Universidade Lúrio

Faculdade de Engenharia

Edmilson Horácio Joaquim LEI 3 Nível

Continuação de Laboratório 3 SIRC

2. AESWrapRSAExample - Protegendo Chaves Privadas com Key Wrapping:

O código AESWrapRSAExampledemonstra uma técnica de segurança conhecida como key wrap, usada para proteger chaves privadas RSA. Essa abordagem é valiosa para garantir a confidencialidade e integridade das chaves privadas, impedindo o acesso não autorizado a informações confidenciais.

Proteção de Chaves Privadas DSA:

Para proteger o código do fim das chaves privadas DSA, algumas modificações são permitidas:

- Escolha do Algoritmo DSA: Alterar o tipo de chave gerada para DSA (Algoritmo de Assinatura Digital):

KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("DSA", "BC");

DSA é um algoritmo de assinatura digital amplamente utilizado, assim como o RSA, mas com propriedades diferentes.

- Uso da Chave AES para Key Wrapping : Para envolver (wrap) uma chave privada DSA, é aconselhável usar uma chave AES, que é uma excelente escolha devido ao seu desempenho e segurança. A criação da chave AES pode ser realizada da seguinte maneira:

Key wrapKey = Utils3.createKeyForAES(256, random);

Vantagens da Técnica de Proteção de Chaves Privadas

- Melhor desempenho: A criptografia simétrica, como o AES, é mais rápida do que a criptografia assimétrica, como o RSA ou DSA. Portanto, o acesso às chaves privadas protegidas é mais rápido do que se fica em texto simples.
- Proteção Adicional: Uma técnica de empacotamento protege as chaves privadas, tornando-as menos vulneráveis a acessos não autorizados, uma vez que sejam criptografadas e exijam a chave de descritiva correta.
- Flexibilidade: Essa técnica pode ser usada para proteger várias chaves privadas e é independente do algoritmo de chave pública que está sendo usado. Qualquer chave privada pode ser protegida usando essa abordagem.

RSAKeyExchangeExample - Estabelecimento de Chaves e Criptografia de Dados de Sessão:

O código RSAKeyExchangeExampledemonstra como criar envelopes de chave pública para o estabelecimento de chaves de sessão seguras em um canal de comunicação. Isso é fundamental para garantir a confidencialidade dos dados durante a comunicação.

- Geração de Chave Simétrica AES : O código começa gerando uma chave simétrica AES (sKey) e um vetor de inicialização (IV) AES (sIvSpec). Essas chaves serão usadas para criptografar os dados de sessão.
- Criação do Envelope da Chave Simétrica :
    - O envelope da chave simétrica é composto pelo IV AES seguido dos bytes da chave simétrica AES.
    - Em seguida, o envelope é criptografado usando uma chave pública RSA. Para essa operação, o código utiliza a classe Ciphercom o modo de operação OAEP (Optimal Asymmetric Encryption Padding) e preenchimento com SHA-1 e MGF1 (Mask Generation Function 1).
- Criptografia dos Dados de Sessão : Os dados de sessão são criptografados usando a chave simétrica AES e o IV.
- Descriptografia dos Dados de Sessão e Recuperação da Chave Simétrica : O código demonstra a descrição dos dados de sessão e a recuperação da chave simétrica usando a chave privada RSA. Isso é feito em etapas:
    - A chave privada RSA (privKey) é usada para descrever o envelope da chave simétrica.

- O envelope descrito é dividido em IV e chave simétrica.
- A chave simétrica e o IV são usados para descrever os dados de sessão.

3. TwoWayDHExample - Acordo de Chave Diffie-Hellman Bidirecional:

1. **Definindo os Parâmetros de Diffie-Hellman :**

   Os parâmetros g512e p512representam o domínio Diffie-Hellman, onde g512é o gerador e p512é um grande número primo. Ambas as partes utilizam esses parâmetros públicos para o acordo de chave.

2. **Configurando o Gerador de Par de Chaves Diffie-Hellman :**

   O código configura o gerador de pares de chaves Diffie-Hellman com os parâmetros p512e g512.

3. **Geração de Chaves :**

   Duas instâncias de KeyPairGenerator são criadas, uma para cada parte, para gerar pares de chaves públicas e privadas. Essas chaves são geradas usando o algoritmo Diffie-Hellman.

4. **Configurando o Acordo de Chave para Ambas as Partes :**

   Cada parte configura seu acordo de chave usando o algoritmo Diffie-Hellman.

5. **Geração de Par de Chaves para A e B:**

   K**eyPair aPair = keyGen.generateKeyPair():** Gera um par de chaves (pública e privada) para A.

6. **Iniciando o Acordo de Chave :**

   Cada parte inicializa seu acordo de chave com sua chave privada.

7. **Realização do Acordo de Chave :**

   As partes realizam um acordo de chave com a chave pública da outra parte. Esse acordo envolve várias fases e é especificado como a última fase.

ThreeWayDHExample - Acordo de Chave Diffie-Hellman Triangular:

1. **Definição dos Parâmetros de Diffie-Hellman:**

As configurações g512são p512as mesmas que no exemplo bidirecional, e são usadas publicamente por todas as partes.

2. **Configurando o Gerador de Par de Chaves Diffie-Hellman :**

O código configura o gerador de pares de chaves Diffie-Hellman com os mesmos parâmetros.

3. **Geração de Pares de Chaves para as Três Partes:**

Três pares de chaves são gerados, um para cada parte, utilizando o algoritmo Diffie-Hellman.

4. **Iniciando os Acordos de Chave:**

Cada parte inicia seu acordo de chave com sua chave privada.

5. **Realização do Acordo de Chave entre as Três Partes :**

Cada parte realiza um acordo de chave com as chaves públicas das outras duas partes, permitindo o estabelecimento de chaves compartilhadas.

6. **Derivação das Chaves Compartilhadas:**

Após a realização dos acordos de chave, cada parte gera sua chave compartilhada chamando generateSecret(). As chaves compartilhadas resultantes para A, B e C são denominadas aShared, bSharede cShared, respectivamente. Essas chaves compartilhadas podem ser usadas para garantir a confidencialidade dos dados durante a comunicação triangular.

```
import java.math.BigInteger;
import java.security.*;
import java.security.spec.DHParameterSpec;
import java.security.interfaces.DHPublicKey;
import java.security.interfaces.DHPrivateKey;
import javax.crypto.KeyAgreement;
import java.security.MessageDigest;
import java.security.Signature;
import java.util.Arrays;
public class AuthenticatedDHExample {
   private static BigInteger prime = new BigInteger(
         "153d5d6172adb43045b68ae8e1de1070b6137005686d29d3d73a7"
         + "749199681ee5b212c9b96bfdcfa5b20cd5e3fd2044895d609cf9b"
         + "410b7a0f12ca1cb9a428cc", 16);
   private static BigInteger generator = new BigInteger(
         "9494fec095f3b85ee286542b3836fc81a5dd0a0349b4c239dd387"
         + "44d488cf8e31db8bcb7d33b41abb9e5a33cca9144b1cef332c94b"
         + "f0573bf047a3aca98cdf3b", 16);
   public static void main(String[] args) throws Exception {
      DHParameterSpec dhParams = new DHParameterSpec(prime, generator);
      KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DH");
      keyGen.initialize(dhParams);
      // Geração de chaves para Alice
      KeyPair aliceKeyPair = keyGen.generateKeyPair();
```

```
DHPublicKey alicePublicKey = (DHPublicKey) aliceKeyPair.getPublic();
DHPrivateKey alicePrivateKey = (DHPrivateKey) aliceKeyPair.getPrivate();
// Geração de chaves para Bob
KeyPair bobKeyPair = keyGen.generateKeyPair();
DHPublicKey bobPublicKey = (DHPublicKey) bobKeyPair.getPublic();
DHPrivateKey bobPrivateKey = (DHPrivateKey) bobKeyPair.getPrivate();
// Alice e Bob realizam o acordo de chave
KeyAgreement aliceKeyAgree = KeyAgreement.getInstance("DH");
aliceKeyAgree.init(alicePrivateKey);
aliceKeyAgree.doPhase(bobPublicKey, true);
KeyAgreement bobKeyAgree = KeyAgreement.getInstance("DH");
bobKeyAgree.init(bobPrivateKey);
bobKeyAgree.doPhase(alicePublicKey, true);
// Geração dos segredos compartilhados
byte[] aliceSharedSecret = aliceKeyAgree.generateSecret();
byte[] bobSharedSecret = bobKeyAgree.generateSecret();
// Verificação dos segredos compartilhados
if (Arrays.equals(aliceSharedSecret, bobSharedSecret)) {
    System.out.println("Segredos compartilhados coincidem!");
} else {
    System.out.println("Segredos compartilhados não coincidem");
}
```

```
        // Assinatura digital pela Alice e verificação por Bob
        byte[] message = "Hello, Bob!".getBytes();
        Signature aliceSign = Signature.getInstance("SHA256withRSA");
        aliceSign.initSign(alicePrivateKey);
        aliceSign.update(message);
        byte[] aliceSignature = aliceSign.sign();
        Signature bobVerify = Signature.getInstance("SHA256withRSA");
        bobVerify.initVerify(alicePublicKey);
        bobVerify.update(message);
        // Verificação da assinatura
        if (bobVerify.verify(aliceSignature)) {
            System.out.println("A mensagem de Alice foi verificada!");
        } else {
            System.println("Mensagem de Alice não verificada");
        }
    }
}
```