UNIVERSIDADE DE COIMBRA

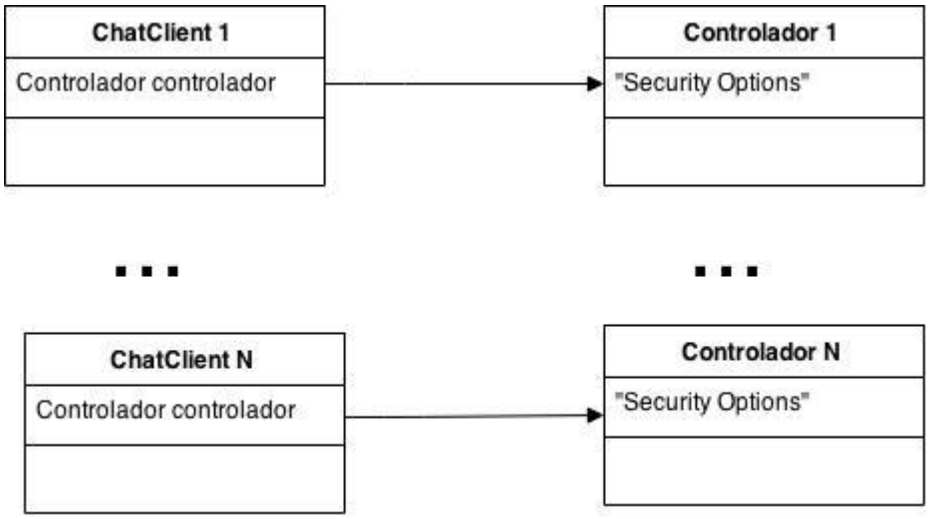
Relatório STI

Trabalho Prático III

Erbi Silva e João Batista

02/05/2015

Classe “Controlador”



O sistema programado lança diversos clientes (“ChatClient”). Cada cliente terá um controlador que será responsável por controlar todas as acções que esse cliente precisa tais como, encriptação das mensagens, , manter a integridade das mensagens, entre outras, basicamente controla toda a segurança do cliente.

Classe “Mensagem”

A classe mensagem foi criada para transportar a mensagem encriptada, a chave, o ID de quem a envia e a “hash” para controlar se a mensagem foi ou não alterada pelo caminho.

Confidencialidade

A confidencialidade é garantida com encriptação e desencriptação das mensagens. Após o cliente escrever a mensagem, o controlador irá ser responsável por encriptar e enviar a mensagem do cliente. Para isso, cada cliente tem uma chave que é renovada de X em X tempo para garantir mais segurança ao utilizador. Assim sendo, são utilizados os seguintes algoritmos:

Gerar a chave:	AES
Encriptar:	AES/CBC/PKCS5Padding
Desencriptar:	AES/CBC/PKCS5Padding
Tamanho:	16 bytes

Existe uma classe Mensagem que é responsável por transportar a mensagem encriptada bem como a chave de quem envia a mensagem.

A chave é criada com o seguinte algoritmo:

```
public void generateKey() throws NoSuchAlgorithmException{
  KeyGenerator kg = KeyGenerator.getInstance("AES");
  SecureRandom random = new SecureRandom();
  kg.init(random);
  key = kg.generateKey();
}
```

Algoritmo Encriptação

O algoritmo de encriptação que utiliza a chave acima gerada é o seguinte:

```
public Mensagem encrypt(final String message, int ID) throws IllegalBlockSizeException,
BadPaddingException, NoSuchAlgorithmException,
NoSuchPaddingException, InvalidKeyException,
UnsupportedEncodingException, InvalidAlgorithmParameterException {
  Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
  cipher.init(Cipher.ENCRYPT_MODE, key, iv);

  byte[] stringBytes = message.getBytes();

  byte[] raw = cipher.doFinal(stringBytes);
  Mensagem m = new Mensagem(Base64.getEncoder().encodeToString(raw), key.getEncoded(), iv.getIV(), ID);
  return m;
}
```

Algoritmo Desencriptação

O algoritmo de desencriptação que utiliza a chave acima gerada é o seguinte:

```
public Mensagem decrypt(final String encrypted, final byte[] chave, final byte[] param, int ID) thr
NoSuchAlgorithmException, NoSuchPaddingException,
IllegalBlockSizeException, BadPaddingException, IOException, InvalidAlgorithmParameterException {

  Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
  IvParameterSpec ivSpec=new IvParameterSpec(param);
  SecretKey keySpec=new SecretKeySpec(chave,"AES");
  cipher.init(Cipher.DECRYPT_MODE, keySpec, ivSpec);
  byte[] raw = Base64.getDecoder().decode(encrypted);
  byte[] stringBytes  = cipher.doFinal(raw);
  String clearText = new String(stringBytes, "UTF8");
  Mensagem m = new Mensagem(clearText, keySpec.getEncoded(), ivSpec.getIV(), ID);
  return m;
}
```

Autenticidade/Integridade

Para garantir estes mecanismos de segurança, a mensagem guarda uma hash que é criada logo após o cliente envia a mensagem. Esta hash é gerada de acordo com o corpo da mensagem de quem a envia para garantir toda esta segurança. O algoritmo que cria a hash é:

```
public void generateMD5(String message){
    MessageDigest md;
    try {
        md = MessageDigest.getInstance("MD5");
        md.update(message.getBytes(), 0, message.length());
        bi = new BigInteger(1, md.digest());
    } catch (NoSuchAlgorithmException ex) {
        System.out.println("Erro ao criar MD5");
    }
}
```

Manutenção da chave

Para garantir a manutenção da chave, esta é renovada de X em X tempo graças a uma thread que o servidor lança. Essa irá apagar todas as chaves anteriores, gerar novas, guardar as mesmas e é responsável por enviar uma mensagem a cada cliente indicando-lhe a sua nova chave.

```
public class RenewKey implements Runnable{
    ChatServer c;

    public RenewKey(ChatServer c){
        this.c = c;
    }
        @Override
        public void run() {
            while(c != null){
                try {
                    Thread.sleep(TIME_TO_REGENERATE_KEY);
                    c.renovaChave();
                } catch (InterruptedException ex) {
                }
            }
        }
```

Non-repudiation

Para garantir "non-repudiation" cada cliente tem um par de chaves. O cliente encripta a sua mensagem com a sua chave privada e, o receptor receberá a chave pública para desencriptar a mesma.

Assinatura para encriptar:

```
try {
    encriptado = conf.encrypt(mensagem, 0, VERBOSE);
    Signature mySign = Signature.getInstance("MD5withRSA");
    KeyFactory kf = KeyFactory.getInstance("RSA");
    byte[] privada = mySignature.getPrivate();

    EncodedKeySpec privateKeySpec = new PKCS8EncodedKeySpec(privada);
    PrivateKey myPrivateKey = kf.generatePrivate(privateKeySpec);
    mySign.initSign(myPrivateKey);
    mySign.update(mensagem.getBytes());
    byte[] byteSignedData = mySign.sign();
    encriptado.setPrivateKey(privada);
    encriptado.setSignature(byteSignedData);
} catch (IllegalBlockSizeException | BadPaddingException | NoSuchAlgorithmExcep
    System.out.println("Erro: " + ex);
} catch (InvalidKeySpecException | SignatureException ex) {
    Logger.getLogger(Controlador.class.getName()).log(Level.SEVERE, null, ex);
```

Com este pedaço de código o utilizador instancia uma "Signature" e cria uma fábrica de chaves. Isto para que se possa ir buscar a chave privada e, criar uma assinatura com a chave privada sobre a mensagem deste utilizador.

Assinatura para desencriptar:

```
Signature myVerifySign = Signature.getInstance("MD5withRSA");
byte[] chavePublica = conf.decryptByte(m.getPublicKey(), desencriptado.iv);
EncodedKeySpec publicKeySpec = new X509EncodedKeySpec(chavePublica);
KeyFactory keyFactory = KeyFactory.getInstance("RSA");
PublicKey newPublicKey = keyFactory.generatePublic(publicKeySpec);
myVerifySign.initVerify(newPublicKey);
if(ALTERA_MENSAGEM_ASSINATURA)
    myVerifySign.update("...!!!!!".getBytes());
else
    myVerifySign.update(desencriptado.getMensagem().getBytes());
byte[] byteSignedData = m.getSignature();
boolean verifySign = myVerifySign.verify(byteSignedData);
if (verifySign == false){
    System.out.println("Error in validating Signature ");
    return null;
}
else
    System.out.println("Successfully validated Signature ");
```

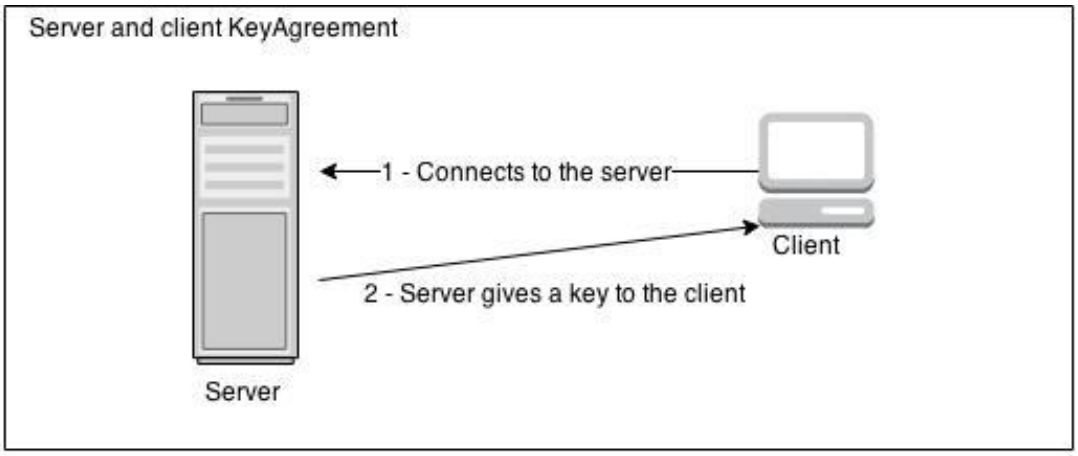
Aqui podemos ver que o código faz praticamente o mesmo mas, vai buscar a chave pública para verificar se combina com a assinatura.

Modelos Segurança

Neste capítulo vão ser descritos e ilustrados os modelos de segurança utilizados:

Criação de chaves

Sempre que um cliente se liga ao servidor, este faz um acordo de chaves com cada cliente. Isto é, o servidor é responsável por dar uma chave que ficou acordada entre ele e o cliente. Assim sendo, cada cliente terá uma chave para encriptar e, o servidor quando receber uma mensagem de determinado cliente, saberá sempre o que fazer com a mesma.



No exemplo do nosso projecto, o cliente encripta a mensagem com a sua chave e envia a mensagem. À chegada do servidor, o mesmo reconhece quem envia a mensagem e, antes de enviar para o destinatário, vai encriptar a chave de remetente com a chave do destinatário. Assim, caso a mensagem seja apanhada a meio, a chave estará sempre segura.

Note-se que o servidor guarda todas as chaves mas, também as encripta com uma chave própria do servidor.

Renovação das chaves

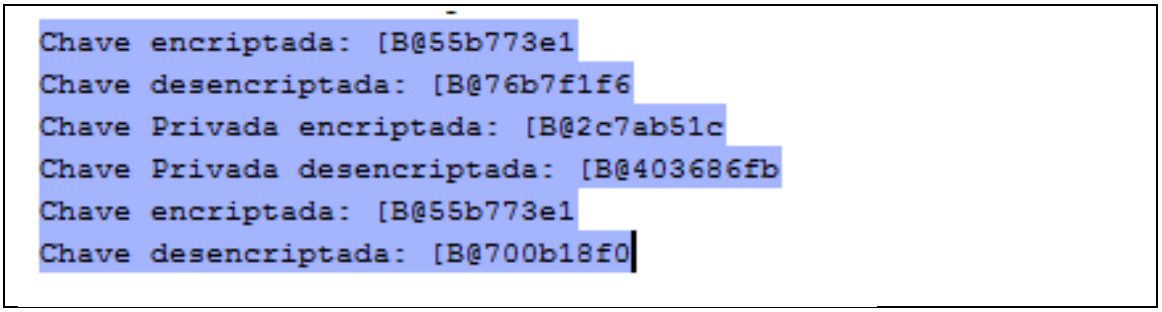
A renovação das chaves é feita de X em X tempo pelo servidor. Este é responsável por criar uma mensagem que será enviada para cada cliente. Estes, ao identificarem essa mensagem, receberão os dados da nova chave e irão actualizar os dados de si próprios.

Testes

Para garantir que todas as funcionalidades estão bem implementadas, foi criada uma classe "Ataques" que é responsável, por exemplo, de alterar o conteúdo de uma mensagem.

Chaves encriptadas no servidor

Com isto, o primeiro teste é para provar como o servidor guarda as chaves encriptadas, pelo que mostramos o seguinte exemplo com um servidor e um cliente:



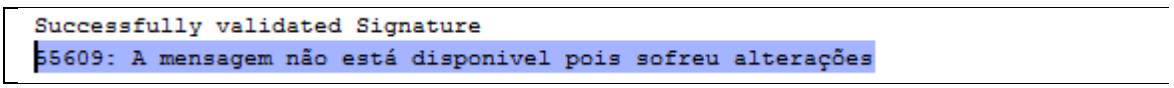
O servidor recebeu uma mensagem do cliente. Então, o servidor pegou na chave encriptada (1) e desencriptou a mesma (2). Como vemos os resultados são diferentes. O mesmo acontece com privadas e públicas.

Alterar a mensagem pelo caminho e comprovar que é diferente da original

No controlador, basta colocar o false a true:

```
private static Boolean ALTERA_MENSAGEM_PARA_MD5 = true;
```

De seguida correr o cliente e enviar uma mensagem. O resultado é o seguinte:



NOTA: Aparece que a assinatura é válida uma vez que nesta situação a mensagem só está a ser alterada, depois e só depois de verificar a assinatura.

Alterar assinatura da mensagem

No controlador, basta colocar o false a true:

```
private static Boolean ALTERA_MENSAGEM_ASSINATURA = true;
```

De seguida correr o cliente e enviar uma mensagem. O resultado é o seguinte:

```
Error in validating Signature
```

Alterar chave da mensagem a meio

No controlador, basta colocar o false a true:

```
private static Boolean ALTERA_MENSAGEM_PARA_MD5 = true;
```

De seguida correr o cliente e enviar uma mensagem. O resultado é o seguinte:

```
Successfully validated Signature
55665: A mensagem não está disponivel pois sofreu alterações
```

NOTA: Aparece que a assinatura é válida uma vez que nesta situação a chave só está a ser alterada, depois e só depois de verificar a assinatura.