Redes de Computadores – Unioeste
Prof. Luiz Antonio

Aula Prática - Criptografia

1. Algoritmos MD5 e SHA-1

MD5

- hash de 128 bits = 16 bytes
- muito utilizado por softwares com protocolo P2P, verificação de integridade e logins.

SHA-1

- hash de 160 bits = 20 bytes
- Usado numa grande variedade de aplicações e protocolos de segurança, incluindo TLS, SSL, PGP, SSH, S/MIME e IPSec.
- É o algoritmo utilizado no eMule para identificar arquivos duplicados.

A API do Message Digest (`java.security.MessageDigest`)

Para que se possa gerar textos criptografados, é necessário seguir os seguintes passos.

1. Obter uma instância do algoritmo a ser usado.
2. Passar a informação que se deseja criptografar para o algoritmo.
3. Efetuar a criptografia.

Para se obter uma instância de um algoritmo de criptografia, utiliza-se o método `getInstance()` da classe `MessageDigest`.

```
MessageDigest md = MessageDigest.getInstance("MD5");
MessageDigest md = MessageDigest.getInstance("SHA-1");
```

Após a chamada a `getInstance()`, você possui uma referência a um objeto pronto para criptografar seus dados utilizando o algoritmo especificado. Neste caso o MD5.

Finalmente, para gerar a chave criptografada, você chama o método `digest()`.
As assinaturas existentes são:

```
byte[] digest();
byte[] digest(byte[] input);
int digest(byte[] buf, int offset, int len) throws DigestException;
```

Código completo

```
public static String md5 (String valor) throws Exception {
   MessageDigest md = MessageDigest.getInstance("MD5");
   BigInteger hash = new BigInteger(1, md.digest(valor.getBytes()));
   String s = hash.toString(16);
   if (s.length() %2 != 0)
      s = "0" + s;
      return s;
   }
}
```

2. Assinaturas Digitais

Assinaturas digitais servem para autenticar o dado sendo transmitido. Junto com o dado, uma assinatura digital é enviada. Comparada a uma chave pública, a assinatura é validada ou rejeitada.

A assinatura possui duas propriedades:

- Dada a chave pública correspondente a chave privada usada para gerar a assinatura, é possível verificar a autenticidade e integridade dos dados sendo transmitidos.
- A assinatura e a chave pública nada revelam sobre a chave privada.

A classe responsável no java por gerar as assinaturas digitais é chamada, não coincidentemente, de **`Signature`**.

Objetos signature são criados atavés da chamada ao método da classe `Signature` chamado.

```
static Signature getInstance(String algorithm)
```
Exemplo: `Signature sig = Signature.getInstance("DSA");`

A geração das chaves é possível através da classe `KeyPairGenerator`. Assim como a classe `Signature`, a `KeyPairGenerator` necessita de um algoritmo para gerar as chaves.

```
static KeyPairGenerator getInstance(String algorithm)
```
Exemplo: `KeyPairGenerator kpg = KeyPairGenerator.getInstance("DSA");`

Note que tanto as chaves quanto a assinatura devem ser geradas utilizando-se o mesmo algoritmo de criptografia. Após a obtenção de um `KeyPairGenerator`, devemos inicializa-lo através do método `initialize()`.

```
void initialize(int keysize, SecureRandom random)
```

Este método requer dois parâmetros: um tamanho de chave e um número aleatório.

A classe responsável por nos fornecer esse numero aleatório é chamada de `SecureRandom`. Ela gera númeors aleatórios que dificilmente se repetirão. O tamanho da chave deve ser compatível com o algoritmo sendo usado. no caso do DSA, podemos usar um tamanho de 512.

```
KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA");
SecureRandom secRan = new SecureRandom();
keyGen.initialize(512, secRan);
KeyPair keyP = keyGen.generateKeyPair();
PublicKey pubKey = keyP.getPublic();
PrivateKey priKey = keyP.getPrivate();
```
Após inicializada, a `KeyPairGenerator` está pronta para gerar nossas chaves pública e privada.

De posse de nossa chave privada, podemos utilizá-la para inicializar nosso objeto signature:

```
sig.initSign(priKey);
```

A partir de agora, podemos utilizar o método `update()` para passar ao algoritmo os dados a serem criptografados. Após o dado ser fornecido, o método sign deve ser chamado para geração da assinatura.
Nota: o método sign reseta o status do algoritmo ao seu estado inicial.

```
String mensagem = "The quick brown fox jumps over the lazy dog";
//Gerar assinatura
sign.update(mensagem.getBytes());
byte[] assinatura = sign.sign();
```

Aqui terminam as responsabilidades do remetente. Tudo o que ele precisa fazer agora é fornecer a chave pública juntamente com o dado a ser enviado e a assinatura correspondente.

O destinatário dos dados sendo enviados deverá receber, juntamente, a assinatura digital e ter acesso a chave pública.

Ele então poderá validar a assinatura junto ao dado recebido utilizando sua chave pública.

```
Signature clientSig = Signature.getInstance("DSA");
clientSig.initVerify(pubKey);
clientSig.update(mensagem.getBytes());

if (clientSig.verify(assinatura)) {
     //Mensagem assinada corretamente
} else {
     //Mensagem não pode ser validada
}
```

Hashes MD5

```
public class Md5 {
   public  static String md5 (String valor) throws Exception {
       MessageDigest md = MessageDigest.getInstance("MD5");
       //MessageDigest md = MessageDigest.getInstance("SHA-1");
       System.out.print(md.getDigestLength());
       BigInteger hash = new BigInteger(1, md.digest(valor.getBytes()));
       String s = hash.toString(16);
       if (s.length() %2 != 0)
           s = "0" + s;
    return  s;
   }

 public  static void main(String args[]) {
       try {
          String chave = md5("The quick brown fox jumps over the lazy dog");
          System.out.println(": "+chave);
          String chave2 = md5("The quick brown fox jumps over the lazy cog");
          System.out.println(": "+chave2);
    } catch (Exception e) {
          e.printStackTrace();
       }
 }
}
```

Geração de Assinaturas com Chave publica e privada – DSA

```
public static void main(String args[]) {
 try  {
       KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA");
       SecureRandom secRan = new SecureRandom();
       keyGen.initialize(512, secRan);
       KeyPair keyP = keyGen.generateKeyPair();
       PublicKey pubKey = keyP.getPublic();
       PrivateKey priKey = keyP.getPrivate();
       System.out.println("publica: "+pubKey);
       System.out.println("privada: "+priKey);
       //Obtem algoritmo para geração da assinatura
       Signature geradorAss = Signature.getInstance("DSA");

       //Inicializar geração
       geradorAss.initSign(priKey);
       String mensagem = "The quick brown fox jumps over the lazy dog";

       //Gerar assinatura
       geradorAss.update(mensagem.getBytes());
       byte[] assinatura = geradorAss.sign();
       //Grava a mensagem num arquivo properties
       Properties p = new Properties();
       p.put("mensagem", mensagem);
       p.put("assinatura", byteArrayToHexString(assinatura));
       p.store(new FileOutputStream("dado.properties"), null);

       //Serializa a chave pública
       ObjectOutputStream oout = new ObjectOutputStream(
                    new FileOutputStream("pubkey.ser"));
       oout.writeObject(pubKey);
       oout.close();
 }
}
```

2. Algoritmos RSA e DES

```
import java.io.*;
import java.security.*;
import javax.crypto.*;
import sun.misc.BASE64Decoder;
import sun.misc.BASE64Encoder;

public class RSA  {  /criptografia RSA
	protected static final String ALGORITMO = "RSA";

	public static void main(String args[]) {
		try {
			KeyPair chaves = generateKey();
			String texto = "Luiz";

			String t1 = encrypt(texto,chaves.getPublic());
			System.out.println("encrypt: "+t1);
			String t2 = decrypt(t1,chaves.getPrivate());
			System.out.println("decrypt: "+t2);
		} catch (NoSuchAlgorithmException e) {
			e.printStackTrace();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}

	/**
	 * Gera o par de chaves publica e privada usando 1024 bytes
	 * @return par de chaves
	 * @throws NoSuchAlgorithmException
	 */
	public static KeyPair generateKey() throws NoSuchAlgorithmException
	{
		KeyPairGenerator keyGen = KeyPairGenerator.getInstance(ALGORITMO);
		keyGen.initialize(1024);
		KeyPair key = keyGen.generateKeyPair();
		return key;
	}

	/**
	 * tEncripta um texto usando a chave publica e converte para Base64
	 * @param texto original
	 * @param chave publica
	 * @return Eteto encriptado em BASE64
	 * @throws java.lang.Exception
	 */
	public static String encrypt(String text, PublicKey key)
	throws Exception {
		String encryptedText=null;
		try  {
			byte[] cipherText = text.getBytes("UTF8");
			// get an RSA cipher object and print the provider
			Cipher cipher = Cipher.getInstance("RSA");
			// encrypt the plaintext using the public key
			cipher.init(Cipher.ENCRYPT_MODE, key);
			cipherText = cipher.doFinal(cipherText);
			encryptedText = encodeBASE64(cipherText);
		} catch (Exception e) {
			e.printStackTrace();
		}
		return encryptedText;
	}
```

```java
/**
	 * Decripta texto codificado em BASE64 encoded usando a chave privada
	 * @param text: exto encriptado, codificado em BASE64
	 * @param key: chave privada
	 * @return texto decriptado em UTF8
	 * @throws java.lang.Exception
	 */
	public static String decrypt(String text, PrivateKey key) throws Exception {
		String result=null;
		try  {
			// decodifica o texto base64
			byte[] dectyptedText = decodeBASE64(text);
			// decripta o texto usando a chave privada
			Cipher cipher = Cipher.getInstance("RSA");
			cipher.init(Cipher.DECRYPT_MODE, key);
			dectyptedText = cipher.doFinal(dectyptedText);
			result = new String(dectyptedText, "UTF8");
		}  catch (Exception e) {
			e.printStackTrace();
		}
		return result;

	}

	/**
	 * codifica o array de bytes para uma string base64
	 * @param bytes: array de bytes
	 * @return String codificada
	 */
	private static String encodeBASE64(byte[] bytes) {
		BASE64Encoder b64 = new BASE64Encoder();
		return b64.encode(bytes);
	}

	/**
	 * Decodifica uma string base54 para um array de bytes
	 * @param text: a string codificada
	 * @return array de bytes
	 * @throws IOException
	 */
	private static byte[] decodeBASE64(String text) throws IOException {
		BASE64Decoder b64 = new BASE64Decoder();
		return b64.decodeBuffer(text);
	}
}
```

Algoritmo DES – Chaves simétricas

```
public class DES
{
	public static void main(String args[]) {
		try {
			// Gerador de chaves com algoritmo DES
			KeyGenerator desGen = KeyGenerator.getInstance("DES");

			SecretKey desKey = desGen.generateKey();
			System.out.println(encodeBASE64(desKey.getEncoded()));
			String plaintext = "Luiz";  // Dados para criptografia

			// Objeto para criptografar/decriptografar
			Cipher cipher = Cipher.getInstance("DES");
			// Inicializa o cipher para criptografia
			cipher.init(Cipher.ENCRYPT_MODE, desKey);
			// Criptografa os dados
			byte[] ciphertext = cipher.doFinal(plaintext.getBytes());
			System.out.println(encodeBASE64(ciphertext));
			cipher.init(Cipher.DECRYPT_MODE, desKey);
			byte[] decryptedMessage = cipher.doFinal(ciphertext);
			System.out.println(new String(decryptedMessage));

		} catch (NoSuchAlgorithmException e) {
			e.printStackTrace();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
```