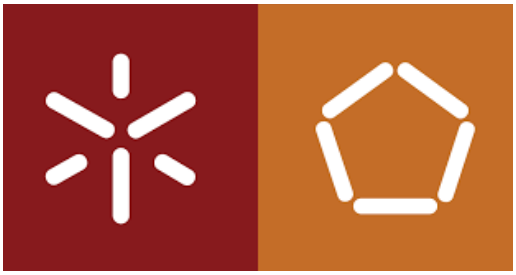


Relatório Criptografia e Segurança em Redes

2023-2024

UNIVERSIDADE DO MINHO

JORGE SOUSA A94153

LUÍS PINHEIRO A58381

Índice

Aplicação de Chat

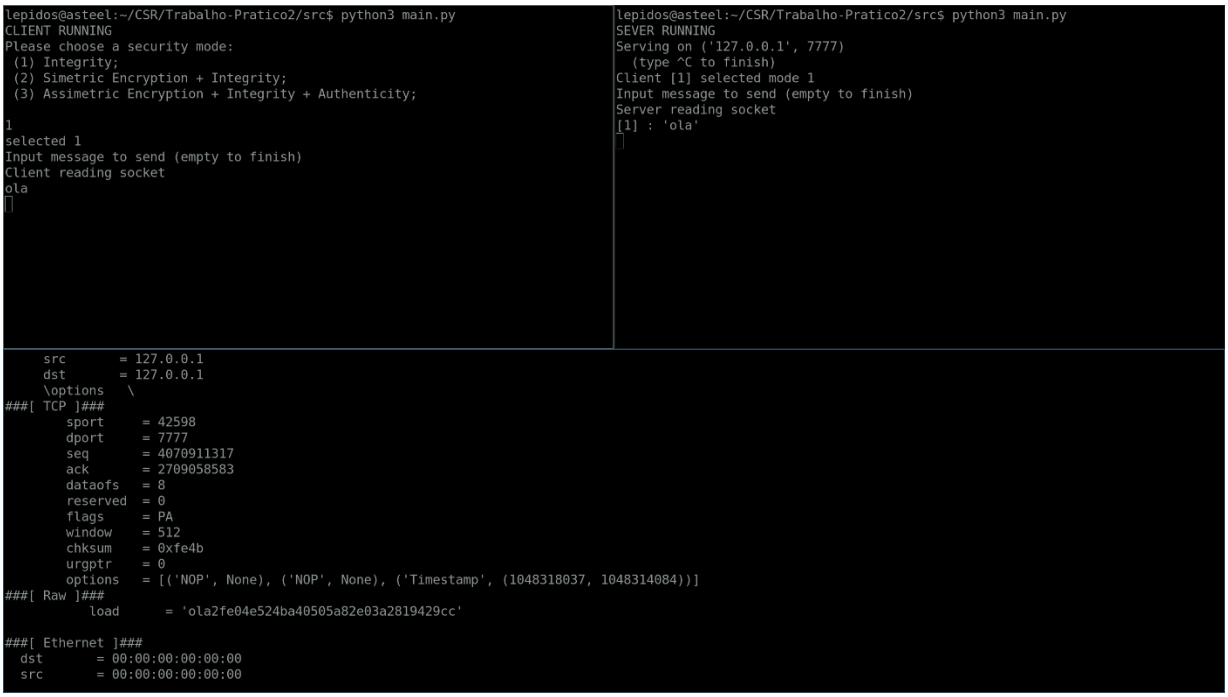
Mais uma vez recorreu-se ao software **Python 3** para desenvolver o trabalho prático.

O código da aplicação de chat do servidor e do cliente estão no mesmo ficheiro `main.py` porque estes partilham quase a totalidade do código. A aplicação ao ser iniciada por uma ponta, tenta ser o cliente, ao falhar cai no papel do servidor. A próxima inicialização vai ser bem-sucedida ao tentar ser o cliente.

De forma a lidar com o problema de bloqueio de entrada-saída, as funções de escrita na socket, que leem do *STDIN,* são sempre executadas noutra thread. A thread termina depois de ser inserido no *STDIN,* só o fim-de-linha, e envia pelo socket o mesmo caratecter *n* para, da mesma forma, o loop-infinito de leitura do socket terminar.

Modo 1: Integridade

Deforma a verificar a integridade das mensagens é produzida e enviada pelo transmissor a **hash** da mensagem que foi digitada. Como o tamanho da hash usado é constante, 32-bit, suficiente para provar a integridade numa aplicação simples. O recetor trunca 32-bits da mensagem tenta replicar a mesma hash com a mensagem que recebeu, caso seja bem-sucedido é assumido que a mensagem não sofreu alterações.



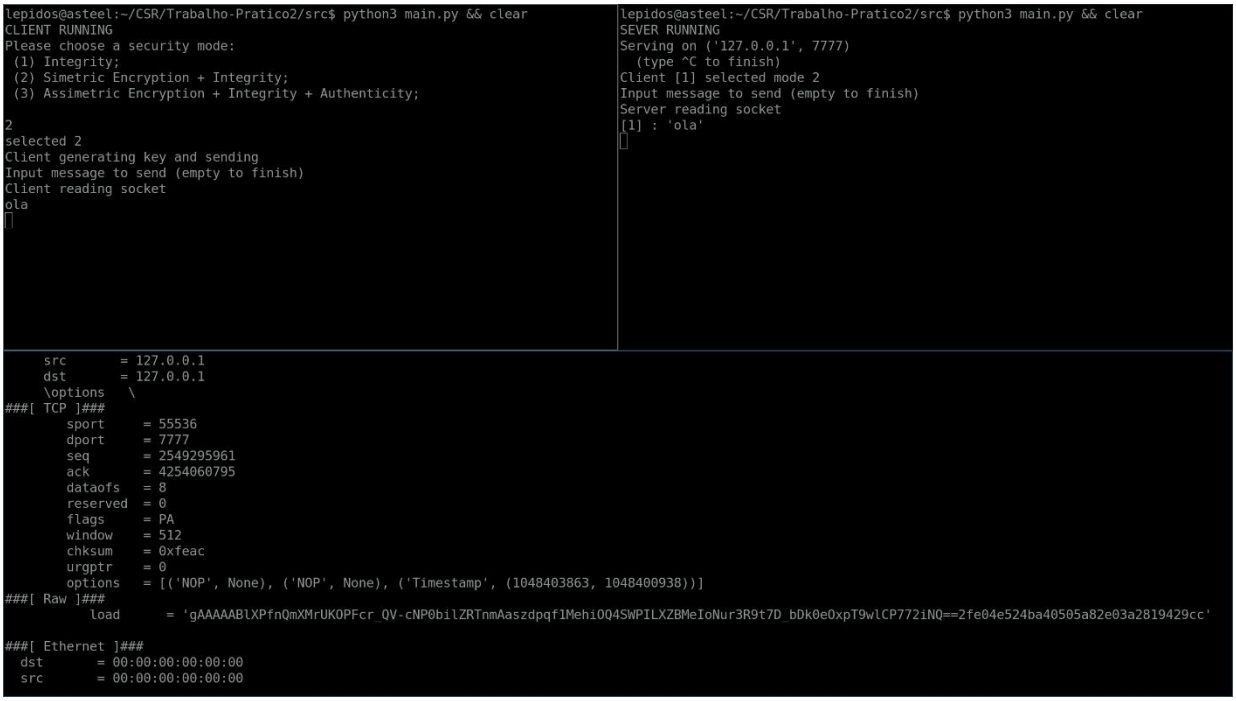
Modo 2: Confidencialidade e Integridade

Dando continuidade ao que processo descrito no modo 1. Em vez de enviar a mensagem em texto+hash, neste modo enviamos o conteúdo da mensagem encriptado usando uma cifra simétrica e a hash do texto, outra vez de 32-bit.

O algoritmo **Fernet** faz um conjunto de operações com uma chave à mensagem. A chave possibilita tanto cifrar como decifrar neste algoritmo. Dando assim garantia de confidencialidade, isto é só quem possui a chave consegue reverter a cifra e obter o texto da mensagem.

```
key = Fernet.generate_key()
```

Sempre que uma ligação começa o cliente gera uma chave e envia ao servidor. Neste momento ainda não é possível usar como suporte a cifras simétrica. Esta é uma vulnerabilidade grave à confidencialidade, a qual as cifras assimétricas estão endereçadas a resolver. Ou assumir que ambos já tinham acordado uma chave antes de iniciar a aplicação e optar por fixar a uma chave como variável constante no código. Caso fosse esta a implementação deveria-se usar nonce para evitar ataques replay.



Modo 3: Confidencialidade, Integridade e Autenticidade

Neste modo, criptografia assimétrica, o algoritmo escolhido foi RSA, a integridade está aninhada no método de que garante a autenticidade.

```
signature = private_key.sign(ciphertext,
            padding.PSS(mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH),hashes.SHA256())
```

A confidencialidade é garantida pelo mecanismo assimétrico, agora são geradas duas chaves por cada um, uma para encriptar e outra para desencriptar. As chaves de encriptar são enviadas no início da comunicação. Assim cada um encripta a mensagem usando a chave que recebeu. Garantindo que só quem tem a correspondente chave de desencriptar consegue reverter a transformação. Assim nem o próprio transmissor da mensagem consegue decifrar o que cifrou, pelo que usou a chave que recebeu do outro lado da comunicação.