|  |
| --- |
|  |



Ana Correia - 31831

Diogo Cardoso - 32466

João Silvestre - 32766

25/10/11

*Instituto Superior Engenharia de Lisboa*

# 1ª Serie

Engenheiro: José Simão

Engenharia Informática e de Computadores

Semestre de Inverno 2011/2012

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

Índice

[Introdução 4](#_Toc307586913)

[Parte Teórica 5](#_Toc307586914)

[ 1º Exercício 5](#_Toc307586915)

[o Alínea 1.1 5](#_Toc307586916)

[o Alínea 1.2 6](#_Toc307586917)

[o Alínea 1.3 6](#_Toc307586918)

[ 2º Exercício 6](#_Toc307586919)

[o Alínea 2.1 6](#_Toc307586920)

[o Alínea 2.2 7](#_Toc307586921)

[ 3º Exercício 7](#_Toc307586922)

[o Alínea 3.1 7](#_Toc307586923)

[o Alínea 3.1 7](#_Toc307586924)

[o Alínea 3.1 8](#_Toc307586925)

[o Alínea 3.1 8](#_Toc307586926)

[ 4º Exercício 8](#_Toc307586927)

[o Alínea 4.1 8](#_Toc307586928)

[o Alínea 4.2 8](#_Toc307586929)

[o Alínea 4.3 9](#_Toc307586930)

[o Alínea 4.4 9](#_Toc307586931)

[ 5º Exercício 10](#_Toc307586932)

[ 6º Exercício 10](#_Toc307586933)

[ 7º Exercício 13](#_Toc307586934)

[Conclusão 15](#_Toc307586935)

# Introdução

# Parte Teórica

## 1º Exercício

### Alínea 1.1

Figura - Algoritmo de decifra do novo modo de operação.

RV

Ds

X'1

X1

Y1

…

…

X'2

Ds

X2

Y2

K

X'L

Ds

XL

YL

K

K

O processo de cifra do CBC não pode ser paralizável porque para efectuar cifra de um bloco necessita do resultado anterior desta mesma operação, ou seja, para efectuar o processo de cifra é necessário o anterior bloco cifrado , como é mostrado na FIGURA X. O processo de cifra do modo de operação proposto no trabalho pode ser paralizável, porque cada processo de cifra de um bloco é independente dos outros blocos, isto é, para cifrar um bloco é só necessário o vector aleatório.

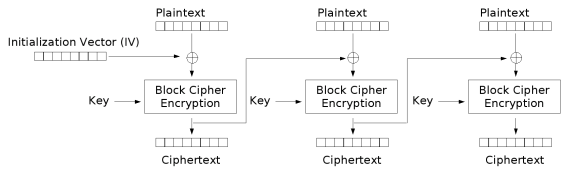


Figura - Algoritmo cifra do modo CBC

No modo de operação que foi proposto no trabalho, existe possibilidade de padrões no texto em claro serem evidenciados no texto cifrado, ou seja, para blocos iguais a sua cifra será igual, porque é utilizado o mesmo vector aleatório para a cifra de todos os blocos, como consequência, se existir padrões de texto em claro na mensagem, são notados na mensagem cifrada. No CBC não existe essa possibilidade, porque é utilizado o bloco cifrado anterior que é diferente para cada cifra.

### Alínea 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.

### Alínea 1.3

A arquitectura dos esquemas assimétricos de cifra, adiciona-se informação aleatória à mensagem, para impedir que padrões que existem no texto em claro sejam visíveis na mensagem cifrada.

## 2º Exercício

### Alínea 2.1

O *MAC* é utilizado para garantir a integridade do *keystore*, isto é, como o *keystore* tem a funcionalidade de guardar chaves(privadas ou secretas) ou certificados de confiança, logo essa informação só pode ser visualizada ou modificada por uma pessoa de confiança. Por isso para aceder ou modificar a informação que se encontra neste formato é necessário conhecer a palavra-chave que protege a chave do *MAC*, para que seja verificada a integridade do *keystore.*

### Alínea 2.2

O método update deve ser invocado para encriptar ou decriptar dados usando a operação *multiple-part.* Esta operação deve ser utilizada quando não se indica qual é o tamanho total dos dados ou quando não existe memória para guardar os dados todos. No final desta operação deve-se chamar o método doFinal para caso exista a necessidade de introduzir padding, porque o update adiciona padding aos dados.

## 3º Exercício

### Alínea 3.1

Os mecanismos existentes no protocolo *Kerberos* para a protecção contra a ataques de *replay* são N e L. O N é um número aleatório que é gerado pelo o cliente e enviado para o KDC no pedido de autenticação, este evita os ataques de replay porque deve único para cada pedido de um serviço e o L é um timestamp que é gerado e guardado no ticket.

### Alínea 3.1

A entidade B usando o bilhete de A não consegue autenticar-se como A perante outra entidade, porque o bilhete contem informação do serviço B, e é cifrado com uma chave que só o serviço B ou KDC conhece, por isso a única entidade que consegue decifrar o bilhete é o próprio B.

### Alínea 3.1

Não se pode substituir a cifra do bilhete por um esquema MAC, porque tinha-se que enviar o bilhete em claro, e este contem a chave de sessão de A para B, por isso caso um atacante estivesse a escutar a comunicação e obtivesse essa chave, este poderia se autenticar como A ao serviço B.

### Alínea 3.1

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.

## 4º Exercício

### Alínea 4.1

Ao validar a cadeia do certificado emissor, pode-se dizer que a chave publica é a chave de autenticação do certificado.

### Alínea 4.2

O protocolo record evita os ataques replay utilizando um número de sequencia que se encontra protegido por um MAC, desta forma é impossível replicar as mensagens.

### Alínea 4.3

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

### Alínea 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.

## 5º Exercício

Realizou-se um aplicação para obtenção de hash criptográficos de ficheiros, em que o seu funcionamento consiste:

* O utilizador indica qual é a função de hash e o ficheiro que pretende obter a hash.
* É apresentado ao utilizador no standard output o valor do hash.

Na implementação foi utilizada a classe MessageDigest do tipo de hash que o utilizador introduziu. É efectuado a leitura do ficheiro recebido pelo utilizador utilizando o método update da classe anterior. Por fim é obtido o hash ao invocar o método digest e apresentar esse valor ao utilizador.

## 6º Exercício

Este exercício foi dividido em duas aplicações:

* Cifra de um ficheiro indicado pelo o utilizador
* Decifra desse ficheiro

O objectivo desta aplicação é cifra um ficheiro e produzir um ficheiro meta-data com a informação necessária para decifrar esse ficheiro.

Para realizar a operação de cifra era necessário que o utilizador indica-se os seguintes parâmetros:

* Localização do ficheiro para cifra.
* Certificado para cifrar a chave simétrica.
* Directoria de certificados intermédios.
* Key store com as âncoras de confiança.

Antes de efectuar a operação de cifra era necessário validar a cadeia de certificados do certificado recebido do utilizador. A validação foi efectuado com a classe CertPathValidator que recebe um CertPath que contem o certificado folha e os certificados intermédios e *BuilderParameters* que contem as raízes de confiança que se encontram no keystore recebido como parâmetro, e um selector que indica qual é o certificado a ser validado, caso esta validação falhe é lançada uma excepção. Como não existe listas de revogação, é necessário desactivar a pesquisa desta lista.

Caso a validação do certificado seja aprovada, é efectuado a cifra do ficheiro recorrendo à classe Cipher em modo ENCRYPT\_MODE. Por fim, é criado um ficheiro meta-data com a seguinte informação:

* Algoritmo simétrico
* Algoritmo assimétrico
* Chave simétrica
* Certificado utilizado para cifrar a chave simétrica

Para cifrar a chave simétrica é usada a classe Cipher no modo WRAP\_MODE com a chave pública do certificado.

Na aplicação de decifra é recebido pelo o utilizador o ficheiro cifrado e o ficheiro da meta-data. É efectuado a decifrado a chave simétrica utilizado a chave privado do certificado que se encontra na meta-data e por fim, decifrado o ficheiro usando a classe Cipher em modo DECRYPT\_MODE.

#### Alínea 6.1

## 7º Exercício

Para que o servidor Apache consiga aceitar ligações HTTPS com autenticação de cliente e servidor era necessário modificar o ficheiro conf/extra/httpd-ssl.conf.

O servidor funciona à base de ficheiros PEM que podem ser obtidos recorrendo à aplicação “openssl”, como demonstra na Listagem , 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, como demonstra na Listagem X.

.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

Listagem 1 - Comandos OpenSSL

Exportar um certificado de um keystore .jks

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

Listagem 2 - Comandos Keytool

Para que o servidor passe a funcionar com o protocolo TLS é necessário que consiga apresentar o seu certificado e o caminho de validação deste certificado, por isso, é necessario modificar a propriedade *SSLCertificateFile* com o caminho do certificado (no formato PEM) e para que este apresente o caminho de validação é necessário modificar a propriedade *SSLCertificateChainFile* com um ficheiro, que contenha todos os certificados da sua cadeia no formato PEM. 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.

# Conclusão

Esta serie de exercícios serviu para consolidar-mos a matéria dada nas aulas bem como ter um contacto directo com as APIs que o java dispõe para manipular certificados, chaves publicas e privadas, etc.

Serviu ainda para aprofundar-mos os conhecimentos passados nas aulas sobre os protocolos de comunicação segura.