* 1. i)

RV

yi

1.2

As “*Stream Ciphers”* ao contrário das “*Block Ciphers”* não têm um tamanho fixo e invariável de bits que devem ser processados, a cifra é feita bit a bit usando uma “*key stream*” permitindo assim que blocos mais pequenos, do que os valores habituais nas “*block ciphers”,* sejam processados.

1.3

Para não permitir que sejam identificados padrões que possam levar a alguma previsão dos dados que se encontram cifrados.

2

2.1

O esquema MAC cifra toda a informação das chaves e certificados usando uma chave que estará protegida por palavra-chave, isto permite que a informação não seja lida permitindo ainda que seja verificada a integridade do *keystore*.

2.2

Caso os bits fornecidos ao método doFinal não sejam de um tamanho múltiplo ao tamanho de um bloco este irá inserir padding no final, sendo que o padding apenas deve ser inserido no último bloco. O método update não irá inserir padding, irá apenas guardar em buffer os bits que não atingiram um bloco até ser chamado o método doFinal ou novamente o update.

3

3.1 São gerados tickets para autenticar o cliente perante o serviço, estes tickets são cifrados usando a chave do cliente ou serviço, consoante o seu destinatário, e contêm um *timestamp* que irá impossibilitar ataques de replay.

3.2 Não, pois quando o *ticket* é gerado este é gerado usando informação de B. Logo, se B tentar usar para outro serviço este não irá ser válido.

3.3 Não, pois teria que ser enviado em claro o *ticket* e este contém a chave de sessão. Caso alguém escuta-se a comunicação e obtivesse a chave de sessão poderia identificar-se como sendo A e pedir acesso a serviços.

3.4 Não, pois caso o *authenticator A* fosse em claro, dado a ser um esquema MAC, o atacante poderia obter TA e usando este poderia replicar as mensagens para B e quando B pedisse confirmação de posse da chave de sessão, enviando TA cifrado com a chave de sessão, este poderia simplesmente enviar o que obteve na mensagem em claro, sem ter que proceder a qualquer decifra, validando assim a conversa.

4

4.1 Tem que se obter o certificado do emissor e verificar, usando a chave pública, se o certificado que temos foi realmente emitido usando a chave privada daquele certficiado.

4.2 É usado o número de sequência das mensagens para evitar ataques de replay, esse número é usado na geração de uma marca MAC, o que torna impossível replicar mensagens fora de ordem.

4.3 No protocolo handshake são geradas as chaves usadas para garantir a segurança do canal, essas chaves são geradas usando como base uma outra chave. Esta é gerada aleatoriamente pelo cliente e protegida usando a chave pública do servidor, o facto de esta chave ser aleatória torna impossível que seja usado um ataque de replay pois duas instâncias do protocolo handshake iriam originar chaves distintas.

4.4 A última mensagem do cliente e do servidor irá conter uma marca MAC gerada usando toda a conversa entre os dois sendo a chave o master secret para que o outro possa verificar se ambos viram o mesmo, caso algum hacker modifica-se as mensagens e dado que este não sabe o master secret ele não irá conseguir alterar esta mensagem de forma a que o servidor e/ou o cliente validem a conversa.

5

Para a implementação desta aplicação é usado um *MessageDigest* do tipo do algoritmo escolhido pelo utilizador. É feito um *while* a ler todo o ficheiro do qual se pretende obter a hash e são feitas chamadas ao método *update* sobre o *MessageDigest*, por fim é feita uma chamada ao método *digest* para obter a *hash*.

6

Foram criadas duas aplicações, uma que cifra e outra que decifra o ficheiro, existem também duas classes em que uma é responsável pela validação de cadeias de certificados X509 e outra que representa a metadata.

A validação é feita recorrendo à classe CertPathValidator que recebe um CertPath e parâmetros que incluem as raízes de confiança e um selector que sabe escolher o certificado a ser validado, caso a validação falhe é lançada uma excepção. Para a validação correr como esperado é necessário desactivar as listas de revogação porque a inexistência de uma configurada resulta numa excepção.

A metadata inclui o tipo de algoritmo simétrico usado para a cifra do ficheiro, o algoritmo assimétrico usado na cifra da chave simétrica, a chave simétrica usada (cifrada) e o certificado cuja chave pública é usada para cifrar a chave. Para cifrar a chave simétrica é usada a classe Cipher no modo WRAP\_MODE com a chave pública do certificado, e para a decifra a mesma classe mas no modo UNWRAP\_MODE e usando a chave privada correspondente ao certificado.

A cifra do ficheiro é feita também ela recorrendo à classe Cipher mas utilizando o modo ENCRYPT\_MODE, enquanto a decifra usa o modo DECRYPT\_MODE.

7

As configurações para que o servidor passe a aceitar ligações HTTPS com autenticação de cliente e servidor são todas feitas no ficheiro conf/extra/httpd-ssl.conf. O servidor funciona à base de ficheiros PEM que podem ser obtidos recorrendo à aplicação “openssl” que se encontra na pasta bin, e para extrair certificados de um keystore JKS é usada a tool keytool que vem com a plataforma java.

**Comandos OpenSSL usados:**

.cer (X509) para .pem

openssl x509 -inform der -in certificate.cer -out certificate.pem

.pfx para .pem

openssl pkcs12 -in keyStore.pfx -out keyStore.pem –nodes

**Comandos Keytool usados:**

Exportar um certificado de um keystore .jks

keytool -export -alias my\_home -file my\_home.crt -keystore herong.jks -storepass HerongJKS

Para que o servidor passe a ter autenticação é necessário que este saiba apresentar o seu certificado e o caminho de validação deste certificado. Para que o servidor apresente o seu certificado basta modificar a propriedade *SSLCertificateFile* com o caminho do certificado (no formato PEM) e para que este apresente o caminho de validação é necessário criar um ficheiro, que tem a concatenação de todos os certificados no caminho de validação no formato PEM, e modificar a propriedade *SSLCertificateChainFile* com o caminho deste mesmo ficheiro (Para efectuar a concatenação pode-se usar um editor de texto simples). Para que o cliente reconheça o certificado como válido é ainda necessário que este tenha o certificado raiz instalado como um certificado de confiança, e apenas com estes passos o servidor passa a autenticar-se perante o cliente em ligações HTTPS.

Para a autenticação do cliente é necessário criar outro ficheiro com o caminho de validação, mas desta vez com o caminho de validação dos clientes, e modificar a propriedade *SSLCACertificateFile* com o caminho para este ficheiro. É necessário depois modificar a propriedade *SSLVerifyClient* para *require*, para que este exija ao cliente que se autentique e podemos também modificar o *SSLVerifyDepth* para algo que nos convenha, sendo neste caso 3 suficiente. Por fim, é preciso que o cliente instale o seu certificado como certificado pessoal para que quando aceder ao servidor usando HTTPS este peça um certificado para autenticar o cliente e o *browser* saiba qual apresentar.