Java Crypto Howto

Digest

Uitrekenen van een SHA-1 digest van een object:

```
package info.boppelans.crypto;

import java.security.MessageDigest;

public class Digest {
   public byte[] digest(String object)
   throws Exception {
      MessageDigest sha1 = MessageDigest.getInstance("sha-1");
      byte[] dig = sha1.digest(object.getBytes());

      System.out.println(dig.length);
      return dig;
   }
}
```

DES Encryption

```
package info.boppelans.crypto;

import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.Certificate;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;

public class Des {

   SecretKey key;

   public Des()
   throws NoSuchAlgorithmException {
      key = KeyGenerator.getInstance("DES").generateKey();
   }

   public byte[] encrypt(byte[] data)
   throws NoSuchPaddingException, NoSuchAlgorithmException,
   InvalidKeyException, BadPaddingException, IllegalBlockSizeException {

      Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");

      cipher.init(Cipher.ENCRYPT_MODE, key);
      return cipher.doFinal(data);
   }

   public byte[] decrypt(byte[] cipherData)
   throws NoSuchPaddingException, NoSuchAlgorithmException,
```

```
   InvalidKeyException, BadPaddingException, IllegalBlockSizeException {

      Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");

      cipher.init(Cipher.DECRYPT_MODE, key);
      return cipher.doFinal(cipherData);

   }

}
```

Alternatieven voor DES:

- DESede (3DES)
- AES

DSA en RSA Sign

Keypair genereren, signen en verifiëren

```
package info.boppelans.crypto;

import java.security.InvalidKeyException;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.SecureRandom;
import java.security.Signature;
import java.security.SignatureException;

public class PubPriv {

   private KeyPair keypair;

   public PubPriv()
   throws NoSuchAlgorithmException, NoSuchProviderException {
      KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA");
//      KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");

      SecureRandom random = SecureRandom.getInstance("SHA1PRNG", "SUN");
      random.setSeed(555L);
      keyGen.initialize(1024, random);

      keypair = keyGen.generateKeyPair();

      PrivateKey privatekey = keypair.getPrivate();
      PublicKey publickey = keypair.getPublic();

      System.out.println(privatekey.toString());
      System.out.println(publickey.toString());
   }

   public byte[] sign(byte[] object)
   throws NoSuchAlgorithmException, InvalidKeyException, SignatureException {
      Signature dsa = Signature.getInstance("SHA1withDSA");
//      Signature dsa = Signature.getInstance("SHA1withRSA");

      dsa.initSign(keypair.getPrivate());
      dsa.update(object);
      byte[] sig = dsa.sign();

      return sig;
   }

   public boolean verify(byte[] object, byte[] sig)
   throws NoSuchAlgorithmException, InvalidKeyException, SignatureException {
```

```
      Signature dsa = Signature.getInstance("SHA1withDSA");
//      Signature dsa = Signature.getInstance("SHA1withRSA");

      /* Initializing the object with the public key */
      dsa.initVerify(keypair.getPublic());

      /* Update and verify the data */
      dsa.update(object);
      boolean verifies = dsa.verify(sig);

      return verifies;
   }
}
```

Certificaten

Inlezen van een certificate

```
      FileInputStream fis = new FileInputStream("cacert.pem");
      CertificateFactory cf = CertificateFactory.getInstance("X.509");
      Certificate cert = cf.generateCertificate(fis);
```

Inlezen van een PKCS12 store

```
      // Open een keystore
      KeyStore ks = KeyStore.getInstance("pkcs12");
      FileInputStream ksf = new FileInputStream("cacert.pkcs12");
      ks.load(ksf, "hallowereld".toCharArray());

      // Toon elementen in keystore
      Enumeration<String> aliasEnum = ks.aliases();
      while(aliasEnum.hasMoreElements())
         System.out.println("Alias=" + aliasEnum.nextElement());

      // Haal certificate
      System.out.println(ks.getCertificate("1"));

      // Haal certificate en private key
      KeyStore.PasswordProtection prot =
         new KeyStore.PasswordProtection("hallowereld".toCharArray());

      KeyStore.PrivateKeyEntry pkEntry =
         (KeyStore.PrivateKeyEntry)ks.getEntry("1", prot);

      System.out.println(pkEntry.getCertificate().getPublicKey());
      System.out.println(pkEntry.getPrivateKey());
```

Blind signing

```
   public void blind() {
      try {

         //********************** SETUP *********************************

         RSAPublicKey pubKey;
         RSAPrivateKey privKey;

         //generate the RSA key pair
         KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
         //initialise the KeyGenerator with a random number.
         keyGen.initialize(1024, new SecureRandom());
         KeyPair keypair = keyGen.genKeyPair();
         privKey = (RSAPrivateKey)keypair.getPrivate();
         pubKey = (RSAPublicKey)keypair.getPublic();
         System.out.println(privKey.toString());
```

```
        String message = "X";
        byte [] raw = message.getBytes("UTF8");

        BigInteger m = new BigInteger(raw);
        BigInteger e = pubKey.getPublicExponent();
        BigInteger d = privKey.getPrivateExponent();

        SecureRandom random = SecureRandom.getInstance("SHA1PRNG","SUN");
        byte [] randomBytes = new byte[10];
        BigInteger r = null;
        BigInteger n = pubKey.getModulus();
        BigInteger gcd = null;
        BigInteger one = new BigInteger("1");
        //check that gcd(r,n) = 1 && r < n && r > 1
        do {
           random.nextBytes(randomBytes);
           r = new BigInteger(randomBytes);
           gcd = r.gcd(n);
           System.out.println("gcd: " + gcd);
        }
        while(!gcd.equals(one) || r.compareTo(n)>=0 || r.compareTo(one)<=0);

        //******************** BLIND **********************************

        BigInteger b = ((r.modPow(e,n)).multiply(m)).mod(n);
        System.out.println("\nb = " + b);
        //must use modPow() - takes an eternity to compute:
        //b = ((r.pow(e.intValue)).multiply(m)).mod(n);

        //******************** SIGN **********************************

        BigInteger bs = b.modPow(d,n);
        System.out.println("bs = " + bs);

        //******************** UNBLIND **********************************

        BigInteger s = r.modInverse(n).multiply(bs).mod(n);
        System.out.println("s = " + s);

        //******************** VERIFY **********************************

        //signature of m should = (m^d) mod n
        BigInteger sig_of_m = m.modPow(d,n);
        System.out.println("sig_of_m = " + sig_of_m);

        //check that s is equal to a signature of m:
        System.out.println(s.equals(sig_of_m));

        //try to verify using the RSA formula
        BigInteger check = s.modPow(e,n);
        System.out.println(m.equals(check));

        //BOTH TESTS RETURN FALSE - s must not be a valid signature of m
    }
    catch(Exception ex) {
        System.out.println("ERROR: ");
        ex.printStackTrace();
    }
}
```

-- MarcEnschede - 04 Mar 2007