Guiões das Aulas Práticas

Segurança Informática

Licenciatura em Tecnologias da Informação

Hélder Gomes

Escola Superior de Tecnologia e Gestão de Águeda
Universidade de Aveiro

2018–2019

Conteúdo

2

Criptografia em Java

Resumo:

- Cifras simétricas e assimétricas com Java.
- Funções de síntese em Java.

2.1 Introdução

Para a realização deste trabalho é necessário ter o (*Java Developement Kit* (JDK) instalado no seu computador. Pode usar o JDK do OpenJava (em Linux, instalar com o comando apt-get) ou o da Oracle, que pode obter na página oficial de distribuição da versão SE (*Standard Edition*) do Java[1]. Poderá também ter instalado um IDE da sua preferência, como o NetBeans[2], por exemplo.

Na realização deste trabalho é necessário visualizar/alterar o conteúdo de ficheiros em binário. Para esse efeito, e se estiver em ambiente Linux, pode instalar a aplicação `GHex`, disponível no repositório de aplicações do Ubuntu.

2.2 Criptografia de chave simétrica

2.2.1 Algumas classes e métodos importantes

As seguintes classes que aqui se apresentam são fundamentais para o desenvolvimento de aplicações envolvendo criptografia. Pode encontrar informação detalhada no Security Developers Guide[3] , no site da Oracle

`KeyGenerator`

A classe `KeyGenerator`, do pacote javax.crypto, implementa um gerador de chaves simétricas. Alguns método importantes desta classe são: `getInstance` e `generateKey`. Use o JavaDoc para obter informação sobre a classe e os métodos indicados.

`SecretKey`

A classe `SecretKey`, do pacote javax.crypto, implementa uma chave simétrica. Um método importante desta classe é o `getEncoded`. Use o JavaDoc para obter informação sobre a classe e o método indicado.

`SecretKeySpec`

A classe `SecretKeySpec`, do pacote javax.crypto.spec, permite criar uma chave simétrica a partir de um array de bytes (byte []). Use o JavaDoc para obter informação sobre ela.

[1] `http://www.oracle.com/technetwork/java/javase/downloads/index.html`

[2] `http://www.netbeans.org`

[3] `https://docs.oracle.com/en/java/javase/11/security/java-cryptography-architecture-jca-reference-guide.html`

`Cipher`

A classe `Cipher`, do pacote javax.crypto, permite realizar operações de cifra e decifra. Alguns métodos importantes desta classe são: `getInstance`, `init` e `doFinal`. Use o JavaDoc para obter informação sobre a classe e os métodos indicados.

2.2.2 Geração de uma chave simétrica

Desenvolva um pequeno programa para gerar uma chave simétrica para o algoritmo DES e guardá-la num ficheiro cujo nome é fornecido como parâmetro de entrada.

2.2.3 Cifra de um ficheiro utilizando o algoritmo DES

Crie um programa para cifrar o conteúdo de um ficheiro usando o algoritmo de cifra DES. O programa deve ter os seguintes parâmetros de entrada: (i) o nome do ficheiro a cifrar, (ii) o nome pretendido para o ficheiro cifrado e (iii) o nome do ficheiro com a chave simétrica a usar.

Nota: use apenas o nome do algoritmo na instanciação do objecto Cypher, por exemplo "DES". Internamente corresponde a usar o algoritmo DES em conjunto com o modo de cifra ECB e o padding PKCS#5. Os modos de cifra serão explorados numa secção mais adiante.

2.2.4 Decifra de um ficheiro utilizando o algoritmo DES

Desenvolva um novo programa para fazer a decifra do conteúdo de um ficheiro cifrado utilizando o algoritmo DES. O programa deve ter os seguintes parâmetros de entrada: (i) o nome do ficheiro para decifrar, (ii) o nome pretendido para o ficheiro depois de decifrado e (iii) o nome do ficheiro com a chave a utilizar na operação de decifra.

2.2.5 Cifra e decifra de um ficheiro utilizando o algoritmo AES

Altere os dois programas anteriores para que o algoritmo também seja passado como um parâmetro. Um dos objetivos desta alteração é permitir a utilização do algoritmo AES. Tenha no entanto em atenção que ao contrário do que sucede com o DES, o AES permite vários tamanhos de chave, nomeadamente 128, 192 e 256 (sujeito a restrições de exportação), pelo que

também deve permitir a sua indicação e ter o cuidado de gerar a chave de acordo.

2.3 Modos de Cifra

2.3.1 Vetor inicial

Como sabe, os modos de cifra com realimentação (CBC, OFB e CFB) utilizam um vetor inicial. Quando se usa uma instância da class `Cipher`, para cifrar num dos modos de cifra atrás indicados, ela gera um vetor inicial aleatório (alternativamente podemos ser nós a fornecê-lo), que pode ser lido e que tem que ser fornecido para a realização da correspondente operação de decifra. Além disso, deve-se indicar qual o processamento a dar ao último bloco caso ele não esteja completo (padding). Esta informação é fornecida em conjunto com o algoritmo numa string com a forma "AlgorithmName/CipherMode/Padding", como no seguinte exemplo "AES/CBC/PKCS5Padding"

Altere o programa de cifra desenvolvido em **??** para que receba como parâmetro de entrada, além dos atuais, (i) o algoritmo/modo de cifra/padding a utilizar e (ii) o nome do ficheiro para guardar o vetor inicial utilizado na operação de cifra. Note que apenas deve utilizar o vetor inicial para os modos de cifra acima indicados.

Altere também o programa que desenvolveu em **??** para acrecentar os seguintes parâmetros de entrada: (i) o nome do algoritmo/modo de cifra/-padding a utilizar e (ii) o nome do ficheiro que contém o vetor inicial a utilizar na operação de decifra. Atenção que o vetor inicial apenas é utilizado para os modos de cifra CBC, OFB e CFB.

Sugestão: considera a utilização a classe `IvParameterSpec`.

2.3.2 Reprodução de padrões

Neste exercício vamos analisar o impacto dos modos cifra ECB e CBC na reprodução de padrões existentes num documento em claro. Para esse efeito vamos cifrar um ficheiro BMP e depois visualizá-lo e compará-lo com o ficheiro em claro. No entanto, para se poder visualizar o ficheiro cifrado, o seu cabeçalho não pode estar cifrado, pelo que temos que repor os primeiros 54 octetos a partir do ficheiro original.

Com o seu programa de cifra, e utilizando o modo de cifra ECB com um algoritmo de cifra simétrica à sua escolha, cifre o ficheiro `pic_original.bmp` que lhe foi fornecido. Utilizando um editor de ficheiros em binário (`GHex`, por exemplo) copie os primeiros 54 octetos do ficheiro em claro para o ficheiro

cifrado que obteve. Alternativamente, poderá usar a seguinte sequência de comandos Unix, assumindo que `pic_cifrada.bmp` é o ficheiro produzido pela cifra e `pic_cifrada_1.bmp` é o ficheiro onde pretende combinar o cabeçalho original com o conteúdo cifrado:

```
dd if=pic_original.bmp ibs=1 count=54 > pic_cifrada_1.bmp
dd if=pic_cifrada.bmp ibs=1 skip=54 >> pic_cifrada_1.bmp
```

Abra os ficheiros em claro e cifrado, utilizando um programa para visualizar imagens e compare o seu conteúdo. O que observa?

Cifre novamente o ficheiro `pic_original.bmp`, agora utilizando o modo de cifra CBC e o mesmo algoritmo de cifra que utilizou atrás. Repita as operações indicadas no parágrafo anterior para poder visualizar o ficheiro cifrado. Analise os ficheiros em claro e cifrado. O que observa?

Repita a experiência para os mesmos modos de cifra e para vários algoritmos de cifra. O que conclui?

2.3.3 Corrupção de um criptograma

Neste exercício vamos analisar o impacto que a ocorrência de erros no criptograma induz no correspondente texto em claro, para os modos ECB, CBC, OFB e CFB.

Com o seu programa de cifra, utilizando o modo de cifra ECB com um algoritmo de cifra à sua escolha (AES, por exemplo), cifre o ficheiro `pic_original.bmp` que lhe foi fornecido. Utilizando um editor de ficheiros em binário (`GHex`, por exemplo), altere um bit a um dos octetos do ficheiro cifrado, por exemplo ao octeto na posição 60H.

Decifre o ficheiro cifrado no qual introduziu o erro, utilizando o seu programa de decifra com o mesmo modo e algoritmo de decifra que usou para cifrar o ficheiro no parágrafo anterior. Como o `GHex`, abra o ficheiro original (`pic_original.bmp`) e o ficheiro em claro produzido no passo anterior. Analise as diferenças entre eles e tire conclusões quanto ao impacto do erro introduzido.

Repita a experiência para os restantes modos de cifra e, para cada um deles, retire conclusões quanto ao impacto do erro de um bit no criptograma. Tente perceber, igualmente, quais são os modos de cifra mais e menos sensíveis a erros no criptograma (em termos de aparecimento de erros no resultado da decifra).

2.3.4 Alinhamento com excipiente (*padding*)

Como sabe, um algoritmo de cifra por blocos, apenas cifra textos de dimensão fixa e igual ao tamanho do bloco. No entanto, dificilmente o texto que se pretende cifrar tem uma dimensão múltipla do tamanho do bloco do algoritmo que se pretende usar. Para resolver este problema, faz-se o alinhamento com excipiente (*padding*) do último bloco do texto a cifrar, ou seja, acrescentam-se octetos até perfazer a dimensão do bloco. Estes octetos introduzidos são depois retirados após a operação de decifra.

Existem vários padrões para o alinhamento cm excipiente, sendo um deles indicado pelo PKCS#5. Neste exercício pretende-se que demostre que existe esse alinhamento e que o mesmo obedece ao padrão PKCS#5.

Utilizando o `GHex` e os seus programas de cifra e de decifra, em modo ECB com um algoritmo de cifra à sua escolha e com o alinhamento PKCS#5, idealize uma experiência que mostre como é que o alinhamento do PKCS#5 é efetuado.

Sugestão: use NoPadding na decifra

2.4 Cifra Contínua

2.4.1 Reutilização de chaves

Com base no programa para cifrar um ficheiro que desenvolveu em **??**, crie um outro programa que tenha os mesmos parâmetros de entrada, mais um para receber um vector inicial. Este programa deve cifrar o conteúdo de um ficheiro indicado como parâmetro, usando a transformação, a chave e o vector inicial contidas nos correspondentes ficheiros indicados como parâmetro, e guardar o criptograma produzido num ficheiro cujo nome também é fornecido como parâmetro.

Crie um outro programa que faça o XOR do conteúdo de dois ficheiros cujos nomes devem ser fornecidos como parâmetros e gurade o resultado num outro ficheiro cujo nome deve também ser fornecido como parâmetro. O XOR deve ser feito byte a byte. Caso os ficheiros sejam de tamanho diferente, a saída deve conter o XOR de todos os bytes do ficheiro menor com os correspondentes bytes do ficheiro maior, mais os bytes restantes do ficheiro maior.

Crie dois ficheiros com duas mensagens diferentes para cifrar (e.g. mensagem1.txt e mensagem2.txt).

Gere uma chave simétrica (e.g., para o algoritmo AES). Usando esta chave, cifre o primeiro ficheiro, usando uma cifra contínua (e.g., usando a transformação AES/OFB/NoPadding) e guarde o criptograma resultante

(e.g., no ficheiro mensagem1.cripto) e o vector inicial usado. Cifre o segundo ficheiro usando a mesma chave e o mesmo vector inicial (com o programa de cifra que criou acima) e guarde o criptograma resultante num outro ficheiro (e.g., mensagem2.cripto).

Usando o programa para fazer o XOR de dois ficheiros, faça o XOR dos dois criptogramas atrás obtidos e guarde o resultado num novo ficheiro (e.g., cripto.xor).

Faça agora o XOR do criptograma da primeira mensagem (mensagem1.cripto) com o resultado do XOR anterior (cripto.xor) e guarde o resultado num novo ficheiro (e.g., resultado.txt).

Abra o ficheiro que obteve (resultado.txt) e analise o seu conteúdo. O que conclui?

2.4.2 Manipulação do criptograma

Crie um ficheiro (e.g., mensagem.txt) com a seguinte mensagem: "Mensagem muito secreta!".

Cifre o ficheiro usando uma cifra contínua (e.g., usando a transformação AES/OFB/NoPadding) e guarde o criptograma resultante num ficheiro (e.g., mensagem.cripto).

Usando um editor de ficheiros em binário (e.g., GHex), altere um bit apenas ao criptograma e salve o ficheiro. Decifre o criptograma alterado e compare o resultado com o texto original (mensagem.txt). O que conclui?

Consegue alterar o criptograma de forma a que o resultado da operação de decifra seja a mensagem: "Mensagem pouco secreta!"?

2.5 Password Based Encription

Dada a dificuldade dos humanos em lidar com chaves, frequentemnte usa-se criptografia com base numa password (PBE - *Password Based Encryption*, i.e. a chave simétrica a usar é derivada a partir de uma password e de um sal (*salt*)

O seguinte exemplo ilustra a geração de uma chave AES com base numa password:

```
SecretKeyFactory factory = SecretKeyFactory
       .getInstance("PBKDF2WithHmacSHA1");
KeySpec keySpec = new PBEKeySpec(password.toCharArray(),
       salt, 65536, 256);
SecretKey secretKey = factory.generateSecret(keySpec);
SecretKey secret = new SecretKeySpec(secretKey.getEncoded(), "AES");
```

Desenvolva dois programas que usam AES e password based encryption, sendo um para cifrar e o outro para decifrar. Considere que os dados para cifrar e decifrar estão guardados em ficheiros e que os resultados das operações são também guardados em ficheiro. Guarde também em ficheiro todos os dados resultantes da operação de cifra necessários para a operação de decifra.

2.6 Funções de Síntese

Crie um programa para obter a síntese (*one-way, cryptographic hash*) de um texto fornecido sob a forma de um ficheiro. O programa deverá ter os seguintes parâmetros de entrada: (i) o nome do ficheiro com os dados dos quais se pretende calcular a síntese, (ii) o nome da função de síntese a usar (MD5, SHA-1, SHA-256, SHA-384 ou SHA-512), e (iii) um parâmetro opcional que é o nome de um ficheiro onde guardar a síntese obtida. Como este terceiro parâmetro é opcional, o programa deve sempre mostrar a síntese obtida no ecrã (com a forma de número hexadecimal).

Sugestão: use as classes `MessageDigest` e `DatatypeConverter`.

2.6.1 Efeito de avalanche ou difusão

Um requisito importante para as funções de síntese é que uma pequena alteração no texto deve produzir uma síntese completamente diferente – efeito de avalanche ou difusão. Vamos verificar esta propriedade.

Coloque os dados dos quais pretende calcular a síntese num ficheiro. Utilizando o seu programa para calcular funções de síntese, obtenha a síntese do texto que colocou no ficheiro para várias funções de síntese. Guarde as sínteses obtidas em ficheiros. Verifique os comprimentos das sínteses obtidas para cada uma das funções de síntese.

Altere em apenas um bit o conteúdo do ficheiro com o qual gerou as sínteses anteriores. Gere novas sínteses, utilizando o ficheiro com a alteração de um bit, e para as mesmas funções de síntese. Para cada uma das funções de síntese, compare a síntese agora obtida com a síntese obtida a partir do ficheiro antes de alterar o bit.

2.6.2 Análise estatística de sínteses de mensagens semelhantes

Crie um novo programa baseado no que calcula as sínteses, mas cujo resultado deverá ser a distribuição estatística das diferenças entre as sínteses calculas

sobre um conjunto de N mensagens que diferem de uma inicial em apenas um bit. O programa deverá receber como parâmetros o nome de um ficheiro com o conteúdo inicial relativo ao qual se pretende calcular uma síntese e o valor de N. Cada uma dos N conteúdos alternativos deverá ser uma cópia do inicial na qual foi mudado apenas um bit aleatório. Para escolher esse bit use um gerador de números aleatórios para obter o seu índice. Após calcular a síntese de cada conteúdo modificado, avalie a diferença entre o mesmo e o da síntese original em número de bits diferentes (distância de Hamming em bits). Comente a distribuição das diferenças de bits entre as sínteses obtidas.

Sugestão: use as classes `Random` e use a operação XOR para detetar bits alterados em cada octeto (ficam com o valor 1).

2.7 Código de Autenticação de Mensagens (MAC - Message Authentication Code)

Crie um programa para obter o MAC (*Message Authentication Code*) de um texto fornecido sob a forma de um ficheiro. O programa deverá ter os seguintes parâmetros de entrada: (i) o nome do ficheiro com os dados dos quais se pretende calcular o MAC, (ii) o nome da função de MAC a usar (HmacMD5, HmacSHA1, HmacSHA256, HmacSHA384, HmacSHA512), (iii) Nome do ficheiro com a chave a usar, e (iv) o nome de um ficheiro onde guardar o código MAC obtido. O programa deve sempre mostrar no ecrã o código MAC obtido (em hexadecimal ou base64).

Usando um mesmo algoritmo de MAC, compare os códigos MAC que obtem quando calculados usando a mesma chave e mensagens que diferem entre si em apenas um bit.

Usando um mesmo algoritmo de MAC, compare os códigos MAC que obtem quando calculados sobre uma mesma mensagem usando diferentes chaves.

Sugestão: use a classe `javax/crypto/Mac`

2.8 Criptografia de chave assimétrica

2.8.1 Geração de um par de chaves

Desenvolva um pequeno programa para gerar uma par de chaves assimétricas para o algoritmo RSA e para um comprimento de chave especificado como parâmetro de entrada (1024, 2048, 3072 ou 4096). O programa deve também

guardar o par de chaves em dois ficheiros (um para a chave privada e outro para a chave pública), cujos nomes são fornecidos como parâmetro de entrada.

Sugestão: use as classes `KeyPairGenerator`, `KeyPair`, `PrivateKey` e `PublicKey`

2.8.2 Cifra com o algoritmo RSA

Desenvolva um pequeno programa para cifrar o conteúdo de um ficheiro utilizando o algoritmo RSA. O nome do ficheiro com o conteúdo a cifrar deve ser passado como parâmetro de entrada, bem como o ficheiro com a chave pública a utilizar para realizar a cifra, e o nome do ficheiro onde guardar o criptograma obtido.

Sugestão: use as classes `KeyFactory`, `Cipher` e `X509EncodedKeySpec`. Tenha em atenção o tamanho do ficheiro a cifrar. Por exemplo, utilizando uma chave RSA de 1024 bits (128 octetos), não pode cifrar blocos (de ficheiros) com mais de 117 octetos.

2.8.3 Decifra com o algoritmo RSA

Desenvolva um pequeno programa para decifrar o conteúdo de um ficheiro, a fornecer como parâmetro de entrada, utilizando o algoritmo RSA. Deve também fornecer como parâmetro de entrada o ficheiro que contém a chave privada a utilizar para a operação de decifra e o nome do ficheiro onde se deve salvaguardar a mensagem decifrada.

Sugestão: use as classes `KeyFactory`, `PrivateKey`, `Cipher` e `PKCS8EncodedKeySpec`,

2.8.4 Cifras sucessivas do mesmo texto produzem o quê?

Utilizando o programa de cifra assimétrica que desenvolveu no passo **??**, cifre duas vezes um mesmo texto, utilizando a mesma chave, e guarde os criptogramas gerados em dois ficheiros diferentes. No `GHex`, abra os dois ficheiros com os criptogramas gerados e compare-os. O que conclui?

Use agora o programa de decifra que desenvolveu no passo **??** e decifre os ficheiros com os criptogramas para dois ficheiros diferentes. Compare os ficheiros com o resultado da decifra. O que conclui? Consegue explicar o porquê do que observou?

2.8.5 Como cifrar um ficheiro grande?

Como já foi referido, utilizando RSA e uma chave de 1024 bits, não consegue cifrar textos com uma dimensão superior a 117 octetos. Imagine que possui

uma chave pública que pertence a uma determinada pessoa e que lhe pretende enviar um ficheiro grande, por exemplo 100 MOctetos, de modo que apenas essa pessoa o possa decifrar. Não tem forma de contactar com essa pessoa de forma a previamente combinar uma chave simétrica partilhada, pelo que vai ter que usar obrigatoriamente a chave pública RSA dela, embora possa também usar outros algoritmos de cifra. Dividir o ficheiro em blocos de 117 octetos e cifrar cada um deles com a chave pública está fora de questão porque ia ser uma operação demorada. Consegue indicar uma forma de enviar o ficheiro, cifrado de forma eficiente, e de tal modo que apenas o dono da chave pública o possa decifrar?

2.9 Bibliografia

Java Cryptography Architecture (JCA) Reference Guide, `https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html`

The Java Tutorials: Security Features in Java SE, `http://docs.oracle.com/javase/tutorial/security/index.html`