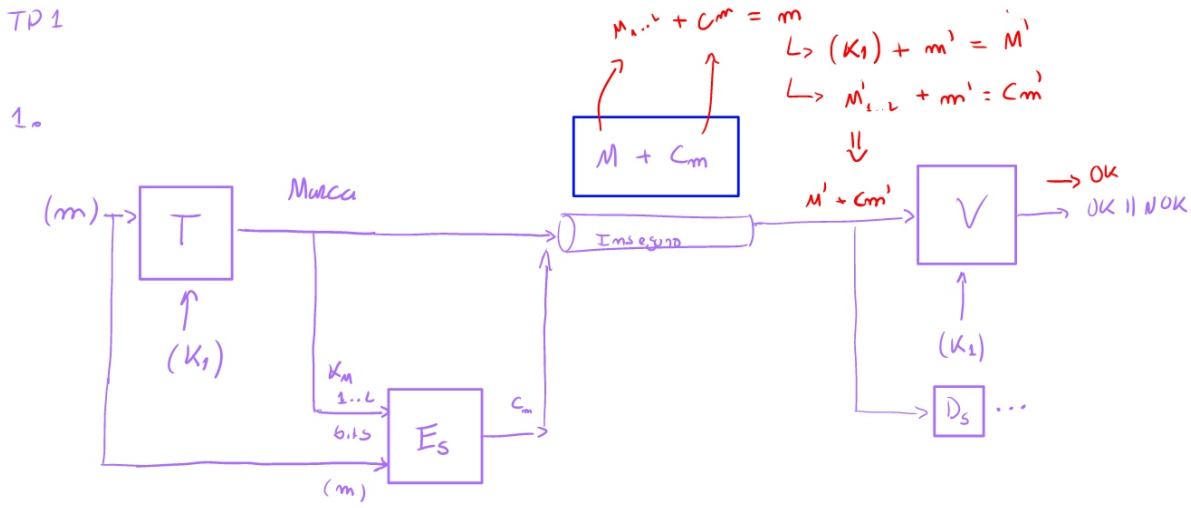
Segurança Informática - Primeiro Trabalho

1.

CI(mensagem) = T(K1) (mensagem) || Es (T(K1) (mensagem) (1..L) (mensagem)

T(k) -> Gerador de marcas (MAC)

Es(k) -> Algoritmo de cifra simétrico (Key com L bits)



Fazendo uso de um gerador de marcas é possível garantir a autenticidade da mensagem cifrada. Mas sendo usado um esquema simétrico para cifrar a mensagem, em que a codificação usa uma chave que pode ser facilmente obtida por um atacante ($M_{1..L}$), através da observação da marca criada anteriormente e que irá junto da mensagem cifrada no canal inseguro, consequentemente, é possivel a descodificação da mensagem antes de esta chegar ao destino.

Sendo que a chave para a geração da marca irá também ser usada para a verificação da autenticidade da mensagem, no outro lado do canal inseguro, qualquer pessoa que tenha ou ganhe acesso a essa chave, e que observe o canal inseguro, consegue fazer alterações na mensagem decifrada e voltar a gerar uma nova marca, antes de esta chegar ao seu destino, pondo em causa a autenticidade dessa mensagem, sem os comunicantes se aperceberem.

2.

Utilizar o modo de operação de CBC implica que a cifra vai ser realizada por uma primitiva que recebe e crifra *x* bytes de cada vez. Se restarem menos de *x* bytes num bloco para cifrar, os restantes bytes tem que ser preenchidos com algo para completar até *x* bytes.

Pelo contrário, em modo Counter, é gerada uma chave usando o IV e uma chave, que corresponde ao numero exato de bits da mensagem a ser cifrada, que irá ser passado por um XOR juntamento com a mensagem, nunca ocorrendo diferenças entre tamanho, não sendo necessário qualquer tipo de padding.

3.

Ao fazer uma troca de mensagens usando um esquema assimétrico implica verificar a autenticidade a partir da chave privada com a chave pública emitidas pelo emissor da mensagem. Sabendo que a cifra é um processso determinístico, o atacante que tem acesso ao canal inseguro e à chave pública, poderá gerar todas as cifras possíveis para uma mensagem de dominio suficiente pequeno e comparar com a cifra observada no canal inseguro, obtendo, por fim, a mensagem em claro.

4.

A classe *MAC* não necessita de um método *verify*, ao contrário da classe *Signature*, pois a cifra com autenticação *MAC* usa a mesma chave para gerar a marca e para verificar a marca. Neste caso, a chave é indicada na funçao *init* são usadas as funções *update* e *doFinal* para verificar a autenticidade da mensagem.
A autenticação por assinatura usa um esquema assimétrico, sendo que dessa forma é necessário ter dois métodos distintos para cada operação: assinar e verificar assinatura.

5.

5.1.

Nos casos em que não sejam *root certificates*. Para cada certificado que não seja raíz é necessário verificar o mesmo a partir do *issuer* (emissor). Esta verificação é feita a partir da chave pública do certificado emissor sendo possível validar a partir da chave privada (que corresponde à chave pública) do certificado a validar. Esta cadeira realiza-se recursivamente até se chegar ao certificado raíz.

5.2

Na cadeia de verificação de certificados, as *basic constraint*'s são usadas para indicar o tipo de certificado (*end-entity* ou *CA*) e o tamanho da cadeia/caminho (*PathLength*).
Os certificados do tipo *end-entity* são certificados assinados por uma *Certificate Authority* com o objetivo de ser usado por um utilizador, servidor, sistema, entre outros.
Neste caso, se a aplicação ignorar esta extensão, pode não conseguir validar a autenticidade da mensagem por não chegar a uma raíz de confiança ou até tentar assinar para além de certificados de confiança.
Numa situação em que um atacante consegue gerar um certificado de confiança e inseri-lo no meio da cadeia, o verificador da cadeia irá aceitar esse certificado porque está a violar as *basic constraint*'s dos outros certificados.

5.3

Os ficheiros *.cer* contêm os certificados X.509. Para verificar a assinatura dos certificados é necessário a chave pública do seu emissor. Estes ficheiros normalmente são fornecidos por uma organização de confiança (*CA*).

Os ficheiros *.pfx* contém o par de chave pública e privada de um *CA* ou *end-entity*. Neste caso a verificação do certificado é feito usando a chave privada em conjunto com a chave pública.

6.

6.1.

Ficheiro: *encGen*

rand usa *srand* como *seed*, quando não se chama *srand* o valor 1 é usado como *default*. Ou seja, a sequência vai ser sempre a mesma. É necessário dar uma *seed* que não se repita ou que seja única.

Quando se dá feed com *time* continua a ser um problema porque o atacante só precisa de saber o tempo que foi usado para a *seed*.

6.2.

Descrição

Este programa serve para encontrar a chave usada para cifrar o documento.

Ficheiros: *keyGen* e *keyGuess*

Utilização

keyGen

`keyGen <start time> <end time> [key file]`

De acordo com o Lab, os valores são: *start time = 1524013729* e *end time = 1524020929*

Exemplo:
`keyGen 1524013729 1524020929 keys`

Faz gerar o ficheiro *keys* com as chaves geradas a partir do algoritmo usado.

keyGuess

`java -jar keyGuess.jar <key file>`

7.

Descrição

Programa cifra mensagem usando cifra simétrica e cifra a chave gerada com cifra assimétrica usando certificados, formando assim uma cifra híbrida.

Ficheiro: HybridCipher

Utilização

Cifrar

`java -jar HybridCipher.jar -enc <data file> <certificate file> [cipher output file] [key output file] [iv]`

Exemplo: `java -jar HybridCipher.jar -enc files/input/LoremIpsum.txt files/certs/end-entities/Bob_1.cer files/output/LoremIpsum.cipher files/output/LoremIpsum.key 1234567890`

Decifrar

`java -jar HybridCipher.jar -dec <ciphered file> <ciphered key file> <certificate private file> [output file] [iv] [private key password]`

Exemplo: `java -jar HybridCipher.jar -dec files/output/LoremIpsum.cipher files/output/LoremIpsum.key files/certs/pfx/Bob_1.pfx files/output/LoremIpsum.decipher 1234567890 changeit`