Aulas TP de SSI
Criptografia Aplicada

SDC — MEI

Ano Lectivo de 2014/15

1 Aulas 1 e 2

Estes projectos deverão ser realizados por grupos de 2. O código deverá ser enviado por e-mail ao responsável pela disciplina à medida que as tarefas forem concluídas. A submissão regular de soluções com qualidade será um dos parâmetros da avaliação TP. Quaisquer dúvidas podem ser esclarecidas por e-mail.

1.1 Ambiente de Desenvolvimento

O objectivo principal desta alínea é o de escolher/instalar o ambiente de desenvolvimento Java que será utilizado durante o curso. Pretende-se também recordar os princípios básicos da programação em Java, bem como a implementação de aplicações distribuídas utilizando sockets.

Como actividade de programação (para experimentar o ambiente escolhido), deve desenvolver uma pequena aplicação, composta por dois comandos executáveis na consola.

- Esses comandos deverão ser implementados na forma de duas classes Java: a classe Cliente e a classe Servidor.
- A aplicação deverá permitir a um número arbitrário de invocações da aplicação Cliente comunicar com um Servidor que escuta num dado `port` (e.g. 4567).
- O servidor atribuirá um número de ordem a cada cliente, e simplesmente fará o `dump` do texto enviado por cada cliente (prefixando cada linha com o respectivo número de ordem).
- Quando um cliente fecha a ligação, o servidor assinala o facto (e.g. imprimindo `=[n]=`, onde `n` é o número do cliente).
- A aplicação Cliente deverá permitir ao utilizador inserir do teclado as mensagens a enviar para o Servidor.

1.2 Cifra de ficheiro utilizando JCA/JCE

Pretende-se cifrar o conteúdo de um ficheiro. Para tal far-se-á uso da funcionalidade oferecida pela JCA/JCE, em particular a implementação de cifras simétricas.
O objectivo é então o de definir um pequeno programa Java que permita cifrar/decifrar um ficheiro utilizando a cifra simétrica RC4. A sua forma de utilização pode ser análoga a:

```
prog -genkey <keyfile>
prog -enc <keyfile> <infile> <outfile>
prog -dec <keyfile> <infile> <outfile>
```

Sugestões:

- Para simplificar, pode começar por utilizar uma chave secreta fixa, definida no código na forma de um array de bytes. Nesse caso, deverá utilizar a classe `SecretKeySpec` para a converter para o formato adequado.
- É também interessante verificar que o criptograma gerado é compatível com outras plataformas que implementam a mesma cifra. O comando seguinte utiliza o `openssl` para decifrar um ficheiro cifrado com `RC4` (a chave tem de ser fornecida em hexadecimal).

  ```
  openssl enc -d -rc4 -in <infile> -out <outfile> -K <chave>
  ```

Algumas classes relevantes:

- javax.crypto.Cipher
- javax.crypto.KeyGenerator
- javax.crypto.SecretKey (interface)
- javax.crypto.spec.SecretKeySpec
- javax.crypto.CipherInputStream
- javax.crypto.CipherOutputStream

1.3 Implementação da cifra RC4

Esta alínea representa um projecto extra, para aqueles alunos com interesse em perceber melhor a forma de funcionamento das cifras sequenciais, bem como os desafios colocados pela implementação de software criptográfico.

Pretende-se implementar de raiz a cifra RC4 e comprovar que a implementação realizada é compatível com as implementações comerciais da cifra.

Para isso, a classe desenvolvida na questão anterior deverá ser adaptada, por forma a que as chamadas à API do Java para efectuar a cifragem/decifragem sejam substituídas por chamadas a funções desenvolvidas especificamente para esse efeito.

Os detalhes funcionamento da cifra RC4 pode encontrar-se nos slides da disciplina e também em múltiplas referências on-line.

A verificação da correcção da implementação poderá ser feita testando a sua compatibilidade com a classe desenvolvida na alínea anterior, ou directamente com o `openssl`.

2 Aula 2

Pretende-se nesta aula modificar as classes Cliente e Servidor desenvolvidas nas aulas anteriores por forma a garantir a confidencialidade nas comunicações estabelecidas. Pretende-se ainda experimentar o impacto da escolha da cifra/modo na comunicação entre o cliente/servidor. Para tal é conveniente reforçar a natureza interactiva da comunicação modificando os ciclos de leitura/escrita para operarem sobre um byte de cada vez:

Cliente

```
CipherOutputStream cos = ...
int test;
while((test=System.in.read())!=-1) {
     cos.write((byte)test);
     cos.flush();
}
```

Servidor

```
CipherInputStream cis = ...
int test;
while ((test=cis.read()) != -1) {
      System.out.print((char) test);
}
```

Experimente agora as seguintes cifras (e modos) e verifique qual o respectivo impacto nas questões de buffering e sincronização:

- RC4
- AES/CBC/NoPadding
- AES/CBC/PKCS5Padding
- AES/CFB8/PKCS5Padding
- AES/CFB8/NoPadding
- AES/CFB/NoPadding

Procure explicar a diferenças detectadas na execução da aplicação.
Note que em muitos dos modos sugeridos necessita de considerar um IV. Considere para o efeito que o IV é gerado pelo cliente e enviado **em claro** para o servidor (no início da comunicação).
Algumas classes relevantes (para além das já estudadas...):

- javax.crypto.spec.IvParameterSpec
- java.security.SecureRandom