Parte 1

## Confidencialidade e Autenticidade no Esquema CI

O esquema criptográfico CI é definido como:

CI(k1, k2, m) = Es(k1)(m) || T(k2)(k1)

Onde:

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

**Confidencialidade:**

O esquema **garante confidencialidade** da mensagem m. Isto porque m é cifrada usando um esquema de cifra simétrica Es com a chave secreta k1. Desde que o algoritmo de cifragem seja seguro e a chave k1 seja mantida confidencial, um atacante não conseguirá recuperar m a partir de Es(k1)(m).

**Autenticidade:**

O esquema **não garante autenticidade** da mensagem m. O MAC T(k2)(k1) é calculado sobre a chave k1, não sobre a mensagem cifrada ou a própria mensagem. Isto significa que, embora se possa verificar a integridade da chave k1, não há verificação da integridade de Es(k1)(m). Um atacante poderia modificar o texto cifrado Es(k1)(m) sem ser detectado, já que não existe um MAC sobre a mensagem ou o texto cifrado.

* **Confidencialidade:** Garantida para a mensagem m devido à cifragem com Es(k1)(m).
* **Autenticidade:** Não garantida para a mensagem m, pois o MAC não protege m nem o texto cifrado.

## 2. Protocolo de Estabelecimento Seguro de Chave Simétrica

Para estabelecer uma chave simétrica de forma segura entre Alice e Bob utilizando criptografia assimétrica, especialmente considerando que Bob conhece a chave pública de Alice, mas Alice não conhece a chave pública de Bob, podemos definir um protocolo específico que garanta tanto a confidencialidade quanto a integridade da chave simétrica (Ks). A seguir, apresento um protocolo detalhado, passo a passo, incluindo os métodos de encriptação e autenticação necessários.

Pré-requisitos:

Chaves de Bob e Alice:

Alice possui um par de chaves assimétricas:

Chave Pública: Pub\_Alice

Chave Privada: Priv\_Alice

Bob possui um par de chaves assimétricas:

Chave Pública: Pub\_Bob

Chave Privada: Priv\_Bob

Conhecimento de Chaves:

Bob conhece a Pub\_Alice.

Alice não conhece a Pub\_Bob.

Algoritmos de Criptografia:

Criptografia Assimétrica: Utilizaremos RSA para encriptação/decriptação.

Criptografia Simétrica: Utilizaremos AES para cifrar a chave simétrica.

Funções de Hash: SHA-256 para integridade.

Assinaturas Digitais: RSA para assinar mensagens.

Passo a Passo do Protocolo:

1. Geração da Chave Simétrica por Bob

Bob gera uma chave simétrica aleatória (Ks) utilizando um gerador de números aleatórios seguro.

**Ks = Geração\_Aleatória\_AES()**

2. Preparação da Mensagem para Alice

Bob cria uma mensagem contendo a chave simétrica (Ks) e um número aleatório (nonce) Nb para garantir a unicidade da transação.

**Mensagem\_Bob = Ks || Nb**

Bob encripta a mensagem usando a chave pública de Alice (Pub\_Alice) para garantir a confidencialidade de Ks.

**Encrypted\_Message = RSA\_Encrypt(Pub\_Alice, Mensagem\_Bob)**

3. Assinatura da Mensagem por Bob

Para garantir a autenticidade e integridade da mensagem, Bob assina a mensagem original (Mensagem\_Bob) com a sua chave privada (Priv\_Bob).

Assinatura\_Bob = RSA\_Sign(Priv\_Bob, Hash(Mensagem\_Bob))

Bob envia para Alice a seguinte estrutura:

Pacote\_Bob = Encrypted\_Message || Assinatura\_Bob

4. Receção e Verificação da Mensagem por Alice

Alice recebe Pacote\_Bob e separa Encrypted\_Message e Assinatura\_Bob.

Alice decripta Encrypted\_Message usando a sua chave privada (Priv\_Alice) para obter Ks e Nb.

**Mensagem\_Bob = RSA\_Decrypt(Priv\_Alice, Encrypted\_Message)**

**Ks || Nb = Mensagem\_Bob**

Alice verifica a assinatura Assinatura\_Bob utilizando a chave pública de Bob (Pub\_Bob).

Nota: Como Alice inicialmente não conhece Pub\_Bob, assume-se que Bob envia também sua Pub\_Bob de forma segura ou que Alice obtém Pub\_Bob através de um repositório confiável.

**Verificação = RSA\_Verify(Pub\_Bob, Hash(Ks || Nb), Assinatura\_Bob)**

Se a verificação for bem-sucedida, Alice confirma que a mensagem veio realmente de Bob e que Ks não foi alterada.

5. Confirmação da Receção por Alice

Alice gera um número aleatório (nonce) Na para desafiar Bob e confirmar a posse de Ks.

Na = Geração\_Aleatória()

Alice cria uma mensagem de resposta contendo Na, cifrada com Ks para garantir confidencialidade.

Mensagem\_Alice = Na

Encrypted\_Response = AES\_Encrypt(Ks, Mensagem\_Alice)

Alice assina a mensagem de resposta com a sua chave privada (Priv\_Alice) para garantir a autenticidade.

Assinatura\_Alice = RSA\_Sign(Priv\_Alice, Hash(Mensagem\_Alice))

Alice envia para Bob a seguinte estrutura:

Pacote\_Alice = Encrypted\_Response || Assinatura\_Alice

6. Receção e Verificação da Resposta por Bob

Bob recebe Pacote\_Alice e separa Encrypted\_Response e Assinatura\_Alice.

Bob decripta Encrypted\_Response usando Ks para obter Na.

Mensagem\_Alice = AES\_Decrypt(Ks, Encrypted\_Response)

Na = Mensagem\_Alice

Bob verifica a assinatura Assinatura\_Alice utilizando a chave pública de Alice (Pub\_Alice).

Verificação = RSA\_Verify(Pub\_Alice, Hash(Mensagem\_Alice), Assinatura\_Alice)

Se a verificação for bem-sucedida, Bob confirma que a resposta veio realmente de Alice e que Ks foi estabelecida corretamente.

7. Estabelecimento da Chave Simétrica

Após a verificação bem-sucedida em ambos os lados, Alice e Bob possuem a chave simétrica segura (Ks), que pode ser utilizada para comunicação cifrada e autenticada subsequente.

Análise de Segurança do Protocolo

Confidencialidade:

Bob encripta Ks e Nb com Pub\_Alice, garantindo que apenas Alice possa decriptar e obter Ks.

As mensagens subsequentes (Encrypted\_Response) são cifradas com Ks, garantindo que apenas Alice e Bob possam lê-las.

Integridade e Autenticidade:

Assinatura Digital: As assinaturas digitais de Bob e Alice garantem que as mensagens não foram alteradas e que realmente provêm das partes legítimas.

Nonces (Nb e Na): Impedem ataques de repetição, assegurando a unicidade da sessão de estabelecimento de chave.

Não Repúdio:

Assinaturas Digitais garantem que Bob não pode negar o envio da chave (Ks) e que Alice não pode negar a receção e validação da chave.

Proteção contra Ataques Man-in-the-Middle:

A utilização de assinaturas digitais e criptografia assimétrica protege contra a interceptação e modificação das mensagens, prevenindo que um atacante consiga interceptar ou alterar a chave simétrica.

Autenticação Mútua:

Embora inicialmente apenas Bob conheça a Pub\_Alice, as assinaturas digitais permitem que Alice e Bob autentiquem-se mutuamente após a troca inicial de mensagens.

## 3. 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 mais 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 do titular.
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.

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 |