Aulas TP de Aplicações Criptograficas

MSI

Ano Lectivo de 2019/20

1 Aulas 1 e 2: Infra-estrutura base

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. Pode ser utilizada qualquer linguagem de programação que ofereça uma API criptográfica semelhante às disponíveis em Java, C#, etc. 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 que será utilizado durante o curso. Pretende-se também recordar os princípios básicos da programação de aplicações distribuídas, multi-thread, 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: o comando Cliente e o comando 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 (colocando como prefixo em cada linha 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

Pretende-se cifrar o conteúdo de um ficheiro. Para tal far-se-á uso da funcionalidade oferecida pela API criptográfica, em particular a implementação de cifras simétricas.
O objectivo é então o de definir um pequeno programa que permita cifrar/decifrar um ficheiro utilizando a cifra simétrica AES-CTR. 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.
- 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 AES-CTR (a chave tem de ser fornecida em hexadecimal).

  ```
  openssl enc -d -aes-128-ctr -in <infile> -out <outfile> -K <chave>
  ```

Se optar pelo ambiente Java, aqui estão 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 AES-CTR

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 AES-CTR 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 AES-CTR pode encontrar-se 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 3 - Confidencialidade na comunicação Cliente-Servidor

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. Por exemplo em Java o código poderia ser qq coisa como:

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).

Se optar pela API do Java, as seguintes classes são relevantes:

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

3 Aula 4 - Autenticação do canal

Pretende-se complementar o programa com a autenticação das mensagens cifradas trocadas entre Cliente e Servidor.

Neste sentido, as aplicações devem derivar duas chaves simétricas a partir da chave de sessão k acordada. Para isso devem calcular $k_1 = H(k,'1')$ e $k_2 = H(k,'2')$, sendo H uma função de hash criptográfica. Chave k_1 será utilizada para parametrizar a cifra simétrica.

Todos os criptogramas enviados deverão ser acompanhados de um MAC (sugere-se a utilização do algoritmo HMAC) parametrizado com a chave k_2. O MAC deve ser calculado sobre o criptograma e um número de sequência que não tem de ser transmitido e é mantido independentemente por cada um dos intervenientes.

Se optar pela API do Java, as seguintes classes são relevantes:

- java.security.MessageDigest
- javax.crypto.Mac

4 Aula 5 - Protocolo Diffie-Hellman

O objectivo desta aula é implementar o acordo de chaves Diffie-Hellman. Algumas sugestões para atacar o problema: Comece por codificar cada passo do protocolo explicitamente. Pode começar por utilizar os seguintes parâmetros para o grupo:

```
-----BEGIN DSA PARAMETERS-----
MIIELQKCAgEApS/ayPLHJ/quoBmHtamBtb4NKYQXGoboG8aRQJjhxU7ezkKFFSgz
CeA4JhuVUtP5yvWAxz7OvZc2sX8zFleilK3YS8GmIdh/TfiMSnApTC8hh2a1WbQC
QQMQbDBUlk9jN7P4pjAp4qu4nhPVJxXONyW2lwQze+c+nGcfhTGJgct7Uys46T2X
ped9jM8nydPL940Hj/prB4ttZ74AKRYRC6h6y2uKi77MZrtHOTgxBeRFyKIug6Oq
HDBzibhMbjxtMAQH4mg8VOxDXxqjJomw0chv9HHYyIGhuus6EHvvvuRKyvBJn0dR
RCyI1ktAV+UNobtYUPvJzXLbBd6JzQ2n8lKduq2bsGccSQKUkEnShsE/95IUSAC3
2q3cylb/fR2k06mRlEP3ByiJooxGu/ZMqUqhURkqfCDBsaWC1zzAAJ4pg8tXVLsY
pYUNjjWPXo+5XCDgnIcVleRSLCWbRG1tTXL/8NdKWdolzkKciRXjeUIcmakJoEdo
UoqbzDKciHBVNtBu58UXJSmMyZddMIT1LSM3BMyTkD7vFvhso+5rgd273Pbsnee8
F8snLY+bnRoPHqrzH6NGmqSZQROo+PSTzTmi4N6Lkz9QgXkpw0CUQhZKlQvfnkMi
MLanxgFi1a5OmCzVoGJz6oYDeYobZntdHx+AoZqel3OS36+Fw9QoDpsCIQCOC52I
EjPMN2565kI7CPJrx0zP0OiuTNlsj2fZzIFq8QKCAgEAhK0wHjsHe1s8BnOZ5wUK
F7uUBEuVAqzXxw+M5wYMGgaf5Ybb2O2hwxO84wA8ILVl0CSTbYTEL1Ah3YCNX1kD
wXL86xEc/NVT0thYkrU+N2VCy7w8UFmIdLUG0iLxC37CQX8HXbOJkirOAzlVs7Kd
+G6H4WCXET74I9bayJDskl4BAPikM+4tZ24nD7cFFwoCLOv51DscIjmc8KkB5NW+
lSstuzp6KUCPJ8dvN8kI96z1xRUK3x/4Stya53TswXZROu7lDuvHYTOcMIgkGsLX
7sSO9PD6Akn36mT62mSQzayJapLpPVnnLuOcXE9al/6+fw2+tvMmSb8rJmrKqxZc
hwCrUVQzfOcgiRk5U5ZM9APxwKNLZ+6A5iKd4VxXDW82Y6eAPCGQFVI2TB9wKLr+
Q+Gr6nd115zcM40R8QIAIRXf8ztAPQJKS2c7jbJMfQYzjg4Xc7Tx358QIvKFqo+R
950BIZyo+SoOJij2d01yNrjg1GC5c2d9GSV/aahfrruNhnYuJOOokf7Qo3kngo3u
Afzz3O8B8DhGrwFU0aVUYuyl50Tq3HsPfj2gIowquBDudFlW/p25V4gX4Ykp6Mnz
hUk0Ai2lhwsoA0ERlGxbmJvYIRHcuHzyAUhCQUVBu9M31ejXGOui+idMecJsOKOZ
i9709w0usRKlXYC7y05Mwtc=
-----END DSA PARAMETERS-----
```

Para recuperar os parâmetros p, q e g deve utilizar o `openssl dsaparam`. Estes parâmetros são públicos e devem estar fixos de ambos os lados da comunicação. Recorde que, cada parceiro calcula o seu parâmetro público como $g^s \pmod p$, em que s é um valor amostrado de forma uniforme na gama $0..q-1$.

Depois altere o seu programa Cliente-Servidor para que a pré-chave, ou seja o input da função de derivação de chaves para cifra e MAC, seja agora o resultado do acordo Diffie-Hellman.

5 Aula 6 - Codificação do protocolo Station-to-Station

Pretende-se complementar o programa com o acordo de chaves Diffie-Hellman para incluir a funcionalidade do protocolo Station to Station. Recorde que nesse protocolo é adicionado uma troca de assinaturas (cifrada com a chave de sessão negociada K), i.e.:

$$Alice \rightarrow Bob \quad : \quad g^x \tag{1}$$
$$Alice \leftarrow Bob \quad : \quad g^y, E_K(Sig_B(g^y, g^x)) \tag{2}$$
$$Alice \rightarrow Bob \quad : \quad E_K(Sig_A(g^x, g^y)) \tag{3}$$

Um requisito adicional neste protocolo é a manipulação de pares de chaves de cifras assimétricas (e.g. RSA). Para tal deve produzir um pequeno programa que gere os pares de chaves para cada um dos intervenientes e os guarde em ficheiros que serão lidos pela aplicação Cliente/Servidor.

6 Aula 8 - Utilização de Certificados X.509

Nesta sessão iremos substituir as chaves de assinatura RSA geradas de forma adhoc no guião anterior por chaves geradas por uma CA, com os correpondentes certificados de chave pública.

Para já iremos assumir que Cliente e Servidor conhecem de antemão os certificados de chave pública um do outro e que acreditam na autenticidade desses certificados.

As chaves e certificados estão aqui: `https://moodle.up.pt/mod/page/view.php?id=105915`

7 Aula 9: Construção de uma mini PKI

Nesta sessão iremos fazer uso do `openssl` para construir uma pequena PKI. Cada grupo irá ter uma pequena *autoridade de certificação*, que estará integrada na hierarquia da autoridade da UC de Aplicações Criptográficas, gerida pelo docente. O objectivo final é todos os alunos estarem dotados de certificados de chave pública que possam ser utilizados para comunicação autenticada e cifrada através de clientes de e-mail *standard* e, posteriormente, poderem gerar certificados para as aplicações a desenvolver nestes guiões.

O processo de emissão de qualquer certificados X509 passa pelos seguintes passos:

1. O (futuro) titular gera um par de chaves;
2. O (futuro) titular produz um "pedido de certificado"que contém a sua identificação e a sua chave pública, que é assinado com a sua chave privada (o que certifica que quem solicita o certificado está na posse da respectiva chave privada);
3. A autoridade de certificação valida os dados contidos no certificado e, no caso de certificar positivamente esses dados, emite o certificado de chave pública correspondente.
4. O certificado é enviado ao titular e, eventualmente, publicado por outros meios (e.g. serviço de directoria).

Note-se que o "titular"pode ser um cidadão, uma aplicação cliente ou servidor, ou mesmo uma outra autoridade de certificação.

Os próximos passos estão apresentados para o caso particular de cada grupo poder criar uma autoridade de certificação de segundo nível, com um certificado emitido pela AC gerida pelo docente.

Para cada um destes passos iremos fazer uso dos comandos respectivos do `openssl`. "XPTO" refere uma string de identificação do grupo.

Geração de chaves:

```
openssl genrsa -out grupoXPTO.key
```

Geração do pedido de certificado:

Cada grupo deverá pedir um certificado para a sua autoridade de certificação ao docente da disciplina, utilizando o seguinte comando:

```
openssl req -new -key grupoXPTO.key -out newreq.pem
```

É obrigatório utilizar estes parâmetros no nome da CA:

```
countryName               = PT
organizationName          = Universidade do Porto
organizationalUnitName    = MSI
```

O `commonName` deve ser começar por `SubCA` seguido de um identificador do grupo. O ficheiro `newreq.pem` resultante deve ser enviado ao docente, que devolverá o certificado `newcert.pem`.

Criação da SubCA

A emissão de certificados é normalmente realizada com o auxílio de *scripts* que invocam o comando *openssl* com os argumentos apropriados. Uma *script* normalmente utilizada para este efeito chama-se `CA.pl` e é distribuída com o `openssl`.

Para criar uma nova CA basta executar `./CA.pl -newca`. Neste caso não queremos que o *openssl* que gere um novo par de chaves, e devemos fornecer o nome do ficheiro `newcert.pem`.

Será criada uma pasta `demoCA`, e nessa pasta deverá ser colocada a chave privada préviamente gerada para a CA (a chave gerada no primeiro passo substitui ficheiro `cakey.pem` na subpasta *private*) e ser criado um ficheiro de texto com nome `serial` com o conteúdo "00" (ou qualquer hexadecimal com o primeiro número de série a utilizar).

Emissão de certificados

A partir deste ponto é possível assinar pedidos de certificado criados por qualquer pessoa colocando um ficheiro `newreq.pem` na pasta onde está o script `CA.pl` e executando `./CA.pl -sign`.

Verificação dos certificados:

Para validar um certificado, pode usar-se o seguinte comando

```
openssl verify -CAfile cacert.pem cert1.crt cert2.crt ...
```

Validação

Para comprovar que os conceitos foram adquiridos e a CA de cada grupo foi corretamente gerada, os grupos devem agora usar a sua CA para criar um par de chaves e um certicado para cada um dos alunos que o constituem, que depois deverão enviar ao docente um email assinado digitalmente.

Para instalar a chave privada e o certificado num cliente de email (webmail não pode ser utilizado), é necessário empacotar essas chaves num ficheiro do tipo P12.

De facto, para certas utilizações (e.g. *browsers*, leitores de *email*, etc.) é conveniente encapsularmos o certificado e respectiva chave privada num ficheiro `PKCS12`.

```
openssl pkcs12 -export -chain -CAfile chain.pem \
     -name nomechave -aes128 -inkey chave.key -in newcert.p12 -out pacote.p12
```

Para as aplicações de email aceitarem o certificado, é necessário incluir toda a cadeia de certificação no ficheiro `PKCS12`, e o certificado pessoal deve incluir no campo email o endereço que corresponde à conta a utilizar.

A cadeia de certificados inclui o certificado de raiz (publicado no Moodle), o certificado da SubCA que emitiu o certificado do utilizador, e o certificado do utilizador.

Pista: alguns clientes de email, nomeadamente o OSX mail exigem um conjunto de extensões muito específico para aplicações email no certificado. É necessário configurar o openssl para incluir essas extensões.

Integração Aplicacional

Para utilizar a chave privada no `Java` ou em qualquer outra linguagem é conveniente converter o seu formato para `PKCS8`

```
openssl pkcs8 -topk8 -nocrypt -in chave.key \
     -outform der -out chave.der
```

8 Aula 10: Integração da PKI

Pretende-se certificar as chaves públicas utilizadas no protocolo Station-to-Station com base em certificados X509 emitidos pela sub-CA criada por cada grupo. Para tal, cada grupo deve utilizar a sua sub-CA para gerar:

- Certificado de chave pública do Servidor: server.cer
- Certificado de chave pública do Cliente: client.cer

Que substituam aqueles fornecidos na Aula 8, seguindo os processos de criação de certificados da Aula 9.
A utilização de certificados pressupõe a sua validação: neste caso Cliente e Servidor apenas deverão conhecer de antemão o certificado de raiz da CA gerida pelo docente, devendo as duas cadeias de certificação ser trocadas e validadas no início da sessão.
O Java disponibiliza uma API específica que deverá utilizar para o efeito (documentação abaixo). Para facilitar esse estudo recomenda-se o estudo/adaptação de um programa de exemplo (ver Moodle) que verifica a validade de uma cadeia de certificação. Utilizando esse programa, podemos verificar a validade do certificado do servidor através da linha de comando:

```
java ValidateCertPath ca.cer servidor.cer
```

Uma segunda questão que surgirá neste trabalho é a manipulação dos formatos das chaves: as chaves privadas correspondentes aos certificados geralmente necessitam de estar codificadas no formato standard PKCS8. Dependendo da linguagem, a importação dessas chaves e a sua manipulação exigirá processos específicos.