Criptografia e Hashing em Java - Um Guia Completo para Proteção de Dados com AES, RSA e SHA em Java 8, 11 e 17

A criptografia é um dos pilares da segurança da informação, garantindo que dados em trânsito e em repouso sejam protegidos contra acessos não autorizados.

Este artigo explora os principais tipos de criptografia, simétrica e assimétrica, e o uso de funções de hash para integridade de dados.

Abordaremos, com exemplos práticos, como implementar algoritmos de criptografia (AES e RSA) e hashing (MD5, SHA-1, SHA-256) em Java 8, 11 e 17, com foco em bibliotecas e boas práticas.

Tipos de Criptografia

Criptografia Simétrica

Na criptografia simétrica, uma única chave é usada para encriptar e decriptar os dados. Este método é eficiente e rápido, sendo ideal para grandes volumes de dados. Um exemplo comum é o AES (Advanced Encryption Standard).

Criptografia Assimétrica

Na criptografia assimétrica, usa-se um par de chaves: uma pública e outra privada. A chave pública encripta, enquanto a privada decripta os dados. O RSA (Rivest-Shamir-Adleman) é o algoritmo mais utilizado em criptografia assimétrica.

Implementação de Criptografia Simétrica com AES

AES em Java 8, 11 e 17

AES é amplamente utilizado para criptografia de dados em massa. O exemplo a seguir demonstra o uso de AES com a biblioteca javax.crypto:

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import java.util.Base64;

public class AESCryptographyExample {
    public static void main(String[] args) throws Exception {
        KeyGenerator keyGen = KeyGenerator.getInstance("AES");
        keyGen.init(128);
        SecretKey secretKey = keyGen.generateKey();

        String text = "Texto de exemplo para encriptar com AES";

        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encryptedBytes = cipher.doFinal(text.getBytes());

        String encryptedText = Base64.getEncoder().encodeToString(encryptedBytes);
        System.out.println("Texto encriptado: " + encryptedText);

        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(encryptedText));
        String decryptedText = new String(decryptedBytes);
        System.out.println("Texto decriptado: " + decryptedText);
    }
}
```

Implementação de Criptografia Assimétrica com RSA

RSA em Java 8, 11 e 17

O RSA é ideal para troca de chaves e assinaturas digitais, como no exemplo abaixo

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import javax.crypto.Cipher;
import java.util.Base64;

public class RSACryptographyExample {
    public static void main(String[] args) throws Exception {
        KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("RSA");
        keyPairGen.initialize(2048);
        KeyPair pair = keyPairGen.generateKeyPair();
        PublicKey publicKey = pair.getPublic();
        PrivateKey privateKey = pair.getPrivate();

        String text = "Texto para encriptar com RSA";
```

```
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);
        byte[] encryptedBytes = cipher.doFinal(text.getBytes());

        String encryptedText = Base64.getEncoder().encodeToString(encryptedBytes);
        System.out.println("Texto encriptado: " + encryptedText);

        cipher.init(Cipher.DECRYPT_MODE, privateKey);
        byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(encryptedText));
        String decryptedText = new String(decryptedBytes);
        System.out.println("Texto decriptado: " + decryptedText);
    }
}
```

Hashing em Java

Funções de hash transformam uma entrada de tamanho arbitrário em uma saída de tamanho fixo. Hashes são unidirecionais, sendo usados principalmente para verificar integridade de dados.

Exemplo de MD5 em Java 8, 11 e 17

MD5 gera um hash de 128 bits. Embora seja menos seguro, é útil para verificar a integridade de dados.

```
import java.security.MessageDigest;
import java.util.Base64;

public class MD5HashExample {
    public static void main(String[] args) throws Exception {
        String text = "Texto para gerar o hash MD5";

        MessageDigest md = MessageDigest.getInstance("MD5");
        byte[] hash = md.digest(text.getBytes());

        String hashBase64 = Base64.getEncoder().encodeToString(hash);
        System.out.println("Hash MD5 em Base64: " + hashBase64);
    }
}
```

Exemplo de SHA-256 em Java 8, 11 e 17

SHA-256 é mais seguro que o MD5 e gera um hash de 256 bits.

```
import java.security.MessageDigest;
import java.util.Base64;

public class SHA256HashExample {
    public static void main(String[] args) throws Exception {
        String text = "Texto para gerar o hash SHA-256";

        MessageDigest sha256 = MessageDigest.getInstance("SHA-256");
        byte[] hash = sha256.digest(text.getBytes());

        String hashBase64 = Base64.getEncoder().encodeToString(hash);
```

```
        System.out.println("Hash SHA-256 em Base64: " + hashBase64);
    }
}
```

Variações de Hashing e Usos

- **MD5**: Útil para integridade de arquivos não críticos.
- **SHA-1**: Mais seguro que MD5, mas suscetível a colisões. Evitar em novas implementações.
- **SHA-256 e SHA-512**: Altamente recomendados para segurança e integridade de dados em aplicações que exigem alta confiabilidade.

Conclusão

Escolher o algoritmo de criptografia ou hash correto depende das necessidades de segurança. AES é excelente para dados em massa, enquanto RSA é ideal para trocas seguras de chaves. Para hashing, SHA-256 é preferível, garantindo maior segurança contra ataques de colisão.

Em Java, a implementação é facilitada pelas bibliotecas nativas javax.crypto e java.security, com suporte adicional de bibliotecas como BouncyCastle para configurações avançadas.

EducaCiência FastCode para comunidade