Métodos Criptográficos em Java

Ana Karina Dourado Salina de Oliveira[1]
João Paulo Delgado Preti[2]

Resumo: Este trabalho tem como objetivo descrever as implementações em linguagem Java dos principais algoritmos de criptografia utilizados atualmente nos sistemas para internet. Estes algoritmos têm como função proteger o acesso à informação de pessoas não autorizadas. A criptografia simétrica e assimétrica provém mecanismos de segurança para garantir: que os sistemas que manipulam os dados sejam confiáveis; a veracidade das informações; e a autenticação das pessoas, através da assinatura digital. O Java possui vários pacotes de funções prontas para facilitar a implantação de segurança nos sistemas para internet, que podem ser utilizados pelos desenvolvedores de aplicações para internet.

Palavras-chave: Segurança em Java, criptografia, funções *hash*, chaves.

Abstract: This paper aims to describe the implementations in Java Language of the key algorithms. The encryption algorithm currently used in systems for the internet is also described. These algorithms have as a designed function to protect the information access for unauthorized users. The symmetric and asymmetric encryption offer security mechanisms to guarantee the systems that handle the data are reliable, ensuring the accuracy of information and also ensuring, through digital signature, the people authentication. The Language Java has several functions packages ready to facilitate the use of security systems for the internet, which might be used by the developers of internet applications.

Keywords: Security into Java, cryptography, functions hash, keys.

1 Mestra em Ciência da Computação, pela Universidade Federal de Santa Catarina (UFSC); professora do Deptº de Informática do Cefet-MT. E-mail: anakarina@inf.cefetmt.br.

2 Mestre em Ciência da Computação, pela Universidade Federal de Santa Catarina (UFSC); professor do Deptº de Informática do Cefet-MT. E-mail: joaopaulo@inf.cefetmt.br.

Introdução

A criptografia vem sendo amplamente utilizada nos sistemas para internet devido à necessidade de proteger os dados contidos nos bancos de dados das empresas.

Algoritmos de criptografia estão se tornando cada vez mais eficientes, visando oferecer o máximo de proteção aos sistemas que o utilizam. Hoje, sabemos que o uso da senha para restringir o acesso aos dados somente a pessoas autorizadas é de fundamental importância e sua implementação de forma segura é alvo de bastante pesquisa.

A ética em Computação vem sendo incluída nas grades dos cursos de Informática buscando conscientizar as pessoas sobre as regras que devem ser seguidas quando se buscam dados na internet.

Sistemas na plataforma Java têm crescido muito, dada a simplicidade da linguagem, por ela ser orientada a objetos, oferecer segurança, portabilidade e recursos para a programação distribuída. Esta linguagem já faz parte de uma grande quantidade de sistemas para internet e a implementação de segurança é de essencial importância para manter a confiabilidade da empresa que oferece os serviços por esse meio.

De acordo com Deitel (2005), "o Java impõe uma forte segurança para se certificar de que os programas Java que chegaram pela rede não danificam os arquivos ou sistemas (como vírus e vermes de computador)".

Neste artigo, serão definidos os principais conceitos de segurança utilizados nas aplicações Java, seguidos dos tipos de criptografia existentes atualmente, as gerações de senhas, as funções *hash* e as assinaturas digitais. Posteriormente, serão apresentados alguns tipos de implementações de segurança em Java.

Conceitos de Segurança

Conforme Basham e Sierra (2005), para que a segurança de um Sistema em Java não tenha falhas, é preciso que se implementem as seguintes questões relativas à segurança:

Integridade – garante que a mensagem recebida seja a mesma que foi enviada, sem ter sido alterada durante o caminho por pessoas mal-intencionadas.

Confidencialidade – disponibiliza os dados somente a pessoas autorizadas.

Autenticação – prova que a pessoa é quem diz ser, com a utilização da assinatura digital.

Autorização – certifica que a pessoa está autorizada a ver determinados dados restritos ao próprio usuário.

Todos estes conceitos são implementados com a utilização de funções *Hash*, e criptografias simétrica e assimétrica, onde são utilizados diferentes tipos de chaves para cada serviço de segurança.

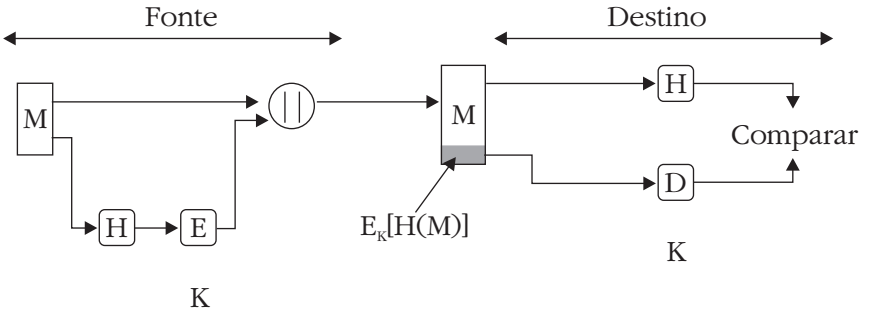
Funções *Hash*

Esta técnica permite que, ao ser aplicada a uma mensagem de qualquer tamanho, seja gerado um resumo criptografado de tamanho fixo e bastante pequeno como, por exemplo, 128 bits. Com a função *Hash*, pode-se garantir que o conteúdo de uma mensagem não foi alterado, que ela está íntegra (CUSTODIO, 2000).

A idéia básica é que o resultado *hash* forneça uma identidade única para uma mensagem e que a proteção de uma pequena identidade seja mais facilmente obtida do que a proteção da mensagem como um todo. Este tipo de algoritmo não serve para cifrar conteúdo.

A Figura 1 ilustra como é feito para gerar autenticação utilizando a função *Hash* (H), aplicada na mensagem (M); o *hash* gerado é então cifrado com uma chave simétrica (K). Este esquema provê autenticação, pois H(M) é cifrado com a K que a fonte (A) e o destinatário (B) compartilham.

Figura 1. *Hash*.



Esses códigos *hash* são extremamente úteis para a segurança de senhas. Como não podem ser descriptografados, precisam ser regerados e comparados com a seqüência disponível anteriormente. Se a seqüência for equivalente, significa que a senha está correta e o acesso aos módulos do sistema é liberado.

Geração de Senhas

A senha (chave) é um dos principais itens ligados à segurança. Através da senha, cada usuário garante a proteção e o acesso às suas informações. Os algoritmos de segurança utilizam chaves para cifrar e decifrar os dados e a segurança do sistema passa a residir não mais no algoritmo e sim na chave. Desta forma, a geração destas chaves é de fundamental importância para melhorar a segurança dos algoritmos.

Muitas pesquisas envolvem a melhor forma de geração das chaves, para que estas se tornem impossíveis de serem decifradas. Quanto maior a chave, mais difícil será para decifrá-la. Porém, o tamanho não é o único ponto levado em consideração na geração das chaves. Muitos cálculos matemáticos são utilizados para garantir a sua segurança e, conseqüentemente, a dos dados.

Os números randômicos são muito utilizados nos algoritmos que geram chaves, por produzir uma seqüência de números aleatórios, melhorando o embaralhamento das informações que são cifradas utilizando estas chaves.

Criptografia

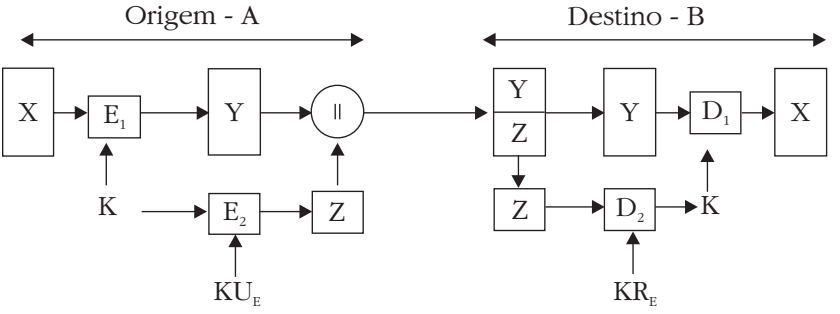
A criptografia tem como função tornar um texto original ilegível para pessoas não autorizadas (criptoanalistas). Quando se cifra um texto original x, utilizando uma chave (K), obtém-se um texto ilegível (texto cifrado) y, onde fK(x) = y. Para recuperar o texto original a partir do texto cifrado, usa-se o algoritmo inverso para decifrá-lo, onde f - 1K(y), conforme Terada (2000).

A Criptografia possui duas formas: Simétrica e Assimétrica. Na Criptografia Simétrica, utiliza-se a mesma chave (K) para cifrar (E) e decifrar (D). A Criptografia Assimétrica utiliza um par de chaves, uma pública (KU) e uma privada (KR). O que for cifrado com uma das chaves somente poderá ser decifrado com a outra. A chave pública é disponível para todos, já a chave privada somente o proprietário a conhece.

O DES, o *TripleDES* e o *Blowfish* são algoritmos de criptografia simétrica utilizados para cifrar grande quantidade de dados. O RSA é um algoritmo de criptografia assimétrica utilizado para a troca de chaves e autenticação de mensagens.

Cifrar uma grande quantidade de informações usando chave assimétrica não é muito eficiente, apesar de mais seguro, porque são algoritmos mais lentos. O mais recomendado é cifrar os dados usando um algoritmo (E1) de chave simétrica e, depois, cifrar a chave simétrica (K) usando um algoritmo (E2) de chave assimétrica com a chave pública do Destinatário (KUB). Depois, empacota-se tudo e envia-se para o receptor, que utilizará o algoritmo Assimétrico (D2) com a chave privada de B (KRB) para obter a chave simétrica (K); depois, utiliza-se o Algoritmo Simétrico (D1) para decifrar a informação, conforme mostra a Figura 2.

Figura 2. Criptografia simétrica e assimétrica.



Assinaturas Digitais

A Assinatura Digital é um método de autenticação de informação digital. Sua utilização providencia a prova inegável de que uma mensagem veio do emissor. Para verificar este requisito, uma assinatura digital deve ter as seguintes propriedades: autenticação, integridade e não repúdio (o emissor não pode negar a sua autenticidade).

As assinaturas digitais servem para autenticar o remetente da informação e garantir que o dado é confiável. Trabalham com uma tríade: a assinatura em si, uma chave pública e uma chave privada.

É de responsabilidade do remetente gerar a tríade e fornecer a chave pública aos destinatários de suas mensagens. No ato do envio, o remetente gera uma assinatura para o dado que deseja enviar, usando a chave privada. O destinatário recebe o dado e a assinatura, e valida a informação usando sua chave pública.

Se a validação for efetuada com sucesso, o destinatário tem a garantia de que a mensagem foi enviada por um remetente confiável, possuidor da chave privada.

Implementações

Algumas implementações desses métodos de segurança mais utilizados em Java serão mostradas a seguir.

Funções *Hash*: MD5 e SHA1

O Java possui uma API que implementa os algoritmos *hashing* de *Message Digest*: o MD5 e o SHA-1. Para a utilização dessas funções, 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; e
3. Realizar a criptografia.

Para se obter a instância de um algoritmo de criptografia *Message Digest*, utiliza-se o método *getInstance*() da classe *Message Digest*.

```
1 MessageDigest md = MessageDigest.getInstance
  algoritmo).
```

O parâmetro algoritmo pode ser o MD5 ou o SHA1.

Após a chamada à *getInstance*(), é obtida a referência a um objeto pronto para criptografar os dados utilizando o algoritmo especificado.

Quando se obtém a instância de um algoritmo, ela estará pronta para ser utilizada. Em seqüência, executa-se o método *update*() para passar os dados a serem criptografados.

```
1 //Faz o update do digest utilizando o byte
  especificado;
2 // Este método é útil quando não se tem controle do
  tamanho da mensagem a ser criptografada;
3 //(por exemplo, vinda de um stream)
4
5 void update(byte input);
6 //Exemplo
7 int i;
8
9 while((i = inputStream.read())!=              -1) {
10     md5.update((byte)i);
11     }
```

O método *reset*() pode ser executado para que o algoritmo volte ao estado original (antes de ter sido chamado qualquer *update*()).

Para gerar a chave criptografada, executa-se o método *digest*(). Eis as assinaturas existentes:

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

O primeiro método realiza a operação nos *bytes* que foram fornecidos até o momento (através do método *update()*).

O segundo método realiza um *update()* final, utilizando o *array* de *bytes* fornecido, e completa a operação.

O terceiro método armazena em *buf* o resultado do *hashing*. O *offset* e o *length* especificam onde, no *array* de destino, o *hashing* deve ser colocado. Este método retorna a quantidade de *bytes* escrita em *buf*.

Após a execução desses métodos, chama-se o método *reset*() para devolver o algoritmo a seu estado inicial.

Geração de Chaves

O Java possui um utilitário *keytool* que permite criar e gerenciar chaves e certificados de segurança. A informação de autenticação gerenciada pela ferramenta envolve uma cadeia de certificados X.509 e uma chave privada associada, referenciada por uma *alias*.

Para gerar uma chave com o utilitário *keytool*, tem-se a opção –*genkey*. As opções normalmente fornecidas a este comando são –*alias*, que indicam o nome do *alias* a ser associado à chave, e –*keystore*, que indica o nome do banco de dados de *keystore* onde a chave será armazenada.

A classe *KeyGenerator* gera uma única chave, utilizada geralmente para chaves simétricas.

A classe *KeyPairGenerator* é utilizada para a geração do par de chaves (pública e privada) que, por sua vez, é utilizada para a geração de chaves assimétricas. Assim como a classe *Signature*, a *KeyPairGenerator* necessita de um algoritmo assimétrico para gerar as chaves.

```
1 static KeyPairGenerator getInstance(String
  algorithm)
2 //Exemplo
3 KeyPairGenerator kpg = KeyPairGenerator.get
  Instance(DSA)
```

Tanto as chaves quanto a assinatura digital são geradas utilizando-se o mesmo algoritmo de criptografia DSA.

Após a obtenção de um par de chaves, faz-se a inicialização através do método *initialize()*.

```
1 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 números aleatórios que dificilmente se repetirão.

O tamanho da chave deve ser compatível com o algoritmo sendo usado. No caso do DSA, pode-se usar um tamanho de 512.

Após a inicialização, a *KeyPairGenerator* está pronta para gerar as chaves pública e privada.

```
1 KeyPairGenerator keyGen = KeyPairGenerator.get
  Instance (DSA);
2 SecureRandom secRan = new SecureRandom();
3 keyGen.initialize(512, secRan);
4 KeyPair keyP = keyGen.generateKeyPair();
5 PublicKey pubKey = keyP.getPublic();
6 PrivateKey priKey = keyP.getPrivate().
```

Criptografia Simétrica

A seguir, será mostrada a implementação para algoritmos de criptografia simétrica, como o DES e *Blowfish*. Para utilizar os algoritmos de criptografia simétrica, é necessário baixar a API para estas implementações de segurança na página do GNU.

Para cifrar textos utilizando o algoritmo DES, é necessário definir uma chave simétrica (*key*) e criar uma instância do algorimo DES, que será utilizada para cifrar (*des.Encode*) e decifrar (*des.Decode*).

```
1. String key = "keyword_xpto";
2. DES des = new DES(key);
3. String des_encode = new String(des.Encode(«text to
   encode...»));
4. System.out.println(des_encode);
5. String des_decode = des.Decode(des_encode. getBytes());
6. System.out.println(des_decode) (VELASQUES, 2007).
```

Abaixo, segue um exemplo de criptografia de dados, onde é utilizado o algoritmo *Blowfish*. Será criado um objeto *Cipher*, uma chave e depois a chave será usada para cifrar os dados. O programa usa, então, a mesma chave para cifrar os dados (TAYLOR, BUEGES e LAYMAN, 2003).

```
1  //Apanha o gerador de chave e cria uma chave;
2
3  keyGenerator kgen = keyGenerator.getInstance (Blo-
   wfish);
4  SecretyKey secretyKey = Kgen.generateKey();
5  byte[] bytes = secretKey.getEncoded();
6  SecretKeySpec specKey = new SecretKeySpec
   (bytes,Blowfish);
7
8  //Cria o objeto Cipher;
```

```
9  Cipher cipher = Cipher.getInstance(Blowfish);
10 cipher.init(Cipher.ENCRYPT_MODE, specKey);
11 String target = "Ana Karina";
12 byte[]  encrypted  =  cipher.doFinal(target.  ge-
   tBytes());
13
14 //Decifrando...
15 cipher.init(Cipher.DECRYPT_MODE, specKey);
16 byte[] decrypted = cifer.doFinal(encrypted).
```

Assinaturas Digitais

No Java, a classe responsável por gerar as assinaturas digitais é chamada de *Signature*. Objetos *signature* são criados atavés da chamada ao método da classe *Signature*.

O Algoritmo Assimétrico utilizado nesta implementação é o DSA (algoritmo de assinatura digital). Com a chave privada, é possível utilizá-la para inicializar o objeto *signature*:

```
1 sig.initSign(priKey.
```

O método *update*() passará ao algoritmo os dados a serem criptografados. Após o dado ser fornecido, o método sign deve ser chamado para a geração da assinatura.

```
1 String mensagem = "Elvis is Live!!!";
2 //Gerar assinatura
3 sign.update(mensagem.getBytes());
4 byte[] assinatura = sign.sign().
```

O método *sign* reseta o status do algoritmo ao seu estado inicial. Assim, o remetente precisa enviar ao destinatário a chave pública, juntamente com o dado a ser enviado e a assinatura correspondente.

O destinatário deverá receber, juntamente, a assinatura digital e ter acesso à chave pública, então ele poderá validar a assinatura junto ao dado recebido utilizando sua chave pública.

```
1 Signature clientSig = Signature.getInstance DSA);
2 clientSig.initVerify(pubKey);
3 clientSig.update(mensagem.getBytes());
4
5 if (clientSig.verify(assinatura)) {
6 //Mensagem assinada corretamente
7 } else {
8 //Mensagem não pode ser validada
9 }
```

JAAS

O pacote Java Authentication and Authorization Service (JAAS) oferece uma API (*Application Program Interface*) de segurança, que permite que as aplicações Java tenham um controle de autenticação e autorização.

> O JAAS implementa uma versão Java do *framework* padrão *Pluggable Authentication Module* (PAM), e suporta autorização baseada em usuário, permitindo que as aplicações fiquem independentes desse controle de segurança. Serve para controlar permissões de vários tipos de recursos como: arquivos, diretórios, conteúdos, URLs (VIANA, 2007).

METODOLOGIA

Neste trabalho, foi utilizado o método analítico, comparando as diversas bibliografias com as descrições constantes em sítios. Logo após, foram analisadas as implementações dos principais algoritmos.

Conclusões

Hoje, a grande dependência das pessoas de sistemas computacionais torna a segurança das informações um fator essencial a ser implantado.

O estudo da criptografia vem tornando os algoritmos de segurança cada vez mais confiáveis, disponibilizando serviços como comércio eletrônico, bancos e correio eletrônico autenticados e criptografados.

Algoritmos tradicionais de criptografia simétrica e assimétrica já foram implementados em Java e são facilmente encontrados na internet.

O Java, por ser uma linguagem aberta, possui uma série de ferramentas gratuitas disponíveis, que poderão garantir a segurança das aplicações para internet. A decisão de qual das implementações utilizar está relacionada com o tipo de questão de segurança que se deseja obter do sistema, como: autenticação, autorização, integridade dos dados e confidencialidade.

Através desta pesquisa, foi possível descrever como utilizar os principais algoritmos de segurança disponíveis em linguagem Java.

Esses algoritmos foram descritos de forma a orientar qual dos pacotes disponíveis para a linguagem Java é mais adequado para a utilização no desenvolvimento de sistemas, conforme o nível de segurança desejado da informação.

Referências Bibliográficas

BASHAM, B.; SIERRA, K. *Use a cabeça!* Servlets & JSP. Rio de Janeiro: Ed. Alta Books, 2005.

CUSTODIO, Ricardo Felipe. *Segurança em computação*. Florianópolis: UFSC, 2000.

DEITEL, H. M. *Java*: como programar. São Paulo: Prentice Hall, 2005.

TAYLOR, Art; BUEGES, Buege; LAYMAN, Randy. *Segurança contra hackers J2EE e Java*. São Paulo: Ed. Futura, 2003.

TERADA, Routo. *Segurança de dados*. São Paulo: Ed. Edgard Blucher Ltda., 2000.

VELASQUES, Eduardo F. *Criptografia + Java + Descriptografar*. Disponível em: <http://www.guj.com.br/posts/list/51900.java>. Acesso em: 1 set. 2007.

VIANA, Fábio. *Tutorial de JAAS*. Disponível em: <http://www.guj.com.br/java.tutorial.artigo.184.1.guj>. Acesso em 14 set. 2007.