoặc trả về null nếu không hỗ trợ mã hóa.
getFormat()	Trả về tên của định dạng mã hóa của khóa này, hoặc trả về null nếu không hỗ trợ mã hóa.

`java.security.PrivateKey` là giao diện thể hiện một khóa private. Thừa kế giao diện Key.

`java.security.PublicKey` là giao diện thể hiện một khóa public. Thừa kế giao diện Key.

Lớp KeyPair

`java.security.KeyPair` là lớp final thể hiện một cặp khóa (khóa public + khóa private). Một số phương thức chính:

getPrivate()	Trả về một đối tượng PrivateKey, thể hiện khóa private trong cặp khóa.
getPublic()	Trả về một đối tượng PublicKey, thể hiện khóa public trong cặp khóa.

Lớp KeyPairGenerator

`java.security.KeyPairGenerator` là lớp trừu tượng đại diện cho các lớp cài đặt các thuật toán khác nhau dùng sinh cặp khóa. Một số phương thức chính:

getInstance()	Trả về đối tượng KeyPairGenerator với thuật toán sinh khóa chỉ định.
initialize()	Khởi tạo quá trình sinh khóa với kích thước khóa chỉ định.
generateKeyPair()	Sinh một cặp khóa và trả chúng về trong đối tượng KeyPair.
getAlgorithm()	Trả về tên thuật toán sinh khóa được chỉ định.
getProvider()	Trả về nhà cung cấp đối tượng sinh khóa, mặc định là Sun.

Chương trình sau sẽ tạo đối tượng KeyPairGenerator với thuật toán sinh khóa và kích thước khóa chỉ định. Gọi phương thức generateKeyPair() của đối tượng này để tạo đối tượng KeyPair chứa cặp khóa. Cuối cùng gọi các phương thức getPrivate() và getPublic() của đối tượng KeyPair để kết xuất khóa private và khóa public ta cần.

```
import java.io.*;
import java.math.*;
import java.security.*;
import java.security.interfaces.*;
class JcaKeyPair {
  public static void main( String[] args ) {
    if ( args.length < 3 ) {
      System.out.println( "Usage: java JcaKeyPair keySize output algorithm" );
      return;
    }
    int keySize = Integer.parseInt( args[0] );
    String output = args[1];
    String algorithm = args[2];  // RSA, DSA
    try {
      getKeys( keySize, output, algorithm );
    } catch ( Exception e ) { e.printStackTrace(); }
  }

  private static void getKeys( int keySize, String output, String algorithm ) throws Exception {
    KeyPairGenerator kg = KeyPairGenerator.getInstance( algorithm );
    kg.initialize( keySize );
    System.out.println( "KeyPairGenerator Object Info: " );
    System.out.println( "Algorithm = " + kg.getAlgorithm() );
    System.out.println( "Provider = " + kg.getProvider() );
    System.out.println( "Key Size = " + keySize );

    KeyPair pair = kg.generateKeyPair();
    PrivateKey priKey = pair.getPrivate();
    PublicKey pubKey = pair.getPublic();

    String fl = output + ".pri";
    FileOutputStream out = new FileOutputStream( fl );
    byte[] ky = priKey.getEncoded();
    out.write( ky );
    out.close();
```

```
    System.out.println( "\nPrivate Key Info: " );
    System.out.println( "Algorithm = " + priKey.getAlgorithm() );
    System.out.println( "Saved File = " + fl );
    System.out.println( "Size = " + ky.length );
    System.out.println( "Format = " + priKey.getFormat() );
    System.out.println( "toString = " + priKey.toString() );

    fl = output + ".pub";
    out = new FileOutputStream( fl );
    ky = pubKey.getEncoded();
    out.write( ky );
    out.close();
    System.out.println( "\nPublic Key Info: " );
    System.out.println( "Algorithm = " + pubKey.getAlgorithm() );
    System.out.println( "Saved File = " + fl );
    System.out.println( "Size = " + ky.length );
    System.out.println( "Format = " + pubKey.getFormat() );
    System.out.println( "toString = " + pubKey.toString() );
  }
}
```

Kết quả chạy chương trình:

```
javac -classpath . JcaKeyPair.java
java -cp . JcaKeyPair 512 diff DiffieHellman

KeyPairGenerator Object Info:
Algorithm = DiffieHellman
Provider = SunJCE version 1.6
Key Size = 512

Private Key Info:
Algorithm = DH
Saved File = diff.pri
Size = 213
Format = PKCS#8
toString = SunJCE Diffie-Hellman Private Key:
x:
    a83c0aa6 f686a91b b5bd5c3a b03062b4 b141a7df 1fc26114 ef1b4546 2375cf22
    21e7f093 944a9421 ad4267ad 6c6b673a
p:
    fca682ce 8e12caba 26efccf7 110e526d b078b05e decbcd1e b4a208f3 ae1617ae
    01f35b91 a47e6df6 3413c5e1 2ed0899b cd132acd 50d99151 bdc43ee7 37592e17
g:
    678471b2 7a9cf44e e91a49c5 147db1a9 aaf244f0 5a434d64 86931d2d 14271b9e
    35030b71 fd73da17 9069b32e 2935630e 1c206235 4d0da20a 6c416e50 be794ca4
l:
    384

Public Key Info:
Algorithm = DH
Saved File = diff.pub
Size = 226
Format = X.509
toString = SunJCE Diffie-Hellman Public Key:
y:
    12a4ead1 9efdb672 2df31bd9 55605996 15aadb92 a7700c1d b15d9303 ea80271a
    62dc909e 8a1e7ef7 63344012 6bdeb592 35de78e6 b1f6651e d68cd89d 7905d8c0
p:
    fca682ce 8e12caba 26efccf7 110e526d b078b05e decbcd1e b4a208f3 ae1617ae
    01f35b91 a47e6df6 3413c5e1 2ed0899b cd132acd 50d99151 bdc43ee7 37592e17
g:
    678471b2 7a9cf44e e91a49c5 147db1a9 aaf244f0 5a434d64 86931d2d 14271b9e
    35030b71 fd73da17 9069b32e 2935630e 1c206235 4d0da20a 6c416e50 be794ca4
l:
    384
```

Lưu trữ khóa

Khóa private và public sau khi tạo ra được mã hóa dưới một định dạng theo chuẩn để lưu chuyển. Quá trình chuyển khóa thành định dạng theo chuẩn này gọi là mã hóa khóa (key encoding).

JDK cung cấp một nhóm các lớp hỗ trợ thao tác với khóa đã mã hóa:

+ KeyFactory – lớp chuyển đổi khóa giữa đối tượng Key và đối tượng EncodedKeySpec.
+ EncodedKeySpec – lớp trừu tượng đại diện các lớp cài đặt các chuẩn mã hóa khóa khác nhau.
+ PKCS8EncodedKeySpec – lớp con của EncodedKeySpec thể hiện mã hóa ASN.1 của khóa private theo chuẩn PKCS#8.

+ X509EncodedKeySpec – lớp con của EncodedKeySpec thể hiện mã hóa ASN.1 của khóa public theo chuẩn X.509.

Lớp PKCS8EncodedKeySpec

java.security.spec.PKCS8EncodedKeySpec là một lớp con của EncodedKeySpec, thể hiện mã hóa ASN.1 của khóa private theo chuẩn PKCS#8. Có 3 phương thức:

PKCS8EncodedKeySpec()	Khởi gán cho đối tượng PKCS8EncodedKeySpec từ mảng byte chỉ định chứa khóa private đã mã hóa theo chuẩn PKCS#8.
getEncoded()	Trả về khóa đã mã hóa lưu trong mảng byte của đối tượng này.
getFormat()	Trả về tên của phương pháp mã hóa áp dụng cho đối tượng này.

Lớp X509EncodedKeySpec

java.security.spec.X509EncodedKeySpec là một lớp con của EncodedKeySpec, thể hiện mã hóa ASN.1 của khóa public theo chuẩn X.509. Có 3 phương thức:

X509EncodedKeySpec()	Khởi gán cho đối tượng X509EncodedKeySpec từ mảng byte chỉ định chứa khóa public đã mã hóa theo X.509.
getEncoded()	Trả về khóa đã mã hóa lưu trong mảng byte của đối tượng này.
getFormat()	Trả về tên của phương pháp mã hóa áp dụng cho đối tượng này.

Lớp KeyFactory

java.security.KeyFactory là lớp dùng chuyển đổi khóa giữa đối tượng Key và EncodedKeySpec. Một số phương thức chính:

getInstance()	Trả về đối tượng KeyFactory với thuật toán chỉ định.
generatePrivate()	Sinh ra một đối tượng PrivateKey dựa trên đối tượng EncodedKeySpec chỉ định rồi trả về.
generatePublic()	Sinh ra một đối tượng PublicKey dựa trên đối tượng EncodedKeySpec chỉ định rồi trả về.
getKeySpec()	Chuyển một đối tượng Key thành một đối tượng EncodedKeySpec rồi trả về.
getAlgorithm()	Trả về tên thuật toán áp dụng cho đối tượng.
getProvider()	Trả về tên nhà cung cấp đối tượng.

Chương trình sinh cặp khóa với thuật toán chỉ định, lưu trữ khóa private và public vào tập tin với định dạng mã hóa mặc định. Sau đó, một đối tượng KeyFactory được tạo ra với cùng thuật toán sinh khóa. Nhờ đối tượng này, khóa private được đọc vào chương trình tạo thành đối tượng PrivateKey với sự hỗ trợ của lớp PKCS8EncodedKeySpec. Khóa public cũng được đọc vào chương trình thành đối tượng PublicKey với sự hỗ trợ của lớp X509EncodedKeySpec.

```
import java.io.*;
import java.math.*;
import java.security.*;
import java.security.interfaces.*;
import java.security.spec.*;
class JcaKeyFactoryTest {
  public static void main( String[] args ) {
    if ( args.length < 3 ) {
       System.out.println( "Usage: java JcaKeyFactoryTest keySize output algorithm" );
       return;
    }
    int keySize = Integer.parseInt( args[0] );
    String output = args[1];
    String algorithm = args[2];  // RSA, DSA
    try {
      writeKeys( keySize, output, algorithm );
      readKeys( output, algorithm );
    } catch ( Exception e ) { e.printStackTrace(); }
  }

  private static void writeKeys( int keySize, String output, String algorithm ) throws Exception {
    KeyPairGenerator kg = KeyPairGenerator.getInstance( algorithm );
    kg.initialize( keySize );
    KeyPair pair = kg.generateKeyPair();
    PrivateKey priKey = pair.getPrivate();
    PublicKey pubKey = pair.getPublic();

    String fl = output + ".pri";
    FileOutputStream out = new FileOutputStream( fl );
    byte[] ky = priKey.getEncoded();
    out.write( ky );
    out.close();

    fl = output + ".pub";
    out = new FileOutputStream(fl);
    ky = pubKey.getEncoded();
    out.write(ky);
    out.close();
  }

  private static void readKeys( String input, String algorithm ) throws Exception {
    KeyFactory keyFactory = KeyFactory.getInstance( algorithm );
    System.out.println( "\nKeyFactory Object Info: " );
```

```
    System.out.println( "Algorithm = " + keyFactory.getAlgorithm() );
    System.out.println( "Provider = " + keyFactory.getProvider() );

    String priKeyFile = input + ".pri";
    FileInputStream priKeyStream = new FileInputStream( priKeyFile );
    int priKeyLength = priKeyStream.available();
    byte[] priKeyBytes = new byte[priKeyLength];
    priKeyStream.read( priKeyBytes );
    priKeyStream.close();
    PKCS8EncodedKeySpec priKeySpec = new PKCS8EncodedKeySpec( priKeyBytes );
    PrivateKey priKey = keyFactory.generatePrivate( priKeySpec );
    System.out.println( "\nPrivate Key Info: " );
    System.out.println( "Algorithm = " + priKey.getAlgorithm() );
    System.out.println( "Saved File = " + priKeyFile );
    System.out.println( "Length = " + priKeyBytes.length );
    System.out.println( "toString = " + priKey.toString() );

    String pubKeyFile = input + ".pub";
    FileInputStream pubKeyStream = new FileInputStream( pubKeyFile );
    int pubKeyLength = pubKeyStream.available();
    byte[] pubKeyBytes = new byte[pubKeyLength];
    pubKeyStream.read( pubKeyBytes );
    pubKeyStream.close();
    X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec( pubKeyBytes );
    PublicKey pubKey = keyFactory.generatePublic( pubKeySpec );
    System.out.println( "\nPublic Key Info: " );
    System.out.println( "Algorithm = " + pubKey.getAlgorithm() );
    System.out.println( "Saved File = " + pubKeyFile );
    System.out.println( "Length = " + pubKeyBytes.length );
    System.out.println( "toString = " + pubKey.toString() );
  }
}
```

Biên dịch và chạy chương trình:

```
javac –classpath . JcaKeyFactoryTest.java
java -cp . JcaKeyFactoryTest 512 rsa RSA

KeyFactory Object Info:
Algorithm = RSA
Provider = SunRsaSign version 1.5

Private Key Info:
Algorithm = RSA
Saved File = rsa.pri
Length = 347
toString = Sun RSA private CRT key, 512 bits
  modulus:          11003888284993245494907659801671829487428429487074286690683
960279930570016537264685748235509700080951277097769392825545683397471811830411
835176950963219
  public exponent:  65537
  private exponent: 996372642336495680545051418207744793693604929462688741929828
783272295022036594557927742526250765857980019847291830178612585564383615965574
65874845562913
  prime p:          109357834439850516248568581865730406612013474379214109161024
380621260761632331
  prime q:          100622770571098435552854300796570551969786867102635965581462
713232020740009049
  prime exponent p: 847503610357509150596578768170719952366474566080272915191178
76798660818824573
  prime exponent q: 625550992779386233689357580810027007477459500185040075939627
28309385858525545
  crt coefficient:  857021660690930146721604682351041540563579380353536096463881
84675283308103002

Public Key Info:
Algorithm = RSA
Saved File = rsa.pub
Length = 94
toString = Sun RSA public key, 512 bits
  modulus: 110038882849932454949076598016718294874284294870742866906836960279930
570016537264685748235509700080951277097769392825545683397471811830411418351769 50
963219
  public exponent: 65537
```

Digital Signature

Chữ ký số giúp đảm bảo hai vấn đề bảo mật: xác thực (authentication) và toàn vẹn (integrity).
Người gởi "ký" lên tài liệu: Message Digest của tài liệu được mã hóa bằng khóa private của người gởi để tạo chữ ký số. Chữ ký số được gởi kèm với tài liệu đến người nhận, khóa public của người gởi cũng được phân phối đến người nhận.
Người nhận xác minh chữ ký số: chữ ký số được giải mã bằng khóa public của người gởi để nhận được Message Digest, Message Digest này được so sánh với Message Digest đối chứng tạo từ người nhận với cùng thuật toán băm. Nếu cả hai được so trùng, chữ ký được xác minh.

Lớp Signature

java.security.Signature là lớp trừu tượng đại diện cho các lớp cài đặt các thuật toán chữ ký số khác nhau. Một số phương thức chính:

getInstance()	Trả về đối tượng Signature với thuật toán tạo chữ ký số chỉ định.
initSign()	Khởi tạo đối tượng Signature cho quá trình ký với khóa private chỉ định.
initVerify()	Khởi tạo đối tượng Signature cho quá trình xác minh với khóa public chỉ định.
update()	Thêm dữ liệu cho đối tượng Signature khi ký hoặc xác minh.
sign()	Sinh chữ ký với dữ liệu mà đối tượng Signature nhận được rồi trả về trong mảng byte.
verify()	Xác minh chữ ký chỉ định có phù hợp với dữ liệu mà đối tượng Signature nhận được, trả về true hoặc false.
getAlgorithm()	Trả về tên thuật toán áp dụng cho đối tượng Signature.
getProvider()	Trả về tên nhà cung cấp của đối tượng Signature.

Trước tiên, chương trình dùng thuật toán sinh khóa keyAlgo để sinh cặp khóa. Khóa private sẽ tham gia quá trình ký tài liệu với thuật toán ký signAlgo. Khóa public sẽ tham gia quá trình xác minh chữ ký với thuật toán ký signAlgo. Khi ký hoặc xác minh, dữ liệu lần lượt đưa vào một mảng byte, đối tượng Signature gọi phương thức update() để cập nhật dữ liệu từ mảng byte này. Sau đó phương thức sign() được gọi để ký hoặc phương thức verify() được gọi để xác minh.

```
import java.io.*;
import java.security.*;
class JcaSignatureTest {
  public static void main( String[] args ) {
    if ( args.length < 4 ) {
      System.out.println( "Usage: java JcaSignatureTest input output keyAlgo signAlgo" );
      return;
    }
    String input = args[0];
    String output = args[1];
    String keyAlgo = args[2];
    String signAlgo = args[3];
    try {
      KeyPair pair = getKeys( keyAlgo );
      PrivateKey priKey = pair.getPrivate();
      PublicKey pubKey = pair.getPublic();
      byte[] sign = sign( input, output, signAlgo, priKey );
      verify( input, signAlgo, sign, pubKey );
    } catch ( Exception e ) { e.printStackTrace(); }
  }

  private static KeyPair getKeys( String algorithm ) throws Exception {
    KeyPairGenerator kg = KeyPairGenerator.getInstance( algorithm );
    int keySize = 512;
    kg.initialize( keySize );
    KeyPair pair = kg.generateKeyPair();
    return pair;
  }

  private static byte[] sign( String input, String output, String algorithm, PrivateKey priKey )
     throws Exception {
    Signature sg = Signature.getInstance( algorithm );
    sg.initSign( priKey );
    System.out.println( "\nSignature Object Info: " );
    System.out.println( "Algorithm = " + sg.getAlgorithm() );
    System.out.println( "Provider = " + sg.getProvider() );
    FileInputStream in = new FileInputStream( input );
    int bufSize = 1024;
    byte[] buffer = new byte[bufSize];
    int n = in.read( buffer, 0, bufSize );
    int count = 0;
    while ( n != -1 ) {
      count += n;
      sg.update( buffer, 0, n );
      n = in.read( buffer, 0, bufSize );
    }
    in.close();
    FileOutputStream out = new FileOutputStream( output );
    byte[] sign = sg.sign();
```

```
    out.write( sign );
    out.close();
    System.out.println( "\nSign Processing Info: " );
    System.out.println( "Number of input bytes = " + count );
    System.out.println( "Number of output bytes = " + sign.length );
    return sign;
  }

  private static boolean verify( String input, String algorithm, byte[] sign, PublicKey pubKey )
     throws Exception {
    Signature sg = Signature.getInstance( algorithm );
    sg.initVerify( pubKey );
    System.out.println( "Signature Object Info: " );
    System.out.println( "Algorithm = " + sg.getAlgorithm() );
    System.out.println( "Provider = " + sg.getProvider() );
    FileInputStream in = new FileInputStream( input );
    int bufSize = 1024;
    byte[] buffer = new byte[bufSize];
    int n = in.read( buffer, 0, bufSize );
    int count = 0;
    while ( n != -1 ) {
      count += n;
      sg.update( buffer, 0, n );
      n = in.read( buffer, 0, bufSize );
    }
    in.close();
    boolean ok = sg.verify( sign );
    System.out.println( "Verify Processing Info: " );
    System.out.println( "Number of input bytes = " + count );
    System.out.println( "Verification result = " + ok );
    return ok;
  }
}
```

Thuật toán sinh khóa là DSA, thuật toán ký là SHA1withDSA (dùng SHA1 để tạo Message Digest và dùng DSA để mã hóa bất đối xứng). Tài liệu được ký và xác minh là JcaSignatureTest.class. Kết quả chạy chương trình:

```
javac -classpath . JcaSignatureTest.java
java -cp . JcaSignatureTest JcaSignatureTest.class sign.dsa DSA SHA1withDSA

Signature Object Info:
Algorithm = SHA1withDSA
Provider = SUN version 1.6

Sign Processing Info:
Number of input bytes = 3364
Number of output bytes = 46
Signature Object Info:
Algorithm = SHA1withDSA
Provider = SUN version 1.6
Verify Processing Info:
Number of input bytes = 3364
Verification result = true
```

Thực tế cần hai chương trình tách biệt:
- Chương trình sinh chữ ký: tạo cặp khóa, lưu cặp khóa, sinh chữ ký theo thuật toán ký chỉ định.
- Chương trình xác minh: đọc khóa public từ tập tin, tiến hành xác minh chữ ký theo thuật toán ký chỉ định.

Certificate

Việc phân phối khóa public không có xác thực dễ bị tấn công xen giữa, người được phân phối có thể nhận khóa public giả mạo. Chứng thực khóa công khai (certificate) liên kết khóa public với định danh của chủ thể tạo ra nó (subject), bằng cách ký lên khóa public với chữ ký của bên thứ ba tin cậy. Bên thứ ba tin cậy gọi là Certifiace Authority (CA) hoặc bên phát hành khóa (issuer).
Certificate thường lưu trong định dạng chuẩn X.509.
Để có được certificate cho khóa public của mình, ta dùng một dịch vụ chứng thực nào đó (ví dụ VeriSign) hoặc dùng một công cụ như keytool sinh ra certificate tự ký.
Thường CA ban đầu (với certifiacte tự ký) sẽ ký cho khóa public của CA cấp thấp hơn, và cứ như thế cho đến khi CA kế tiếp ký cho khóa public của chủ thể yêu cầu ký. Như vậy một khóa public thường liên kết với một chuỗi chứng thực khóa công khai (certificate chain).

keytool và keystore

keystore là cơ sở dữ liệu dùng lưu các entry bảo mật. Mỗi entry có thể là:
+ certificate: chứng thực khóa công khai của một khóa public.
+ key: cặp khóa, bao gồm khóa private và khóa public (liên kết với chuỗi certificate).

Mỗi entry trong keystore có một tên duy nhất, entry cũng lưu cả thời gian thay đổi entry gần nhất.
keystore được bảo vệ bằng mật khẩu. Khi keystore được lưu lần đầu, phải nhập mật khẩu và mật khẩu này cần khi nạp lại keystore.

Entry kiểu key cũng được bảo vệ bằng mật khẩu, chủ yếu là bảo vệ khóa private trong cặp khóa.
keytool là một công cụ dòng lệnh dùng quản lý khóa và certificate trong keystore. keytool cung cấp một số tùy chọn cho phép truy xuất keystore:
-genkey Sinh một cặp khóa và lưu trữ nó vào entry kiểu key trong keystore.
Ví dụ sau sinh một cặp khóa và lưu nó vào entry kiểu key của keystore. Khóa public trong cặp khóa liên kết với certificate tự ký của keystore.

```
keytool -genkey -alias my_home -keystore myhome.jks
Enter keystore password: MyHomeJKS                          # không hiển thị mật khẩu nhập vào
Re-enter new password: MyHomeJKS                            # mật khẩu này dùng bảo vệ keystore
What is your first and last name?
  [Unknown]:  Karl Marx
What is the name of your organizational unit?
  [Unknown]:  My Unit
What is the name of your organization?
  [Unknown]:  My Home
What is the name of your City or Locality?
  [Unknown]:  My City
What is the name of your State or Province?
  [Unknown]:  My State
What is the two-letter country code for this unit?
  [Unknown]:  VN
Is CN=Karl Marx, OU=My Unit, O=My Home, L=My City, ST=My State, C=VN correct?
  [no]:  yes

Enter key password for <my_home>
        (RETURN if same as keystore password): My1stKey     # không hiển thị mật khẩu nhập vào
Re-enter new password: My1stKey                             # mật khẩu này dùng bảo vệ entry kiểu key
```

-list Liệt kê tất cả entry trong một keystore.
Ví dụ sau liệt kê các entry của keystore myhome.jks. Cần có mật khẩu bảo vệ keystore để truy xuất keystore.

```
keytool -list -keystore myhome.jks –storepass MyHomeJKS

Keystore type: JKS
Keystore provider: SUN

Your keystore contains 1 entry

my_home, Aug 17, 2010, PrivateKeyEntry,
Certificate fingerprint (MD5): 2C:C0:A2:05:9F:0C:8C:55:69:2C:2A:5D:17:B3:EF:10
```

-export Kết xuất certificate của entry kiểu key chỉ định hoặc của entry kiểu certificate ra tập tin.
-printcert In thông tin của certifiacte từ tập tin chứa certificate.
Ví dụ sau kết xuất certificate liên kết với khóa public của entry my_home ra tập tin my_home.crt. Sau đó xuất thông tin của certificate lưu trong tập tin này. Cần có mật khẩu bảo vệ keystore để truy xuất keystore.

```
keytool -export -alias my_home -file my_home.crt -keystore myhome.jks -storepass MyHomeJKS
Certificate stored in file <my_home.crt>

keytool -printcert -file my_home.crt
Owner: CN=Karl Marx, OU=My Unit, O=My Home, L=My City, ST=My State, C=VN
Issuer: CN=Karl Marx, OU=My Unit, O=My Home, L=My City, ST=My State, C=VN
Serial number: 4c6a206f
Valid from: Tue Aug 17 12:38:55 ICT 2010 until: Mon Nov 15 12:38:55 ICT 2010
Certificate fingerprints:
         MD5:  2C:C0:A2:05:9F:0C:8C:55:69:2C:2A:5D:17:B3:EF:10
         SHA1: 90:83:02:49:E4:DF:7C:4E:6B:FA:3F:9D:86:53:59:AE:E6:C8:AD:5D
         Signature algorithm name: SHA1withDSA
         Version: 3
```

Có thể kết xuất certificate dưới dạng in được, ví dụ mã hóa Base64, dựa trên đặc tả RFC 1421. Cần có mật khẩu bảo vệ keystore để truy xuất keystore.

```
keytool -export -alias my_home_crt -file my_h
ome.rfc -rfc -keystore myhome.jks -storepass MyHomeJKS
Certificate stored in file <my_home.rfc>

type my_home.rfc
-----BEGIN CERTIFICATE-----
MIIDDjCCAsygAwIBAgIETGogbzALBgcqhkjOOAQDBQAwajELMAkGA1UEBhMCVk4xETAPBgNVBAgT
CE15IFN0YXRlMRAwDgYDVQQHEwdNeSBDaXR5MRAwDgYDVQQKEwdNeSBIb21lMRAwDgYDVQQLEwdN
eSBVbml0MRIwEAYDVQQDEwlLYXJsIE1hcngwHhcNMTAwODE3MDUzODU1WhcNMTAxMTE1MDUzODU1
WjBqMQswCQYDVQQGEwJWTjERMA8GA1UECBMITXkgU3RhdGUxEDAOBgNVBAcTB015IENpdHkxEDAO
BgNVBAoTB015IEhvbWUxEDAOBgNVBAsTB015IFVuaXQxEjAQBgNVBAMTCUthcmwgTWFyeDCCAbgw
ggEsBgcqhkjOOAQBMIIBHwKBgQD9f1OBHXUSKVLfSpwu7OTn9hG3UjzvRADDHj+AtlEmaUVdQCJR
+1k9jVj6v8X1ujD2y5tVbNeBO4AdNG/yZmC3a5lQpaSfn+gEexAiwk+7qdf+t8Yb+DtX58aophUP
BPuD9tPFHsMCNVQTWhaRMvZ1864rYdcq7/IiAxmd0UgBxwIVAJdgUI8VIwvMspK5gqLrhAvwWBz1
```

```
AoGBAPfhoIXWmz3ey7yrXDa4V7l5lK+7+jrqgvlXTAs9B4JnUVlXjrrUWU/mcQcQgYC0SRZxI+hM
KBYTt88JMozIpuE8FnqLVHyNKOCjrh4rs6Z1kW6jfwv6ITVi8ftiegEkO8yk8b6oUZCJqIPf4Vrl
nwaSi2ZegHtVJWQBTDv+z0kqA4GFAAKBgQDbwrYYdZ0Q+roNPbmHvH5J3v1GQyhXgooyN6hzXZE/
h6bDOEOrYzA4q5zCYPJveZf7z9/mF0rJu9MjKRnWAeR1kHOxE7FakRRWRCLJ5soxjD1IgoLp/ur/
uGyKQLdjT9OYH4obaG6PX4fc6DDb6kPKPQeBm42kg01MLCXd0p82DzALBgcqhkjOOAQDBQADLwAw
LAIUN65mbdI1km+iWchTl0oScmRGchQCFAQ4hmjm+2+e8Z2twpeog+RhGW4r
-----END CERTIFICATE-----
```

-import Thêm certificate vào một entry kiểu certificate trong keystore ra tập tin.

Ví dụ sau thêm certificate lưu trong tập tin my_home.crt vào keystore, lưu với entry tên my_home_crt. Cần có mật khẩu bảo vệ keystore để truy xuất keystore.

```
keytool -import -alias my_home_crt -file my_home.crt -keystore myhome.jks -storepass MyHomeJKS
Certificate already exists in keystore under alias <my_home>
Do you still want to add it? [no]:  yes
Certificate was added to keystore
```

-keyclone Nhân bản entry với tên mới.

Ví dụ sau nhân bản entry kiểu key my_home thành my_copy. Cần có mật khẩu bảo vệ keystore để thêm entry, cần có mật khẩu entry để sao chép được cặp khóa (chủ yếu là khóa private trong đó), mật khẩu của entry mới là My2ndKey.

```
keytool -keyclone -alias my_home -dest my_copy -keypass My1stKey -new My2ndKey
        -keystore myhome.jks -storepass MyHomeJKS
```

-selfcert Thay thế certificate liên kết với entry kiểu key bằng một certificate tự ký mới.

Ví dụ sau thay thế certificate liên kết với entry my_copy bằng certificate tự ký khác. Cần có mật khẩu bảo vệ keystore để truy xuất keystore, cần có mật khẩu entry để thay đổi entry.

```
keytool -selfcert -alias my_copy -keypass My2ndKey -dname "cn=Vladimir Ilyich Lenin, ou=My Unit 2,
o=My Office, c=RU" -keystore myhome.jks -storepass MyHomeJKS
```

-delete Xóa một entry với tên chỉ định.

Ví dụ sau xóa entry my_copy trong keystore. Cần có mật khẩu bảo vệ keystore để truy xuất keystore. Dùng tùy chọn -list để thấy entry đã bị xóa.

```
keytool -delete -alias my_copy -keystore myhome.jks -storepass MyHomeJKS
```

Tuy là ứng dụng chuẩn của JDK, keytool vẫn khó sử dụng do giao diện dòng lệnh, ta có thể dùng công cụ có giao diện đồ họa trực quan hơn. Ví dụ Keystore Explorer (http://www.lazgosoftware.com/kse/downloads.html).

Lớp Certificate

java.security.cert.Certificate là lớp trừu tượng đại diện cho các lớp cài đặt kiểu certificate khác nhau. Một số phương thức chính:

getEncoded()	Trả về mảng byte dạng mã hóa của Certificate này. Định dạng được dùng tùy theo kiểu certificate. Với certificate kiểu X.509, định dạng mã hóa sẽ là "ASN.1 DER".
getPublicKey()	Trả về khóa public được ký với Certificate này.
getType()	Trả về kiểu của đối tượng Certificate.
verify()	Xác minh Certificate với khóa public chỉ định.

Lớp CertificateFactory

java.security.cert.CertificateFactory là lớp dùng đọc các kiểu certificate khác nhau từ tập tin chứa certificate.

getIntance()	Trả về đối tượng CertificateFactory theo kiểu certificate chỉ định.
generateCertificate()	Đọc vào một certificate từ stream nhập chỉ định và trả về một đối tượng Certificate. Có thể đọc certificate trong cả hai định dạng: binary (mã hóa DER - Distinguished Encoding Rules) và dạng in được (chuẩn RFC). Chú ý chỉ có certificate đầu trong stream nhập được xử lý.
generateCertificates()	Đọc vào nhiều certificate từ stream nhập chỉ định, trả về một mảng Certificate. Có thể đọc các certificate trong cả hai định dạng: DER và RFC.
getType()	Trả về kiểu của đối tượng CertificateFactory.
getProvider()	Trả về nhà cung cấp đối tượng CertificateFactory.

Chương trình sau sẽ đọc một certificate từ tập tin chứa nó, xuất ra một tập tin certificate khác. Chương trình cũng hiển thị thông tin của certificate và khóa public liên kết với nó.

```
import java.io.*;
import java.security.*;
import java.security.cert.*;
class JcaCertificateTest {
  public static void main( String[] args ) {
    if ( args.length < 2 ) {
      System.out.println( "Usage: java JcaCertificateFactoryTest input output" );
      return;
    }
    String input = args[0];
    String output = args[1];
    try {
      test( input, output );
    } catch ( Exception e ) { e.printStackTrace(); }
  }

  private static void test( String input, String output ) throws Exception {
    CertificateFactory cf = CertificateFactory.getInstance( "X.509" );
```

```
    System.out.println( "\nCertificateFactory Object Info: " );
    System.out.println( "Type = " + cf.getType() );
    System.out.println( "Provider = " + cf.getProvider() );

    FileInputStream fis = new FileInputStream( input );
    java.security.cert.Certificate cert = cf.generateCertificate( fis );
    fis.close();
    System.out.println( "\nCertificate Object Info: " );
    System.out.println( "Type = " + cert.getType() );
    System.out.println( "toString = " + cert.toString() );

    PublicKey pubKey = cert.getPublicKey();
    System.out.println( "\nPublicKey Object Info: " );
    System.out.println( "Algorithm = " + pubKey.getAlgorithm() );
    System.out.println( "Format = " + pubKey.getFormat() );
    System.out.println( "toString = " + pubKey.toString() );

    FileOutputStream fos = new FileOutputStream( output );
    byte[] certBytes = cert.getEncoded();
    fos.write( certBytes );
    fos.close();
  }
}
```

Tập tin certificate đầu vào được tạo bằng keytool (.crt), chú ý tập tin certificate với định dạng in được (.rfc) cũng hợp lệ. Kết quả chạy chương trình:

```
javac -classpath . JcaCertificateTest.java
java -cp . JcaCertificateTest my_home.crt temp.crt

CertificateFactory Object Info:
Type = X.509
Provider = SUN version 1.6

Certificate Object Info:
Type = X.509
toString = [
[
  Version: V3
  Subject: CN=Karl Marx, OU=My Unit, O=My Home, L=My City, ST=My State, C=VN
  Signature Algorithm: SHA1withDSA, OID = 1.2.840.10040.4.3

  Key:  Sun DSA Public Key
    Parameters:DSA
        p:     fd7f5381 1d751229 52df4a9c 2eece4e7 f611b752 3cef4400 c31e3f80 b6512669
    455d4022 51fb593d 8d58fabf c5f5ba30 f6cb9b55 6cd7813b 801d346f f26660b7
    6b9950a5 a49f9fe8 047b1022 c24fbba9 d7feb7c6 1bf83b57 e7c6a8a6 150f04fb
    83f6d3c5 1ec30235 54135a16 9132f675 f3ae2b61 d72aeff2 2203199d d14801c7
        q:     9760508f 15230bcc b292b982 a2eb840b f0581cf5
        g:     f7e1a085 d69b3dde cbbcab5c 36b857b9 7994afbb fa3aea82 f9574c0b 3d078267
    5159578e bad4594f e6710710 8180b449 167123e8 4c281613 b7cf0932 8cc8a6e1
    3c167a8b 547c8d28 e0a3ae1e 2bb3a675 916ea37f 0bfa2135 62f1fb62 7a01243b
    cca4f1be a8519089 a883dfe1 5ae59f06 928b665e 807b5525 64014c3b fecf492a

  y:
    dbc2b618 759d10fa ba0d3db9 87bc7e49 defd4643 2857828a 3237a873 5d913f87
    a6c33843 ab633038 ab9cc260 f26f7997 fbcfdfe6 174ac9bb d3232919 d601e475
    9073b113 b15a9114 564422c9 e6ca318c 3d488282 e9feeaff b86c8a40 b7634fd3
    981f8a1b 686e8f5f 87dce830 dbea43ca 3d07819b 8da4834d 4c2c25dd d29f360f

  Validity: [From: Tue Aug 17 12:38:55 ICT 2010,
              To: Mon Nov 15 12:38:55 ICT 2010]
  Issuer: CN=Karl Marx, OU=My Unit, O=My Home, L=My City, ST=My State, C=VN
  SerialNumber: [    4c6a206f]

]
  Algorithm: [SHA1withDSA]
  Signature:
0000: 30 2C 02 14 37 AE 66 6D   D2 35 92 6F A2 59 C8 53  0,..7.fm.5.o.Y.S
0010: 97 4A 12 72 64 46 72 14   02 14 04 38 86 68 E6 FB  .J.rdFr....8.h..
0020: 6F 9E F1 9D AD C2 97 A8   83 E4 61 19 6E 2B        o.........a.n+

]

PublicKey Object Info:
```

```
Algorithm = DSA
Format = X.509
toString = Sun DSA Public Key
    Parameters:DSA
        p:     fd7f5381 1d751229 52df4a9c 2eece4e7 f611b752 3cef4400 c31e3f80 b6512669
    455d4022 51fb593d 8d58fabf c5f5ba30 f6cb9b55 6cd7813b 801d346f f26660b7
    6b9950a5 a49f9fe8 047b1022 c24fbba9 d7feb7c6 1bf83b57 e7c6a8a6 150f04fb
    83f6d3c5 1ec30235 54135a16 9132f675 f3ae2b61 d72aeff2 2203199d d14801c7
        q:     9760508f 15230bcc b292b982 a2eb840b f0581cf5
        g:     f7e1a085 d69b3dde cbbcab5c 36b857b9 7994afbb fa3aea82 f9574c0b 3d078267
    5159578e bad4594f e6710710 8180b449 167123e8 4c281613 b7cf0932 8cc8a6e1
    3c167a8b 547c8d28 e0a3ae1e 2bb3a675 916ea37f 0bfa2135 62f1fb62 7a01243b
    cca4f1be a8519089 a883dfe1 5ae59f06 928b665e 807b5525 64014c3b fecf492a

  y:
    dbc2b618 759d10fa ba0d3db9 87bc7e49 defd4643 2857828a 3237a873 5d913f87
    a6c33843 ab633038 ab9cc260 f26f7997 fbcfdfe6 174ac9bb d3232919 d601e475
    9073b113 b15a9114 564422c9 e6ca318c 3d488282 e9feeaff b86c8a40 b7634fd3
    981f8a1b 686e8f5f 87dce830 dbea43ca 3d07819b 8da4834d 4c2c25dd d29f360f
```

Lớp KeyStore

java.security.KeyStore là lớp thể hiện keystore trong bộ nhớ, như vậy nó cung cấp các phương thức giống keytool, cho phép truy xuất các entry của keystore. Một số phương thức chính:

getIntance()	Trả về đối tượng KeyStore với kiểu keystore chỉ định.
load()	Xóa tất cả entry và nạp các entry mới từ stream nhập chỉ định nối với keystore. Cần mật khẩu của keystore.
store()	Lưu tất cả các entry vào một stream xuất với mật khẩu chỉ định.
size()	Trả về số các entry có trong keystore.
aliases()	Trả về tên danh sách tên các entry trong một đối tượng Enumeration.
deleteEntry()	Xóa entry với tên chỉ định.
getCertificate()	Trả về certificate của entry với tên chỉ định. Với entry kiểu key, trả về certificate đầu của chuỗi certificate.
getCertificateChain()	Trả về chuỗi certificate (trong một mảng Certificate) của entry với tên chỉ định. Với entry kiểu certificate, trả về null.
getKey()	Trả về khóa của entry với tên chỉ định, cần cung cấp mật khẩu. Với entry kiểu certificate, trả về null.
setCertificateEntry()	Thêm một entry kiểu certificate mới với tên và certificate chỉ định.
setKeyEntry()	Thêm một entry liểu key mới với tên chỉ định, chuỗi certificate chỉ định và mật khẩu chỉ định cho entry.
isKeyEntry()	Trả về true nếu entry với tên chỉ định là entry kiểu key.
isCertificateEntry()	Trả về true nếu entry với tên chỉ định là entry kiểu certificate.
getType()	Trả về kiểu của KeyStore.
getProvider()	Trả về nhà cung cấp KeyStore.

Chương trình sau trình bày cách nạp dữ liệu từ một tập tin keystore vào đối tượng KeyStore bằng phương thức load(); trích certificate từ keystore bằng phương thức getCertificate(), lưu vào một tập tin certificate; sau đó nhập certificate từ tập tin vào keystore bằng phương thức setCertificateEntry(). Cuối cùng dùng phương thức store() lưu đối tượng KeyStore vào tập tin keystore ban đầu.

```
import java.io.*;
import java.util.*;
import java.security.*;
import java.security.cert.*;
class JcaKeyStoreTest {
  public static void main( String[] args ) {
    if ( args.length < 3 ) {
      System.out.println( "Usage: java JcaKeyStoreTest store sPass alias" );
      return;
    }
    String store = args[0];
    String sPass = args[1];
    String alias = args[2];
    try {
      test( store, sPass, alias );
    } catch ( Exception e ) { e.printStackTrace(); }
  }

  private static void test( String store, String sPass, String alias ) throws Exception {
    KeyStore ks = KeyStore.getInstance( "JKS" );
    System.out.println( "\nKeyStore Object Info: " );
    System.out.println( "Type = " + ks.getType() );
    System.out.println( "Provider = " + ks.getProvider() );

    FileInputStream fis = new FileInputStream( store );
    ks.load( fis, sPass.toCharArray() );
    fis.close();
    System.out.println( "\nKeyStore Content: " );
    System.out.println( "Size = " + ks.size() );
    Enumeration e = ks.aliases();
```

```
    while ( e.hasMoreElements() ) {
      String name = ( String )e.nextElement();
      System.out.print( "   " + name + ": " );
      if ( ks.isKeyEntry( name ) ) System.out.println( " Key entry" );
      else System.out.println( " Certificate entry" );
    }

    java.security.cert.Certificate cert = ks.getCertificate( alias );
    System.out.println( "\nCertificate Object Info: " );
    System.out.println( "Type = " + cert.getType() );
    FileOutputStream fos = new FileOutputStream( alias + ".crt" );
    byte[] certBytes = cert.getEncoded();
    fos.write( certBytes );
    fos.close();

    CertificateFactory cf = CertificateFactory.getInstance( "X.509" );
    System.out.println( "\nCertificateFactory Object Info: " );
    System.out.println( "Type = " + cf.getType() );
    System.out.println( "Provider = " + cf.getProvider() );
    fis = new FileInputStream( alias + ".crt" );
    cert = cf.generateCertificate( fis );
    ks.setCertificateEntry( alias + ks.size(), cert );
    fis.close();

    fos = new FileOutputStream( store );
    ks.store( fos, sPass.toCharArray() );
    fos.close();
  }
}
```

Kết quả chạy chương trình:

```
javac -classpath . JcaKeyStoreTest.java
java -cp . JcaKeyStoreTest myhome.jks MyHomeJKS my_home

KeyStore Object Info:
Type = JKS
Provider = SUN version 1.6

KeyStore Content:
Size = 2
   my_home_crt:  Certificate entry
   my_home:  Key entry

Certificate Object Info:
Type = X.509

CertificateFactory Object Info:
Type = X.509
Provider = SUN version 1.6
```