

OFFACADEMY



FORMAÇÃO AVANÇADA
EM CIBERSEGURANÇA



NÍVEL A

Princípios Técnicos em Cibersegurança

35 horas / novembro 2025



Laboratório 2 – Sockets, GitHub e Introdução à Criptografia



Laboratório 2

Introdução

Este segundo laboratório tem os seguintes objetivos:

- Analisar programas que usam sockets para comunicação cliente/servidor;
- Realizar operações básicas na plataforma GitHub para partilha de código;
- Aplicar proteções criptográficas para proteger e desproteger ficheiros, via ferramenta OpenSSL e,
- opcionalmente, aplicar proteções criptográficas para proteger e desproteger ficheiros, através de biblioteca criptográfica em Python.

Em anexo ao laboratório existem ficheiros Python com código base para as questões 1 e 3.

Entrega

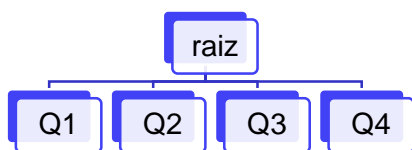
- (ZIP) Código fonte
- (PDF) Documento síntese do laboratório com o endereço do repositório do grupo, ecrãs e respostas solicitadas

1. Aplicação para lista de tarefas com arquitetura cliente/servidor

- Considere os ficheiros `client_todo.py` e `server_todo.py` em anexo.
- Abra duas linhas de comando do sistema (através do Visual Code usando a aplicação nativa do sistema operativo, `cmd.exe` no caso do Windows, `terminal` no caso macOS), ambas na diretoria onde tem o código fonte.
- Indique no código do cliente e do servidor a porta que pretende usar. Execute o cliente e o servidor. Registe no documento de síntese uma captura de ecrã onde seja possível ver os dois programas a executar.
- (extra) Acrescente ao sistema um comando para o cliente saber o número de itens totais da lista que ainda não estão completos, sem o cliente necessitar de obter a listagem individual de cada um dos itens.

2. GitHub

Nsta alínea e nas próximas use uma estrutura de directorias com o nome da alínea e subalíneas, como sugerido na figura:



- Cada elemento do grupo cria uma conta na plataforma GitHub (ou garante o acesso através de uma conta que já tenha criado anteriormente);
- Um dos elementos do grupo cria um repositório GitHub e dá acesso a esse repositório a todos os elementos do grupo e ao formador (utilizador: [Daniel-Alexandre-UTAD](#));
- Opcional: os elementos instalam a aplicação GitHub Desktop (<https://desktop.github.com>);
- Um dos elementos do grupo adiciona o código fonte da alínea 1 no repositório criado;
- Os restantes elementos fazem *pull* do repositório;
- Todos os elementos acrescentam à raiz dos seus repositórios locais um ficheiro de texto cujo conteúdo é o nome do formador;
- Todos os elementos do grupo fazem push para o repositório e verificam a correta sincronização do estado global do repositório.

3. Criptografia com OpenSSL

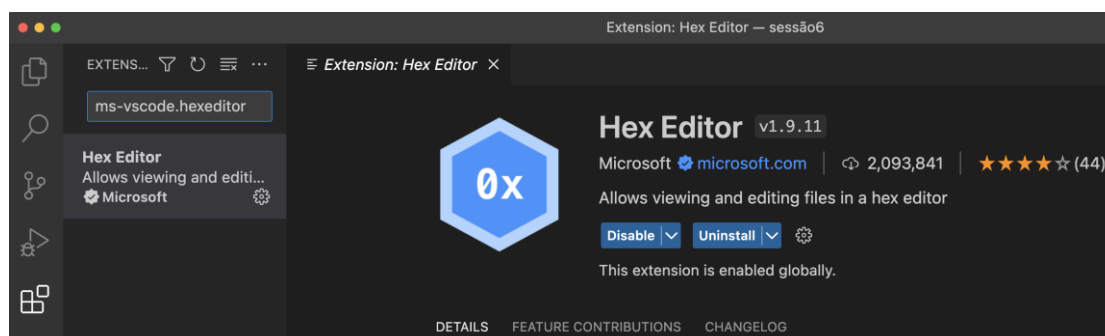
A biblioteca OpenSSL é uma biblioteca com código fonte aberto, escrita maioritariamente na linguagem C, e usada em muitas aplicações comerciais para realizar diferentes operações criptográficas (<https://www.openssl.org>). Existem duas formas de usar esta biblioteca: através do código fonte ou através de ferramenta de linha de comandos. Neste laboratório iremos usar o segundo caso.

Na generalidade dos sistemas operativos Linux e macOS a ferramenta OpenSSL já está instalada e disponível na linha de comandos:

```
$ openssl version.
LibreSSL 3.3.6
```

Nos sistemas Windows, há duas hipóteses: ou se compila o código C, ou se confia em algum binário que já tenha sido compilado por alguém. Vamos usar a segunda hipótese. Faça *download* do executável mais recente através deste link <https://indy.fulgan.com/SSL/>, cuja referência está presente nas páginas de documentação do OpenSSL (<https://wiki.openssl.org/index.php/Binaries>). Coloque o executável alcançável na *path* do sistema operativo (se necessário consulte as instruções para o efeito no laboratório 1).

Recomenda-se também a instalação da extensão “ms-vscode.hexeditor” no editor *visual code* para facilitar a visualização e modificação de ficheiros em modo binário.



a) Cálculo de *hash*

A ferramenta OpensSSL calcula valores de hash com base no conteúdo de um ficheiro usando o comando `dgst`:

```
$ openssl dgst -sha512 myfile
SHA512(myfile)=...abd3244f....
```

Documentação sobre o comando `dgst`: <https://www.openssl.org/docs/man1.1.1/man1/openssl-dgst.html>

- Calcule o *hash* SHA512 de um ficheiro de texto criado por si.
- Crie um cópia do ficheiro e confirme que o hash não é alterado quando se altera o nome do ficheiro;
- Modifique apenas 1 byte no ficheiro e volte a gerar o hash, comparando com o anterior;
- No relatório síntese coloque os ecrãs da geração do hash, da modificação do ficheiro e da segunda geração do hash.

b) Cálculo de *Message Authentication Code* (MAC)

A ferramenta OpenSSL calcula o MAC do conteúdo de um ficheiro e tem como parâmetro uma chave (neste caso uma palavra-passe):

```
$ openssl dgst -sha512 -hmac "cnscs!!2025" myfile
HMAC-SHA512(myfile)=...abc524a...
```

- Calcule o MAC do PDF deste enunciado usando uma chave escolhida pelo grupo;
- No relatório síntese coloque os ecrãs da geração do MAC.

c) Cifra e decifra simétrica

A ferramenta OpenSSL realiza a cifra e decifra simétrica de ficheiros, podendo usar diferentes algoritmos e modos de operação. O comando é sempre o `enc` mas a opção `-d` realiza a decifra. Por exemplo, os seguintes comandos cifram e decifram o ficheiro `x.pdf` usando a primitiva AES e o modo de operação CBC:

Cifra

```
$ openssl enc -aes-256-cbc -in x.pdf -out x.enc
enter aes-256-cbc encryption password:...
```

Decifra

```
$ openssl enc -aes-256-cbc -d -in x.enc -out x_decrypted.pdf
enter aes-256-cbc decryption password:...
```

Documentação sobre o documento enc: <https://www.openssl.org/docs/man1.1.1/man1/enc.html>

- i. Um elemento do grupo cifra o ficheiro PDF do enunciado usando uma chave combinada com todos os elementos;
- ii. O elemento que cifrou o ficheiro adiciona o ficheiro cifrado ao repositório (Q3);
- iii. Os restantes elementos obtêm o ficheiro e confirmam que conseguem realizar a decifra abrindo corretamente o PDF nos seus sistemas.
- iv. No documento de síntese escreva a chave usada para cifrar e decifrar o ficheiro.

d) Modos de operação

Pretende-se realizar a experiência abordada na sessão teórica, onde o mesmo ficheiro com uma imagem é cifrado duas vezes com o mesmo algoritmo, mas com dois modos de operação diferentes.

Nesta experiência é preciso copiar bytes de ficheiros em posições específicas. Nos sistemas Linux e macOS, os comandos `head`, `tail` e `cat` resolvem essa questão. Consideremos o exemplo seguinte:

```
$ head -c 54 p1.bmp > header
$ tail -c +55 p2.bmp > body
$ cat header body > new.bmp
```

Esta sequência de comandos começa por copiar do ficheiro `p1.bmp` os primeiros 54 bytes (dimensão do cabeçalho de imagens no formato BMP) para o ficheiro `header`. Depois, copia para o ficheiro `body` todos os bytes a partir da posição 55 do ficheiro `p2.bmp`, até ao fim do ficheiro. Finalmente, junta no ficheiro `new.bmp` os bytes de ambos os ficheiros `header` e `body`.

O ficheiro Python `copy-file-bytes.py` em anexo tem duas funções que fazem o equivalente a estes comandos. A função `copy_binary_file` copia um intervalo de bytes do ficheiro origem para o ficheiro destino, e a função `join_binary_file` junta o conteúdo de dois ficheiros num só.

- i. Considere a imagem em anexo `c-academy.bmp` e realize a cifra com AES e modo de operação ECB, de maneira a cifrar apenas o corpo da mensagem, mantendo a informação original do cabeçalho no ficheiro produzido;
- ii. Repita o processo mas agora com o modo de operação CBC;
- iii. Adicione os dois ficheiros resultantes da experiência no repositório GitHub;
- iv. Explique no documento síntese a diferença de resultados entre os dois ficheiros cifrados.

e) Criptografia assimétrica

Pretende-se nesta alínea realizar uma experiência com criptografia assimétrica, nomeadamente, gerar um par de chaves assimétricas, obter a assinatura de um ficheiro e verificar a assinatura

Para gerar um par de chaves RSA com dimensão 2048 bits armazenando o material criptográfico no ficheiro `private-key.pem`, use o comando:

```
$ openssl genrsa -out private-key.pem 2048
```

O comando seguinte gera uma assinatura para o ficheiro `x.pdf`. Note que o comando não incorpora os *bytes* da assinatura no PDF, apenas a escreve no ficheiro `signature.bin`:

```
$ openssl dgst -sha256 -sign private-key.pem -out signature.bin x.pdf
```

Os comandos seguintes realizam as seguintes operações:

- extrair a chave pública para o ficheiro `public-key.pem`
- verificar a assinatura gerada anteriormente usando a chave pública

```
$ openssl rsa -in private-key.pem -pubout -out public-key.pem  
$ openssl dgst -sha256 -verify public-key.pem -signature signature.bin x.pdf
```

- Adapte os comandos anteriores para assinar e verificar o PDF do enunciado;
- Modifique 1 byte do ficheiro PDF original usando o editor visual code. Adicione ao relatório de síntese o ecrã capturado com o processo de edição;
- Repita o processo de verificação e verifique que existe falha. Inclua no documento síntese a captura de ecrã da verificação com sucesso e com falha.

4. (extra) Criptografia em Python

Considere o ficheiro `fernet-sym-cypher.py` em anexo, cujo programa gera uma chave simétrica, realizando depois a cifra e a decifra uma string. O programa usa a biblioteca `cryptography`, em particular o módulo simplificado para cifra simétrica autenticada (Fernet). A biblioteca é composta por este módulo e por um conjunto de serviços de criptografia onde é necessário fazer mais algumas parametrizações para um correto uso da biblioteca.

- i. Instale a biblioteca `cryptography` usando a aplicação `pip`: <https://cryptography.io/en/latest/>;
- ii. Usando o módulo `Fernet`, realize um programa que gera e grava em ficheiro uma nova chave para usar neste módulo;
- iii. Considere o ficheiro `server_todo_fernet.py` fornecido em anexo. O programa corresponde a uma versão modificada do servidor de tarefas, onde cada tarefa é guardada cifrada no ficheiro `todo_list.fernet`, cada vez que há uma alteração à lista. O programa tem um erro – quando o ficheiro está a ser carregado para memória na fase inicial da aplicação (caso o ficheiro exista), a informação sobre a tarefa estar completa não está a ser considerada. Corrija o erro;
- iv. Teste o cliente anterior e o novo servidor, observando que o ficheiro é sempre gravado com os itens cifrados. Coloque ecrãs no documento síntese que demonstrem os testes;
- v. Publique o código no repositório.