

**Departamento de Engenharia de Eletrónica e Telecomunicações e de Computadores**

**Primeiro trabalho - Semestre de Inverno de 24/25**

|  |  |  |
| --- | --- | --- |
| Autores: | 44811 | André Santos |
|  |  |  |
|  |  |  |

Relatório para a Unidade Curricular de Segurança Informática da Licenciatura em Engenharia Informática e de Computadores

Professor: Diego Gimenez Passos

23 - 10 - 2024

indice

[Introdução 4](#_Toc180866495)

[1. Confidencialidade e Autenticidade no Esquema CI 5](#_Toc180866496)

[2. Protocolo de Estabelecimento Seguro de Chave Simétrica 7](#_Toc180866497)

[3. Conceito de Não Repúdio e sua Aplicação na Comunicação 7](#_Toc180866498)

[4. Certificados X.509 e Perfil PKIX 8](#_Toc180866499)

[4.1. Confiança em Certificados e Situações de Perda de Confiança 8](#_Toc180866500)

[5. Análise da Propagação de Erros em Criptografia Simétrica: Modos de Operação ECB, CBC e CTR 10](#_Toc180866501)

[6. Gerador de Checksum para Verificação de Ficheiros 13](#_Toc180866502)

[7. Implementação de um Sistema de Criptografia Híbrida para Ficheiros 14](#_Toc180866503)

[Exemplos 14](#_Toc180866504)

# Introdução

Este relatório trata-se de um trabalho prático realizado no âmbito da disciplina de Segurança Informática, com foco na análise de cifras de bloco e modos de operação. O objetivo principal deste trabalho é aprofundar a compreensão dos conceitos de confidencialidade e integridade em criptografia, através da análise de diferentes cifras de bloco e da investigação do impacto da propagação de erros nos diferentes modos de operação. Na primeira parte, são explorados conceitos fundamentais de criptografia simétrica, como cifras de bloco e modos de operação, e são analisadas as características de segurança de algoritmos como o DES (Data Encryption Standard) e o AES (Advanced Encryption Standard). Na segunda parte, é realizada uma experiência prática para analisar o efeito da propagação de erros no modo CBC, utilizando um programa Java (com JCA) desenvolvido com este propósito. Os resultados obtidos são analisados e discutidos para compreender melhor as implicações que existem no âmbito de encriptação.

Parte 1

# Confidencialidade e Autenticidade no Esquema CI

**Análise do Esquema Criptográfico CI**

O esquema criptográfico CI é definido como:

**CI(k1,k2,m)=Es(k1)(m)∣∣T(k2)(k1)**

Onde:

* **Es(k1)(m)** é a cifração simétrica da mensagem m usando a chave secreta k1.
* **T(k2)(k1)** é o código de autenticação de mensagem (MAC) da chave k1 usando a chave k2.
* **||** representa a concatenação de bits.
* **1. Confidencialidade:**

Sim, o esquema CI garante confidencialidade. A confidencialidade é garantida pelo uso do esquema de cifra simétrica Es(k1)(m).A mensagem m é cifrada usando a chave k1, tornando-a ilegível para qualquer pessoa que não possua a chave.

* + Desde que k1 seja mantida confidencial e o algoritmo Es seja seguro, um atacante não conseguirá recuperar m a partir deEs(k1)(m).
* **Exposição Potencial da Chave k1:**
  + O MAC T(k2)(k1)é calculado sobre a própria chave k1.
  + Se k2 for mantida secreta, um atacante não pode calcular k1 a partir de T(k2)(k1).
  + No entanto, se k2 for conhecida pelo atacante ou puder ser comprometida, o MAC pode ser usado para realizar um ataque por força bruta ou análise criptográfica para recuperar k1.
* **Considerações sobre o MAC:**
  + Mesmo que k2 seja mantida secreta, expor T(k2)(k1) pode fornecer informações sobre k1 se o MAC não for resistente a ataques de extensão ou inversão.
  + A segurança do MAC é crucial para garantir que nenhuma informação sobre k1 seja revelada.

**Conclusão sobre a Confidencialidade:**

O esquema **garante a confidencialidade da mensagem m** desde que:

* As chaves k1 e k2 sejam mantidas secretas.
* Os algoritmos Es e T sejam criptograficamente seguros.
* O MAC T(k2)(k1) não revele informações sobre k1.
* **Riscos Potenciais:**
  + Se T não for seguro ou k2 for comprometida, a confidencialidade pode ser quebrada.
  + Portanto, a segurança depende fortemente da segurança dada pelos algoritmos e da proteção das chaves.

**Autenticidade:**

A autenticidade assegura que a mensagem recebida é genuína e não foi alterada durante a transmissão.

* **Proteção da Integridade de Es(k1)(m):**
  + O MAC T(k2)(k1) é calculado sobre k1, não sobre Es(k1)(m) ou m.
  + Não há nenhum mecanismo para detetar alterações em Es(k1)(m).
* **Possibilidade de Alterações Não Detetadas:**
  + Um atacante pode intercetar e modificar Es(k1)(m) para Es(k1)(m′) sem que o recetor detete.
  + Como o MAC não cobre a mensagem cifrada, não há como verificar a integridade de Es(k1)(m)
* **Verificação Limitada:**
  + O recetor pode verificar que T(k2)(k1) é válido, ou seja, que k1 não foi alterada.
  + Contudo, isso não garante que Es(k1)(m) não tenha sido modificado.

Portanto, não, o esquema CI não garante autenticidade. A autenticidade exige que o recetor da mensagem possa verificar a identidade do remetente e garantir que a mensagem não foi alterada durante o seu envio.

O esquema CI utiliza um MAC T(k2)(k1) para gerar um código de autenticação da chave k1 usando a chave k2. No entanto, este MAC não é calculado sobre a mensagem m, mas sim sobre a chave k1. Isso significa que um atacante poderia modificar a mensagem m sem alterar o MAC, comprometendo a integridade da mensagem.

# Protocolo de Estabelecimento Seguro de Chave Simétrica

Este protocolo permite que Alice e Bob estabeleçam uma chave simétrica secreta com confidencialidade e integridade, mesmo que apenas Bob conheça a chave pública de Alice inicialmente.

1. Bob gera um par de chaves assimétricas, uma chave privada (PrB) e uma chave pública (PuB).
2. Bob envia sua chave pública PuB para Alice.
3. Alice gera uma chave simétrica secreta (K) aleatoriamente.
4. Alice cifra a chave simétrica K usando a chave pública de Bob PuB: C = E(PuB, K).
5. Alice envia a chave cifrada C para Bob.
6. Bob decifra a chave cifrada C usando sua chave privada PrB: K = D(PrB, C).
7. Alice e Bob agora compartilham a chave simétrica secreta K e podem usá-la para comunicação segura.

**Análise de Segurança:**

**Confidencialidade**: A chave simétrica K é cifrada usando a chave pública de Bob, garantindo que apenas Bob, com sua chave privada correspondente, possa decifrá-la.

**Integridade**: A integridade da chave simétrica K pode ser garantida adicionando um código de autenticação de mensagem (MAC) à mensagem cifrada no Passo 5. Alice pode gerar um MAC da chave cifrada C usando uma função hash e sua chave privada, e enviar o MAC junto com C. Bob pode então verificar a integridade da chave decifrada calculando o MAC de C usando a chave pública de Alice e comparando-o com o MAC recebido.

# Conceito de Não Repúdio e sua Aplicação na Comunicação

**Definição de Não Repúdio:**

O **não repúdio** é uma propriedade na segurança da informação que assegura que uma parte numa comunicação não pode negar a autoria de uma mensagem ou ação previamente realizada. Em outras palavras, se Alice enviar uma mensagem a Bob, ela não pode posteriormente afirmar que não o fez.

**Aplicação na Comunicação entre Alice e Bob:**

* **Cenário:**
  + Alice e Bob partilham uma chave simétrica estabelecida de forma segura.
  + Utilizam essa chave para cifrar mensagens e proteger com um MAC.
* **Análise:**
  + Como ambos utilizam a mesma chave simétrica, não é possível distinguir qual dos dois criou ou modificou uma mensagem apenas com base na chave utilizada.
  + O MAC assegura a integridade e autenticidade das mensagens durante a comunicação, mas não oferece não repúdio.
* **Não Repúdio não é Garantido:**
  + A comunicação entre Alice e Bob **não garante o não repúdio**. Isto porque, com chaves simétricas, ambos têm a capacidade de gerar e verificar MACs, tornando impossível provar a terceiros qual das partes enviou uma determinada mensagem.
  + Para garantir o não repúdio, seria necessário utilizar assinaturas digitais baseadas em criptografia assimétrica, onde apenas o detentor da chave privada pode assinar mensagens, e a assinatura pode ser verificada por qualquer um que possua a chave pública correspondente.

# 4. Certificados X.509 e Perfil PKIX

## 4.1. Confiança em Certificados e Situações de Perda de Confiança

**Possibilidade de Deixar de Confiar num Certificado:**

Sim, é possível que um sistema Sa deixe de confiar num certificado C que atualmente é considerado confiável.

**Situações em que Isto Pode Ocorrer:**

1. **Expiração do Certificado:**
   * Os certificados têm um período de validade. Após a data de expiração, o certificado não é considerado válido.
2. **Revogação do Certificado:**
   * O emissor do certificado (Autoridade Certificadora) pode revogar o certificado antes da data de expiração.
   * Motivos para revogação incluem comprometimento da chave privada, mudança de informação de identificação ou comportamento malicioso por parte do titular do certificado.
3. **Comprometimento da Autoridade Certificadora:**
   * Se a Autoridade Certificadora que emitiu o certificado for comprometida ou perder a confiança, todos os certificados por ela emitidos podem ser considerados não confiáveis.
4. **Atualizações de Políticas de Segurança:**
   * O sistema Sa pode atualizar as suas políticas e deixar de aceitar certos algoritmos criptográficos ou tamanhos de chave considerados inseguros, afetando a confiança nos certificados que os utilizam.
5. **Problemas na Cadeia de Certificação:**
   * Se algum certificado intermediário na cadeia for revogado ou inválido, isso afeta a confiança em certificados dependentes dessa cadeia.

**4.2. Validação da Cadeia de Certificados com Ciclo**

**Análise da Cadeia:**

A cadeia de certificados fornecida é:

Ca → Cb → Cc → Cd → Cb

Onde Ca → Cb indica que o certificado de A é assinado por B.

**Verificação de Validade:**

* **Ciclo na Cadeia:**
  + Observa-se que a cadeia retorna a Cb após Cd, criando um ciclo (Cd → Cb).
* **Regras de Validação PKIX:**
  + O perfil PKIX para certificados X.509 especifica que as cadeias de certificação devem ser acíclicas.
  + Cada certificado na cadeia deve ser emitido por uma autoridade superior até chegar a uma Autoridade Certificadora Raiz confiável.

**Conclusão:**

* **Cadeia Inválida:**
  + A cadeia apresentada **não é válida** devido ao ciclo criado entre Cd e Cb.
  + Um ciclo impede a validação correta da cadeia, pois não é possível determinar um ponto de confiança final (Autoridade Raiz).
  + Para a cadeia ser válida, deveria ser linear e terminar numa Autoridade Certificadora Raiz confiável sem retornar a certificados anteriores.

Parte 2

# 5. Análise da Propagação de Erros em Criptografia Simétrica: Modos de Operação ECB, CBC e CTR

Primeiramente precisamos de gerar uma chave com tamanho de 128 bits para podermos cifrar a mensagem ccom os diversos modos de encriptação.

Podemos gerar chaves utilizando funções de hash como md5 ou outros tipos de hash para criar a nossa chave de 128 bits e IV da seguinte forma:

echo -n "chave123" | openssl dgst -md5 -hex | cut -d ' ' -f2 > chave.key

Para gerar o vetor de inicialização iremos utilizar openssl em vez de uma funçao de hash de 128 bits:

openssl rand -hex 16 > iv.bin

**5.1**

para encriptar a mensagem com a nossa chave e IV:

Modo ECB:

openssl enc -aes-128-ecb -in mensagem.txt -out cripto\_ecb.dat -K $(cat chave.key)

Modo CBC:

openssl enc -aes-128-cbc -in mensagem.txt -out cripto\_cbc.dat -K $(cat chave.key) -iv $(cat iv.bin)

Modo CTR:

openssl enc -aes-128-ctr -in mensagem.txt -out cripto\_ctr.dat -K $(cat chave.key) -iv $(cat iv.bin)

**5.2**

Para ver qual é o byte na posição 1000 em cada um dos ficheiros encriptados, respetivamente ECB, CBC E CTR:

tail -c +1000 cripto\_ecb.dat | head -c 1 | hexdump -C

00000000 90 |.|

tail -c +1000 cripto\_cbc.dat | head -c 1 | hexdump -C

00000000 bf |.|

tail -c +1000 cripto\_ctr.dat | head -c 1 | hexdump -C

00000000 e4 |.|

5.3

Para separar o prefix e sufixo do byte na posição 1000 para ficheiros distintos:

cripto\_ecb:

head -c 1000 cripto\_ecb.dat > prefixo\_ecb.dat

tail -c +1002 cripto\_ecb.dat > sufixo\_ecb.dat

cripto\_cbc:

head -c 1000 cripto\_cbc.dat > prefixo\_cbc.dat

tail -c +1002 cripto\_cbc.dat > sufixo\_cbc.dat

cripto\_ctr:

head -c 1000 cripto\_ctr.dat > prefixo\_ctr.dat

tail -c +1002 cripto\_ctr.dat > sufixo\_ctr.dat

5.4

Vamos criar o byte que iremos introduzir nos criptogramas de forma a alterar o byte na posição 1000 de cada criptograma original:

echo -n -e '\xAA' > novoByte.txt

cat prefixo\_ecb.dat novoByte.txt sufixo\_ecb.dat > novo\_cripto\_ecb.dat

cat prefixo\_cbc.dat novoByte.txt sufixo\_cbc.dat > novo\_cripto\_cbc.dat

cat prefixo\_ctr.dat novoByte.txt sufixo\_ctr.dat > novo\_cripto\_ctr.dat

Para verificar que os novos criptogramas têm exatamente o mesmo tamanho original:

ls -l cripto\_ecb.dat

-rw-rw-r-- 1 alphabyte alphabyte 68528 Oct 15 16:39 cripto\_ecb.dat

ls -l novo\_cripto\_ecb.dat

-rw-rw-r-- 1 alphabyte alphabyte 68528 Oct 15 16:50 novo\_cripto\_ecb.dat

para decifrar os novos criptogramas gerados com o novo byte entre o prefixo e sufixo, usamos os seguintes comandos:

openssl enc -d -aes-128-ecb -in novo\_cripto\_ecb.dat -out decifrado\_ecb.txt -K $(cat chave.key)

openssl enc -d -aes-128-cbc -in novo\_cripto\_cbc.dat -out decifrado\_cbc.txt -K $(cat chave.key) -iv $(cat iv.bin)

openssl enc -d -aes-128-ctr -in novo\_cripto\_ctr.dat -out decifrado\_ctr.txt -K $(cat chave.key) -iv $(cat iv.bin)

Para colocar o dump hexadecimal e podermos comparar com o correspondente ASCII, usamos o parametro -C juntamente com hexdump e redirecionamos o standard output para um ficheiro:

hexdump -C decifrado\_ecb.txt > dump\_ecb.hex

hexdump -C decifrado\_cbc.txt > dump\_cbc.hex

hexdump -C decifrado\_ctr.txt > dump\_ctr.hex

hexdump -C mensagem.txt > dump\_original.hex

Finalmente para comprarmos o que aconteceu após o processo de decifra, utilizamos a ferramnta diff para fazer uma comparação sucinta entre os diferentes modos de encriptação e decriptação

diff dump\_original.hex dump\_ecb.hex

63c63

< 000003e0 2c 20 69 74 20 66 72 65 71 75 65 6e 74 6c 79 20 |, it frequently |

---

> 000003e0 78 44 7c e0 ae a9 90 75 28 46 9f 13 14 e0 b5 38 |xD|....u(F.....8|

diff dump\_original.hex dump\_cbc.hex

63,64c63,64

< 000003e0 2c 20 69 74 20 66 72 65 71 75 65 6e 74 6c 79 20 |, it frequently |

< 000003f0 69 73 2e 20 20 54 68 69 73 20 6e 6f 74 65 20 64 |is. This note d|

---

> 000003e0 7b 3b d8 bd bf 74 09 6c 64 73 72 fe e3 81 ed 6b |{;...t.ldsr....k|

> 000003f0 69 73 2e 20 20 54 68 69 66 20 6e 6f 74 65 20 64 |is. Thif note d|

diff dump\_original.hex dump\_ctr.hex

63c63

< 000003e0 2c 20 69 74 20 66 72 65 71 75 65 6e 74 6c 79 20 |, it frequently |

---

> 000003e0 2c 20 69 74 20 66 72 65 a7 75 65 6e 74 6c 79 20 |, it fre.uently |

**Resultados:**

**ECB**: A modificação de um único byte no criptograma resultou na alteração de todo o bloco de 16 bytes onde o byte foi modificado no texto decifrado.

**CBC**: A modificação de um único byte no criptograma afetou o próprio bloco onde o byte foi modificado e o bloco seguinte no texto decifrado.

**CTR**: A modificação de um único byte no criptograma resultou na alteração de apenas um byte no texto decifrado, na mesma posição.

A experiência realizada permitiu analisar o efeito da modificação de um único byte em criptogramas gerados com diferentes modos de operação. No caso específico do modo CBC (Cipher Block Chaining), observamos um comportamento que, à primeira vista, pode parecer contraintuitivo: a alteração propaga-se apenas para o próprio bloco modificado e o bloco seguinte, não afetando os blocos subsequentes.

Inicialmente, tínhamos a ideia que a modificação de um byte em um criptograma CBC afetasse todos os blocos seguintes, dado que cada bloco cifrado depende de todos os blocos anteriores. Essa dependência ocorre porque o texto cifrado do bloco anterior é usado na operação XOR com o texto plano do bloco atual antes da cifragem.

No entanto, a análise do processo de decifragem revela um mecanismo de "autocorreção" inerente ao modo CBC. A alteração no byte afeta a decifragem do próprio bloco e do bloco seguinte, mas a partir do terceiro bloco a decifragem prossegue normalmente, limitando a propagação do erro.

Essa característica do CBC, que limita a propagação do erro a apenas dois blocos impede que um erro isolado comprometa a integridade de toda a mensagem decifrada.

Apesar de ser um modo de operação forte (tolerante a falhas), é fundamental lembrar que a modificação de um byte, mesmo que limitada a dois blocos, ainda corrompe os dados decifrados. Para assegurar a integridade da mensagem, a utilização de um mecanismo adicional, como um código de autenticação de mensagem (MAC), é indispensável

# Gerador de Checksum para Verificação de Ficheiros

Compilar:

javac FileHashGenerator.java

Calcular hash de um ficheiro:

java FileHashGenerator <filename> MD5 SHA-256

Exemplo:

java FileHashGenerator certificates-keys/trust-anchors/CA1.cer SHA-256

# Implementação de um Sistema de Criptografia Híbrida para Ficheiros

Compilar o programa:

javac -cp commons-codec-1.17.1.jar HybridFileEncryptor.java

## Exemplos

1. Encriptação e Decriptação com AES/CBC/PKCS5Padding

Comando de encriptação:

java -cp .;commons-codec-1.17.1.jar HybridFileEncryptor -enc mensagem.txt certificates-keys\end-entities\Alice\_2.cer -symAlg AES -asymAlg RSA -transformation AES/CBC/PKCS5Padding

Comando de decriptação:

java -cp .;commons-codec-1.17.1.jar HybridFileEncryptor -dec encrypted\_data.enc encrypted\_key.enc certificates-keys\pfx\Alice\_2.pfx changeit -symAlg AES -asymAlg RSA -transformation AES/CBC/PKCS5Padding

2. Encriptação e Decriptação com AES/CBC/NoPadding

Comando de encriptação:

java -cp .;commons-codec-1.17.1.jar HybridFileEncryptor -enc mensagem.txt certificates-keys\end-entities\Alice\_2.cer -symAlg AES -asymAlg RSA -transformation AES/CBC/NoPadding

Comando de decriptação:

java -cp .;commons-codec-1.17.1.jar HybridFileEncryptor -dec encrypted\_data.enc encrypted\_key.enc certificates-keys\pfx\Alice\_2.pfx changeit -symAlg AES -asymAlg RSA -transformation AES/CBC/NoPadding

3. Encriptação e Decriptação com DES/CBC/PKCS5Padding

Comando de encriptação:

java -cp .;commons-codec-1.17.1.jar HybridFileEncryptor -enc mensagem.txt certificates-keys\end-entities\Alice\_2.cer -symAlg DES -asymAlg RSA -transformation DES/CBC/PKCS5Padding

Comando de decriptação:

java -cp .;commons-codec-1.17.1.jar HybridFileEncryptor -dec encrypted\_data.enc encrypted\_key.enc certificates-keys\pfx\Alice\_2.pfx changeit -symAlg DES -asymAlg RSA -transformation DES/CBC/PKCS5Padding

4. Encriptação e Decriptação com TripleDES (DESede)/CBC/PKCS5Padding

TripleDES é melhor e mais segura que DES:

Comando de encriptação:

java -cp .;commons-codec-1.17.1.jar HybridFileEncryptor -enc mensagem.txt certificates-keys\end-entities\Alice\_2.cer -symAlg DESede -asymAlg RSA -transformation DESede/CBC/PKCS5Padding

Comando de decriptação:

java -cp .;commons-codec-1.17.1.jar HybridFileEncryptor -dec encrypted\_data.enc encrypted\_key.enc certificates-keys\pfx\Alice\_2.pfx changeit -symAlg DESede -asymAlg RSA -transformation DESede/CBC/PKCS5Padding

5. Encriptação e Decriptação com AES/CTR/NoPadding

Comando de encriptação:

java -cp .;commons-codec-1.17.1.jar HybridFileEncryptor -enc mensagem.txt certificates-keys\end-entities\Alice\_2.cer -symAlg AES -asymAlg RSA -transformation AES/CTR/NoPadding

Comando de decriptação:

java -cp .;commons-codec-1.17.1.jar HybridFileEncryptor -dec encrypted\_data.enc encrypted\_key.enc certificates-keys\pfx\Alice\_2.pfx changeit -symAlg AES -asymAlg RSA -transformation AES/CTR/NoPadding