

## 1<sup>a</sup> Série de Exercícios

## Licenciatura em Engenharia Informática e de Computadores

Segurança Informática

Prof. Eng. José Manuel Simão

Grupo 7:

Hugo Fora A42121

Luana Silva A42189

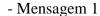
Rui Lima A42200

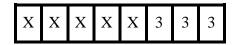
Bruno Lourenço A42400

Lisboa, 18 de outubro de 2017

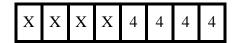
## 1ª Série de Exercícios

- 1. Este esquema não cumpre os requisitos, uma vez que a chave é incluída no criptograma, como tal não garante confidencialidade. Outra razão advém do facto de no caso de o T(k)(m) ser alterado, a mensagem será visualizada como não autêntica, apesar de o ser. Por outro lado, caso o criptograma seja alterado no esquema simétrico de cifra, Es(k)(m), a autenticação mantém a sua validade, apesar de a mensagem já não ser autêntica.
- **2.** A função de *hash* é usada para converter um input arbitrário num valor de tamanho fixo, que é normalmente muito menor, e garante as propriedades da assinatura. A primitiva da assinatura serve para assinar e misturar a chave com a mensagem.
- **3.** Em E(k)(m1||m2), é cifrada a concatenação das mensagens, e em E(k)(m1)||E(k)(m2), é cifrada cada mensagem em separado sendo os seus resultados concatenados. Uma vez que o esquema de cifra simétrico usa *padding*, quer seja para preencher os bits do bloco em falta, quer seja para indicar que o tamanho da cifra é múltiplo do número de bits do bloco, a quantidade de bits de *padding* será sempre diferente, criando assim um resultado sempre diferente para as cifras. Exemplo:

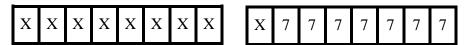




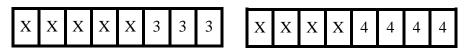
- Mensagem 2



- Cifra após concatenação



- Concatenação após cifra



**4.** O ataque de *Vaudenay* consiste em usar os bits de *padding* para descobrir a mensagem em esquemas cujas primitivas usam modo de operação CBC. Uma vez que o modo *Galois Counter Mode* (GCM) não necessita de usar *padding* nem usa o modo de operação CBC, torna-se por si só invulnerável.

## 1ª Série de Exercícios

- **5.1.** Não, uma vez que as chaves privadas são usadas apenas para a operação de autenticação dos certificados, ficando a validação ao encargo da chave pública.
- **5.2.** Devido á assinatura digital que o certificado possui garante-se a autenticidade da chave pública, uma vez que uma alteração num bit da chave pública invalidaria a assinatura.
- **5.3.** Sendo que os certificados auto assinados são os certificados raiz então é necessário garantir a sua integridade de modo a assegurar que todos os seus certificados derivados possam ser confiados.
- **6.1.** Este exercício tem como objectivo calcular o *hash* dos ficheiros 'm1' e 'm2', e retornar os primeiros k bits do *hash* calculado. Para alcançar este objectivo foi criado um método, *getTrimmedHash()*, que retorna o resultado do *trim()* que recebe o método *getHash()*.
- O método *getHash()* é responsável por gerar o *hash* do conteúdo de um determinado ficheiro, recorrendo à função *hash* criptográfica SHA-1. O conteúdo do ficheiro é passado à função através de um byte[].

A partir do *hash* gerado, a função *trim()*, que recebe o byte[] do conteúdo do ficheiro e o número de bits a retornar, é responsável por obter apenas os primeiros k bits e retorná-los em forma de byte[].

**6.2.** Neste exercício, o objectivo consistia em alterar o ficheiro BadApp.java, de modo a que o seu *hash* fosse igual ao *hash* do ficheiro GoodApp.java, para isso foi adicionado um comentário no fim do ficheiro. De seguida inicia-se um ciclo em que se comparam os *hashes* de ambos os ficheiros e caso sejam diferentes adiciona-se um caracter aleatório ao comentário do BadApp.java, repetindo-se até serem iguais. O ficheiro é mantido em memória e apenas quando os *hashes* dos dois ficheiros são iguais é que o ficheiro alterado é guardado no disco.

No final é apresentada a média de tentativas até alcançar *hashes* iguais.

**6.3.** Neste exercício à semelhança do exercício 6.2 o objectivo é fazer com que os *hashes* dos dois ficheiros sejam iguais. A diferença é que neste exercício vão ser alterados ambos os ficheiros por cada iteração comparando depois com todos os *hashes* gerados ao longo da execução do programa, melhorando assim a performance do programa.

No final é apresentada a média de tentativas até alcançar hashes iguais.

**7.** Neste exercício recebe-se como parâmetro o nome do ficheiro, com a mensagem em claro ou cifrada, e a operação a realizar, cifra ou decifra.

No modo de cifra são necessários dois ficheiros de configuração, um para a cifra simétrica e outro para a cifra assimétrica, estando estes configurados no formato "chave:valor". O ficheiro *SYMConfiguration.txt* usado na cifra simétrica contém a primitiva, o modo de operação e modo de *padding*. O ficheiro *ASYMCipherConfiguration.txt* usado na cifra assimétrica contém a primitiva, o certificado folha e o certificado de confiança.

Neste modo começa-se por gerar uma chave simétrica, sendo esta usada para cifrar simetricamente o ficheiro recebido. Esta chave é mais tarde cifrada assimetricamente usando a chave do certificado folha, sendo necessário previamente validar o certificado, com recurso aos certificados intermédios e raiz.

Este modo produz o ficheiro cifrado e um ficheiro de meta dados contendo pela seguinte ordem: o tamanho do *IV*, o próprio *IV* e a chave cifrada.

No modo de decifra são necessários dois ficheiros de configuração, um para a decifra assimétrica e outro para decifra simétrica, estando estes configurados no formato "chave:valor". O ficheiro *ASYMDecipherConfiguration.txt*, usado na decifra assimétrica contém a primitiva, o *keyStore* e a primitiva da chave. O outro ficheiro, *SYMConfiguration.txt*, é o mesmo usado no modo cifra.

Neste modo começa-se por decifrar o ficheiro de meta dados, com recurso à chave privada pertencente ao mesmo par que a chave pública usada na cifra, contida no ficheiro .pfx correspondente ao certificado usado na cifra. Após a decifra do ficheiro de meta dados obtém-se a chave simétrica gerada na cifra, que será usada para decifrar o ficheiro cifrado recebido como parâmetro.

Este modo produz o ficheiro decifrado.

Os tempos de execução obtidos em milissegundos foram:

Tentativa	AES		Blowfish	
	Cifra	Decifra	Cifra	Decifra
1	516	532	531	516
2	469	594	500	500
3	453	547	531	515
4	500	578	453	531
5	453	531	437	500
Média	478.2	556.4	490.4	512.4