Laboratórios de Informática 2016–2017
DETI - Universidade de Aveiro

Trabalho de aprofundamento 2

Objetivos:

- Planeamento e preparação do trabalho de aprofundamento 2.

Este guião apresenta as regras o segundo trabalho de aprofundamento. Os exercícios sugeridos dão os primeiros passos para a concretização do trabalho recorrendo às ferramentas estudadas.

1.1 Regras

O trabalho deve ser realizado por um grupo de 2 e entregue, via a plataforma `http://elearning.ua.pt`, dentro do prazo lá indicado. A entrega deverá ser feita por **apenas um dos membros** do grupo e deve consistir de **um único arquivo** `.zip`, `.tgz` (TAR comprimido por `gzip`) ou `.tbz` (TAR comprimido por `bzip2`). O arquivo deve conter o código desenvolvido assim como o ficheiro PDF final do relatório, todos ficheiros com código fonte (`.tex`, `.bib`, etc.) e todas as imagens ou outros recursos necessários à compilação do código ou documento. Excluem-se aqui eventuais imagens de máquinas virtuais utilizadas.

Na elaboração do relatório recomenda-se a adopção do estilo e estrutura de relatório descrito nas aulas teórico-práticas e a utilização de recursos de escrita como: referências a fontes externas, referências a figuras e tabelas, tabela de conteúdo, resumo, conclusões, etc. O objectivo do relatório é descrever a implementação, apresentar testes que comprovem o seu funcionamento correto e analisar os resultados obtidos.

É obrigatório incluir uma secção "Contribuições dos autores" onde se descrevem resumidamente as contribuições de cada elemento do grupo e se avalia a percentagem de trabalho de cada um. Esta auto-avaliação poderá afetar a ponderação da nota a atribuir a cada elemento

1.2 Avaliação

A avaliação irá incidir sobre:

1. **cumprimento dos objetivos através das funções suportadas**,
2. **qualidade do código produzido**,
3. **testes realizados**,
4. **estrutura e conteúdo do relatório**,
5. **utilização das funcionalidades de tarefas do code.ua e git**.
6. **o suporte de segurança adicionado**

A soma das componentes 1 a 5 é superior a 10 valores.

Relatórios meramente descritivos sem qualquer descrição da aplicação, apresentação dos resultados obtidos, testes efetuados, ou discussão serão fracamente avaliados.

Só serão avaliados trabalhos enviados via a plataforma `http://elearning.ua.pt`. Ficheiros corrompidos ou inválidos não serão avaliados à posteriori e não será permitido o reenvio.

Deve ser utilizado um projeto na plataforma Code.UA segundo o formato `labi2016-pbox-gX`. Substitua o caractere `X` pelo número 1. Se não for possível criar este projeto, incremente o número até que o resultado seja positivo.

1.3 Tema Proposto

O objetivo do trabalho é o de criar um cliente que permita realizar operações operações com um servidor. Este servidor implementa um modelo de caixas seguras onde é possível deixar pequenas notas para consulta do seu dono. Qualquer pessoa pode criar uma caixa com um nome específico, podendo também enviar uma chave pública para o servidor. Se for fornecida uma chave pública apenas o dono pode obter objetos da caixa segura mas qualquer pessoa pode depositar objetos. Caso não seja fornecida uma chave, a caixa é pública. Em qualquer um dos casos as caixas seguras são apagadas 24 horas após a sua criação.

A comunicação com o servidor faz-se através de um `socket` TCP, trocando objectos `JSON` terminados pelos caracteres `\r\n`. Assume-se que se podem implementar ferramentas distintas para cada operação (i.e. uma para listar, uma para criar, etc...), aceitando argumentos na linha de comandos, não sendo necessário que exista um cliente único. As mensagens possuem uma dimensão máxima de 65536 octetos.

Todas as comunicações do servidor são efetuadas sobre a forma de uma mensagem `RESULT` com um conteúdo que irá depender do pedido.

Assume-se que os computadores onde se executa o cliente possuem uma data correta. O servidor irá rejeitar pedidos com mais ou menos 15 segundos da hora atual do servidor) .

Para testar a interação com o servidor, tal como descrito nas aulas teórico-práticas, é possível utilizar os comandos `telnet` ou `nc`.

É importante que se dividam as operações em pequenos blocos de código (funções) de forma a que seja possível a realização de testes unitários. Na realização destes testes, deve-se validar a invocação das funções com valores dentro e fora do domínio de operação.

As operações suportadas são as descritas de seguida:

LISTAR caixas. Consiste no envio de uma mensagem LIST, ao que o servidor deverá responder com todas as caixas seguras existentes e as chaves públicas de cada caixa (caso existam). A mensagem `LIST` possui a seguinte estrutura:

```
{
    "type": "LIST"
}
```

Sendo que um resultado positivo terá o seguinte formato:

```
{
    "type": "RESULT",
    "timestamp": tempo do servidor em segundos,
    "code": "OK",
    "payload": [
        {
            "name": "nome da caixa",
            "size": número de items na caixa,
            "pubk": "chave pública da caixa"
```

```
        },
        ...
    ]
}
```

De notar que o campo `pubk` é opcional. Se forem implementadas várias ferramentas de linha de comandos pode ser adequado guardar o resultado desta mensagem num ficheiro. Mais tarde, isto pode auxiliar a obtenção das chaves públicas de cada caixa.

CRIAR criar uma caixa: consistindo no envio de uma mensagem CREATE para o servidor. A mensagem deve conter o nome da caixa e o timestamp atual. Opcionalmente pode contar uma chave pública e uma assinatura da mensagem, criada com a respetiva chave privada. Se for fornecida uma chave pública esta caixa só pode ser manipulada com mensagens seguras.

```
{
    "type": "CREATE",
    "name": "nome da caixa",
    "timestamp": tempo em segundos,
    "pubk": "chave pública",
    "sig": "assinatura da pubk, timestamp e name"
}
```

Os campos `pubk` e `sig` apenas são necessários se a caixa for segura.

Uma criação com sucesso irá resultar na seguinte mensagem:

```
{
    "type": "RESULT",
    "timestamp": tempo do servidor em segundos,
    "code": "OK"
}
```

ENVIAR um documento para uma caixa. Devendo ser enviada uma mensagem PUT, contendo um texto:

```
{
    "type": "PUT",
    "name": "nome da caixa",
    "timestamp": tempo atual em segundos,
```

```
    "content": "conteudo da mensagem",
}
```

Um envio com sucesso irá resultar na seguinte mensagem:

```
{
    "type": "RESULT",
    "timestamp": tempo do servidor em segundos,
    "code": "OK"
}
```

RECEBER um documento de uma caixa. Devendo ser enviada uma mensagem GET, contendo o timestamp atual. Opcionalmente pode conter uma assinatura calculada a restante mensagem:

```
{
    "type": "GET",
    "name": "nome da caixa",
    "timestamp": tempo atual em segundos,
    "sig": "assinatura do timestamp e name"
}
```

O campo `sig` só é necessário se a caixa for segura.

Um pedido com sucesso irá resultar na seguinte mensagem:

```
{
    "type": "RESULT",
    "timestamp": tempo do servidor em segundos,
    "code": "OK"
    "content" "corpo da primeira mensagem"
}
```

Após a obtenção de uma mensagem, esta é apagada da caixa no servidor.

1.4 Detalhes do suporte criptográfico

De forma a uniformizar os processos de criptográficos, devem ser considerados os pontos seguintes.

- Cifras Assimétricas: RSA com chaves de 2048 bits, enviadas em formato `PEM` (ver o método `exportKey('PEM')`). Um resultado possível seria o seguinte:

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvTm8pAOMzCIC1CtaZObo
tAh3ERqF+4qkZZkXRkLPMxRweNdBw4uoleKfJnC+OorXGTEB4NvEfThbdkhRLuuo
B6A5Gpi39PzGc4YBGHih2r3Ve3PzKkZhzuiUxg70nkB+dG+rcWlKOZUBNeqO8CJ+
1JLd7bj+5eM+A/CzX424lXXYP5WsxaN6PIOZjMH3mXVneqS7HcaDtus1XIKivebA
bip1urlGpncaZ/Tk0l98gMJfu8F+uq/9D28GQ/AIQ6LgE3wp0pxdCUbvwA9lmgdf
Z9dwYNq6DBoBj+PJ2m5m49WfifUJoqQYLuZ3LqRyDQ/EyabGNUKXiqz6wxqTkq3w
PwIDAQAB
-----END PUBLIC KEY-----
```

- Assinaturas: Método PKCS1_PSS com chaves RSA e sínteses SHA1 sobre a concatenação dos campos relevantes[1].

  No caso das mensages `GET` e verificação da mensagem `request` no servidor é feita da seguinte forma:

```
text = str(request['timestamp'])+request['name']
signer = PKCS1_PSS.new(key)
digest = SHA.new(text)
sig = base64.b64decode(req['sig'])
result = signer.verify(digest, sig)
```

  No caso das mensagens `CREATE` o processo é igual, excepto que utiliza-se igualmente o campo `pubk`:

```
text = request['pubk']+str(request['timestamp'])+request['name']
signer = PKCS1_PSS.new(key)
digest = SHA.new(text)
sig = base64.b64decode(req['sig'])
result = signer.verify(digest, sig)
```

Assume-se ainda que todos os dados transmitidos são compostos por carateres textuais. Conteúdos binários como as chaves, devem primeiro ser convertidos para texto através da utilização do módulo `base64`. O formato `PEM` já é em sí uma conversão para `base64`, com a adição de alguns delimitadores.

[1] ver PyCrypto PKCS1_PSS: https://www.dlitz.net/software/pycrypto/api/2.6/Crypto.Signature.PKCS1_PSS-module.html. **De notar que o argumento do método `sign` deve ser a síntese e não a chave**