Segurança Informática e de Redes de Computadores

Nome: Mateus Fernando Jó Matazeus

LAB4

2.1. Este tipo de construcao é útil em cenários em que pretende-se armazenar chaves privadas de forma segura, tornando-as inacessíveis a qualquer principal que não tenha a chave AES usada para protegê-las. As vantagens de usar esse tipo de construção incluem: Segurança, a chave privada RSA está protegida por uma chave AES forte, tornando-a resistente a ataques; Portabilidade: a chave privada pode ser facilmente transportada e armazenada em segurança, pois está protegida por criptografia simétrica; Flexibilidade: essa construcao pode ser aplicada a outros tipos de chaves privadas, como chaves DSA, para protegê-las da mesma forma.

2.2. Para criar envelopes de chave publica para garantir confidencialidade em um canal de comunicacao, ha necessidade de gerar uma par de chaves (privadas e publicas) para cada principal que deseja se comunicar, ussando o RSA, é importante referenciar que tambem pode-se usar outros varios algoritmos assimetricos, como o El Gammal ou DSA.A partilha de chave de publica para outro principal, a criancao de uma chave de sessao simetrica. O principal que inicia a comunicacao faz um wrap da chave de sessao usando a chave publica de outro principal usando por exemplo o xCipher e usa-a para cifrar dados com sCipher.

3.1. 1 O código fornecido mostra o acordo de chave Diffie-Hellman entre dois principais, respectivamente A e B, o processo comeca com a geracao de par de chaves RSA (Kpub, Kpriv), partindo para a parte de configuração da acordo de chave, onde ambos participantes inicializam suas instâncias de

**KeyAgreement** com suas respectivas chaves privadas. Os principais "A" e "B" realizam o acordo de chave DH, trocando suas chaves públicas. O principal "A" chama **aKeyAgree.doPhase(bPair.getPublic(), true)**, e a parte "B" chama **bKeyAgree.doPhase(aPair.getPublic(), true)**. Ambas os principais geram a chave compartilhada usando **aKeyAgree.generateSecret()** e **bKeyAgree.generateSecret()**. A chave compartilhada é derivada usando uma função de hash (SHA-1) para produzir uma chave em comum.

3.2. É possivel o numero de participantes para varios, criando e ajustando a variável numeroParticipantes para especificar quantos participantes estarão envolvidos no acordo de chave Diffie-Hellman. Isso permite que seja possivel definir o número de participantes que participarão do processo de acordo de chave. Sendo possivel alterar o valor de numeroParticipantes para o número desejado.

Cada participante realiza o acordo de chave com todas as outras partes. Isso é importante porque, no Diffie-Hellman de grupo, cada entidade deve ter um segredo compartilhado com todas as outras entidades no grupo. Um participante (representado pelo índice i) inicializa seu **KeyAgreement** com sua chave privada. Em seguida, o participante executa **keyAgree.doPhase(participants.get(j).getPublic(), true);** para realizar o acordo de chave com o participante representada pelo índice j. O **true** indica que esta é a última fase do acordo de chave. Este processo é repetido para todos os outros participantes (representados pelo índice j).

```
21
22      int numeroParticipantes = 10; // Define o número de participantes
23
24      for (int i = 0; i < numeroParticipantes; i++) {
25          KeyPairGenerator keyGen = KeyPairGenerator.getInstance( algorithm: "DH",  provider: "BC");
26          keyGen.initialize(dhParams, UtilsDH.createFixedRandom());
27          KeyPair keyPair = keyGen.generateKeyPair();
28          keyPairs.add(keyPair);
29          keyAgreements.add(KeyAgreement.getInstance( algorithm: "DH",  provider: "BC"));
30          keyAgreements.get(i).init(keyPair.getPrivate());
31      }
32
33      for (int i = 0; i < numeroParticipantes; i++) {
34          for (int j = 0; j < numeroParticipantes; j++) {
35              if (i != j) {
36                  KeyAgreement keyAgreement = keyAgreements.get(i);
```

Apos a execucao, tem-se:

```
/usr/lib/jvm/jdk-21-oracle-x64/bin/java -javaagent:/home/
Participante 0: 9d9dc120f9cf86ee5471ce4deb0a0b5a40a4b2cb
Participante 1: 9d9dc120f9cf86ee5471ce4deb0a0b5a40a4b2cb
Participante 2: 9d9dc120f9cf86ee5471ce4deb0a0b5a40a4b2cb
Participante 3: 9d9dc120f9cf86ee5471ce4deb0a0b5a40a4b2cb

Process finished with exit code 0
```