

Projeto 2 - Comunicações Seguras

Trabalho Realizado por: Vasco Ramos - nº mec 88931
Diogo Silva - nº mec 89348

Data de Preparação: Aveiro, 18 de Novembro de 2019

Cadeira: Segurança Informática e nas Organizações

Corpo Docente: Professor João Paulo Barraca
Professor Vítor Cunha

Índice

Introdução	3
1. Planeamento	4
2. Desenho	5
3. Implementação	7
3.1 Negociação dos Algoritmos Utilizados	7
3.2 Partilha de Chave Utilizada	8
3.3 Confidencialidade	9
3.4 Integridade	10
3.5 Rotação de Chave Utilizada	11
3.6 Envio do Ficheiro	12
4. Execução	14
Exemplo de Execução - rick.txt	14
5. Conclusão	17
6. Bibliografia e Referências	18

Introdução

No seguimento do plano curricular da disciplina de Segurança Informática e nas Organizações, do curso de Engenharia Informática, da Universidade de Aveiro, este relatório é o resultado da execução do segundo trabalho prático e tem como principal objetivo explorar os conceitos relacionados com o estabelecimento de uma sessão segura entre dois interlocutores e os conceitos relacionados com a troca de chaves, cifras simétricas e controlo de integridade.

Ao longo deste documento vamos apresentar:

- O **planeamento** da solução
- O **desenho** do protocolo
- A **implementação** do sistema
- A **execução** do sistema.

1. Planeamento

Numa primeira fase, abordámos o problema de um ângulo mais teórico de forma a planearmos o melhor possível o que seria necessário fazer.

Assim, focámo-nos nos tópicos descritos no guião:

1. **Desenhar um protocolo** para o estabelecimento de uma **sessão segura** entre o cliente e o servidor, suportando:
 - a. Negociação dos algoritmos usados
 - b. Confidencialidade
 - c. Controlo de integridade
 - d. Rotação de chaves.
2. **Implementar a negociação de algoritmos** de cifra entre cliente e servidor.
3. **Implementar o suporte para confidencialidade**, resultando em mensagens cifradas.
4. **Implementar o suporte para integridade**, resultando na adição de códigos de integridade às mensagens.
5. **Implementar um mecanismo para rotação da chave** utilizada após um volume de dados ou tempo decorrido.
6. **Implementar funções genéricas** de cifra, decifra, cálculo de um MAC, verificação de um MAC de textos.
7. **Criar novos tipos de mensagens a enviar**, incluindo as mensagens já existentes dentro do conteúdo destas novas mensagens (num formato cifrado e com integridade).

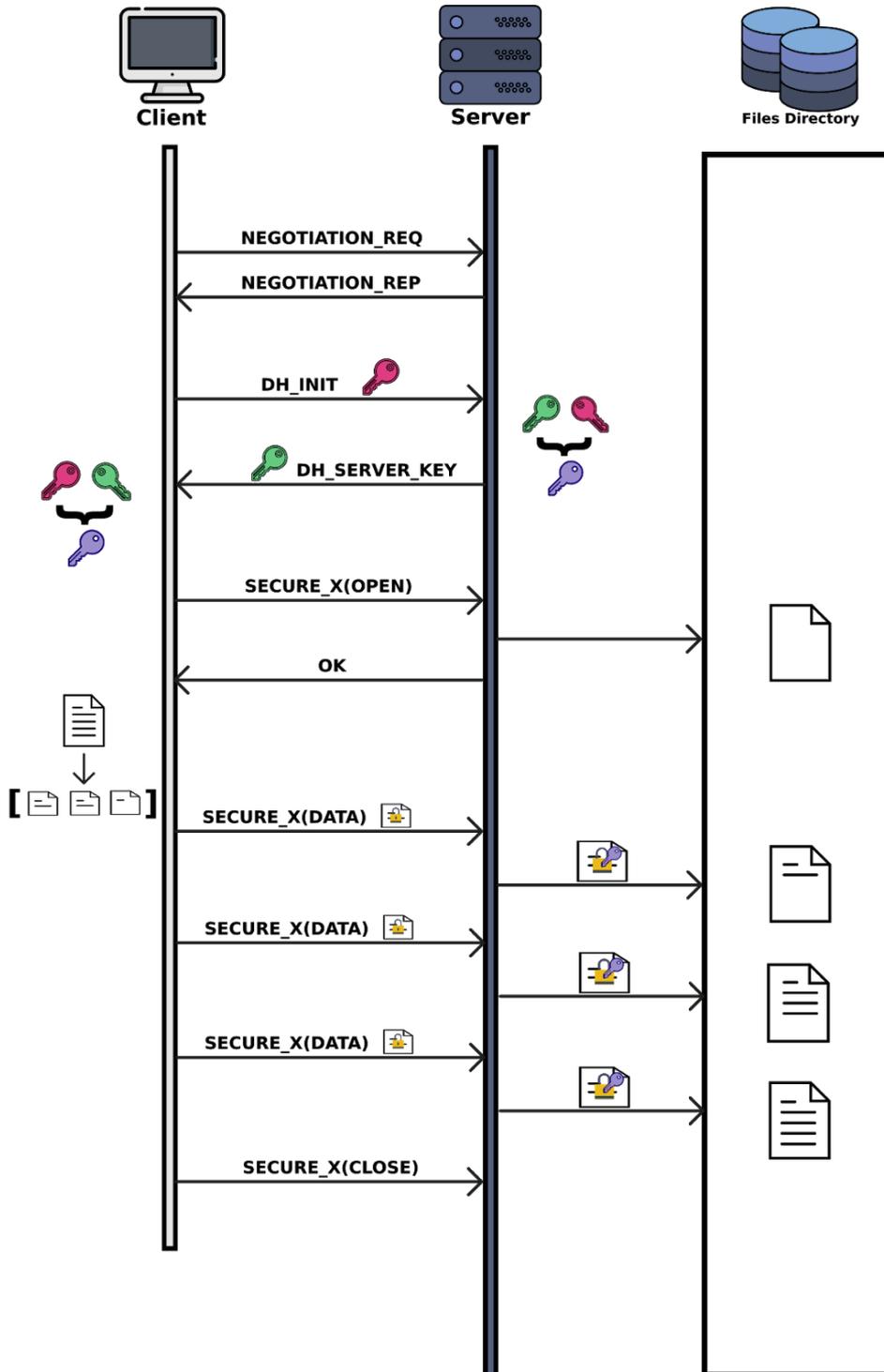
2. Desenho

O protocolo inicia-se com uma troca de mensagens para proceder à negociação dos algoritmos a utilizar no túnel seguro de comunicação. O cliente envia uma mensagem do tipo **NEGOTIATION_REQ** ao servidor, onde inclui os algoritmos que suporta. Ao receber esta mensagem, o servidor analisa os algoritmos suportados pelo cliente e, caso, haja um match, escolhe os algoritmos a utilizar (envia uma mensagem do tipo **NEGOTIATION_REP**, com a decisão); caso contrário, recusa-se a estabelecer o túnel e termina a comunicação (envia uma mensagem do tipo **ERROR**).

O próximo passo é estabelecer a chave para encriptar as mensagens. Para isto utilizámos o algoritmo de Diffie-Hellman. O cliente envia uma mensagem do tipo **DH_INIT** ao servidor, onde inclui os parâmetros partilhados e a sua chave pública. Ao receber esta mensagem, o servidor calcula a sua chave privada (com os parâmetros partilhados), a sua chave pública e a chave partilhada e envia uma mensagem do tipo **DH_SERVER_KEY**, onde inclui a sua chave pública, com a qual o cliente vai calcular a chave partilhada.

Agora que ambos têm a chave de encriptação a ser usada, o cliente manda uma mensagem do tipo **SECURE_X** com o payload de uma mensagem de **OPEN** com o nome do ficheiro à qual o servidor irá responder com um **OK**, iniciando-se assim o processo de envio do ficheiro através de sucessivas mensagens do tipo **SECURE_X** com o payload **DATA**. Quando o cliente termina o envio do ficheiro, envia uma mensagem do tipo **SECURE_X** com o payload de **CLOSE** e a comunicação termina.

O diagrama abaixo mostra um diagrama de comunicação representativo do processo. A renegociação da chave utilizada (e outras mensagens de erro) não estão incluídas no diagrama apenas para manter simplicidade, contudo, serão explicadas mais à frente.



3. Implementação

3.1 Negociação dos Algoritmos Utilizados

Como explicado anteriormente, a sessão entre o **cliente** e o **servidor** inicia-se com a negociação dos algoritmos a utilizar no túnel de comunicação segura:

- Algoritmo de cifra
- Modo de cifra
- Função de síntese.

Para tal, o cliente informa o servidor dos algoritmos que suporta através de uma mensagem do tipo **NEGOTIATION_REQ**:

```
message = {
    "type": "NEGOTIATION_REQ",
    "algorithms": {
        "symetric_ciphers": self.symetric_ciphers,
        "chiper_modes": self.chiper_modes,
        "digest_algorithms": self.digest_algorithms,
    },
}
self._send(message)
```

Ao receber esta mensagem, o servidor processa a informação e caso tenha um match para cada um dos algoritmos, escolhe os mesmos e informa o cliente da decisão. Caso não haja um match para algum dos algoritmos responde com um erro e a sessão é terminada.

```
message = {
    "type": "NEGOTIATION_REP",
    "algorithms": {
        "symetric_cipher": self.used_symetric_cipher,
        "cipher_mode": self.used_chiper_mode,
        "digest_algorithm": self.used_digest_algorithm,
    },
}
if (
    self.used_symetric_cipher is not None
    and self.used_chiper_mode is not None
    and self.used_digest_algorithm is not None
):
    self._send(message)
    return True
return False
```

Optámos por ser o servidor a escolher os algoritmos a utilizar e não o cliente, pois, achámos que seria mais indicado, tendo em conta, que, regra geral, o servidor tem mais conhecimento (tanto sobre a carga do sistema, como sobre a necessidade de segurança) do que o cliente.

3.2 Partilha de Chave Utilizada

O próximo passo é o processo de **troca de chave** a ser utilizada, através do algoritmo **Diffie Hellman**.

O cliente começa por gerar os parâmetros que serão partilhados com o servidor, bem como as suas componentes privada e pública.

```
parameters = dh.generate_parameters(  
    generator=2, key_size=512, backend=default_backend()  
)  
  
private_key = parameters.generate_private_key()  
public_key = private_key.public_key()  
  
p = parameters.parameter_numbers().p  
g = parameters.parameter_numbers().g  
  
public_key_pem = public_key.public_bytes(  
    serialization.Encoding.PEM, serialization.PublicFormat.SubjectPublicKeyInfo  
)
```

De seguida, o cliente envia os parâmetros (p e q) e também a sua componente pública (*public_key_pem*) para o servidor, através de uma mensagem do tipo **DH_INIT**.

Ao receber esta mensagem, o servidor vai reconstituir o objeto *DH_PARAMETERS* e criar as suas componentes pública e privada. Através da componente pública do cliente, o servidor cria a *shared_key* que irá ser derivada para criar a *derived_key*, a chave que efetivamente irá ser utilizada na comunicação entre o cliente e o servidor.

O servidor envia a sua componente pública para o cliente, através de uma mensagem do tipo **DH_SERVER_KEY**.

O cliente, ao receber esta mensagem, à semelhança do que o servidor faz, calcula a *shared_key*, derivando-a para a *derived_key*, concluindo assim o processo de **troca de chaves**.

3.3 Confidencialidade

Para garantir a confidencialidade criamos a mensagem do tipo **SECURE_X** para encapsular as nossas mensagens do tipo **OPEN**, **CLOSE** e **DATA**. Estas são encriptadas (utilizando a nossa função *symmetric_encrypt*) e tratadas (i.e codificadas em Base64) pela nossa função *create_secure_message* sendo posteriormente colocadas no campo *payload* da **SECURE_X**.

```
message = {
  "type": "SECURE_X",
  "payload": None,
  "mac": None,
  "iv": None,
  "nonce": None,
  "tag": None
}
(...)
message["payload"] = base64.b64encode(plaintext).decode()
```

Para além da mensagem em si, a **SECURE_X** possui também campos para guardar o iv, nonce e tag utilizados na encriptação, sendo estes posteriormente necessários para a realização da decriptação.

A *create_secure_message* vai então retornar a **SECURE_X** para que possa ser enviada para o servidor. É da responsabilidade deste realizar o decode dos campos codificados em Base64 da **SECURE_X** e proceder à decriptação do *payload* (utilizando a função *symmetric_key_decrypt*) utilizando a sua Chave Partilhada e, caso necessário, os campos extra (iv, tag e nonce) incluídos na mensagem. Após a decriptação o servidor vai então possuir a mensagem original - **OPEN**, **CLOSE** ou **DATA** - procedendo então ao processamento adequado.

3.4 Integridade

Aquando da criação da mensagem **SECURE_X** é também chamada a função `generate_mac` que vai, utilizando o algoritmo de síntese escolhido e a chave partilhada, criar o MAC do criptograma criado anteriormente.

```
if algorithm == "SHA256":
    hash_algorithm = hashes.SHA256()
elif algorithm == "SHA512":
    hash_algorithm = hashes.SHA512()
elif algorithm == "BLAKE2":
    hash_algorithm = hashes.BLAKE2b(64)
else:
    raise Exception("Hash Algorithm name not found")
mac = hmac.HMAC(key, hash_algorithm, backend=default_backend())
```

Este MAC é incluído no campo `mac` da mensagem **SECURE_X**.

Posteriormente, no lado do servidor, este realiza a verificação da integridade e autenticidade do criptograma recebido gerando ele próprio um MAC do criptograma recebido, utilizando a mesma função mencionada anteriormente, e comparando-o com o incluído na mensagem para verificar se são iguais. Caso não sejam o servidor apaga o ficheiro que estava a escrever mandando também uma mensagem de **ERROR** para o cliente.

```
digest = crypto_funcs.generate_mac(
    actual_message, self.shared_key, self.used_digest_algorithm
)
if mac != digest:
    if self.file_path != None: # If we created a file delete it!
        os.remove(self.file_path)
    logger.warning("The integrity of the message has been compromised")
    ret = False
```

É de notar que, como pode ser deduzido, foi escolhida uma abordagem **ENCRYPT-THEN-MAC** de forma a que a integridade pudesse ser verificada mesmo antes da descriptação da mensagem do lado do servidor.

3.5 Rotação de Chave Utilizada

Tendo em conta que o principal objetivo deste trabalho é enviar um ficheiro de uma forma segura, é necessário que exista um mecanismo de rotação de chaves, garantindo a alteração da chave após ter sido ultrapassado um certo threshold (seja este tempo, número de blocos enviados, etc).

Tal como será explicado no próximo ponto, o ficheiro é dividido em chunks com um tamanho de $60 * block_size$. Ao atingir um threshold de 1000 blocos, o cliente inicia novamente o processo de criação de uma chave partilhada através do algoritmo de Diffie Hellman. O processo da criação da chave é semelhante ao processo descrito anteriormente.

Para além disto guarda-se o número de blocos que já tinha sido enviado para, após a rotação de chaves ter sido concluída, retomar o envio do ficheiro do ponto onde este foi “interrompido”.

3.6 Envio do Ficheiro

No que toca ao envio do ficheiro, decidimos abordar este problema através da **leitura de uma só vez do ficheiro**, sendo este então dividido em vários **chunks** com o mesmo tamanho que serão guardados numa estrutura de dados - i.e um array. Este array é guardado como um parâmetro da nossa classe `ClientProtocol` para possibilitar que, em caso de interrupção do algoritmo de envio devido à necessidade de renegociação de chaves, possamos retomar o envio a partir do último chunk enviado.

```
text = None
with open(file_name, "rb") as reader:
    text = reader.read()
(...)
self.text_chunks = [ # Divide text into chunks
    text[i : i + block_size] for i in range(0, len(text), block_size)]

# Treat empty files:
if self.text_chunks == []:
    self.text_chunks = [str.encode(" ")]
```

O algoritmo de forma muito sucinta realiza, para cada Chunk incluído no array, a criação de uma mensagem do tipo DATA onde incluímos o Chunk codificado em Base64 e a utilização da nossa função `create_secure_message` para gerar uma mensagem **SECURE_X** onde é colocada a outra mensagem. Esta será então enviada ao servidor.

```
for i in range(sent_packages_counter, len(self.text_chunks)):
    chunk = self.text_chunks[i]
    data_message = {"type": "DATA", "data": None}
    data_message["data"] = base64.b64encode(chunk).decode()

    message = crypto_funcs.create_secure_message(
        data_message,
        self.shared_key,
        self.used_symetric_cipher,
        self.used_cipher_mode,
        self.used_digest_algorithm,
    )
```

Note-se também que é utilizada uma variável de contador para inicializar o algoritmo de renegociação de segredo compartilhado caso esta ultrapasse um dado threshold. Isto foi realizado para garantir que a Chave Partilhada é renovada não podendo ser utilizada demasiadas vezes.

```
if (
    sent_packages_counter % 1000 == 0
): # FIXME: to test video should be > 100 000
    (
        self.p,
        self.g,
        self.private_key,
        self.public_key_pem,
    ) = crypto_funcs.diffie_hellman_client()

    message = {
        "type": "DH_INIT",
        "parameters": {
            "p": self.p,
            "g": self.g,
            "public_key": str(self.public_key_pem, "ISO-8859-1"),
        },
    },
}
```

Por outro lado é também de mencionar que o tamanho de cada Chunk de texto enviado depende do algoritmo de cifragem utilizado.

```
if self.used_symetric_cipher == "AES":
    block_size = 16 * 60
elif self.used_symetric_cipher == "3DES":
    block_size = 8 * 60
elif self.used_symetric_cipher == "ChaCha20":
    block_size = 16 * 60
```

4. Execução

Exemplo de Execução - rick.txt

Neste exemplo vamos explicar de forma explícita todo o processo da transferência do ficheiro *rick.txt* do cliente para o servidor:

```
(venv) ds@pop-os:~/Desktop/SIO/Projetos/project-2$ python3 server.py
2019-11-17 17:48:11 pop-os root[14190] INFO Port: 5000 LogLevel: 20 Storage: /home/ds/Desktop/SIO/Projetos/project-2/files
[2019-11-17 17:48:11 +0000] [14190] [INFO] Single tcp server starting @0.0.0.0:5000, Ctrl+C to exit
2019-11-17 17:48:11 pop-os aio-tcpserver[14190] INFO Single tcp server starting @0.0.0.0:5000, Ctrl+C to exit
[2019-11-17 17:48:11 +0000] [14190] [INFO] Starting worker [14190]
2019-11-17 17:48:11 pop-os aio-tcpserver[14190] INFO Starting worker [14190]
```

Img 4.1 - Inicialização do Servidor

```
(venv) ds@pop-os:~/Desktop/SIO/Projetos/project-2$ python3 client.py rick.txt
2019-11-17 17:51:47 pop-os root[14390] INFO Sending file: /home/ds/Desktop/SIO/Projetos/project-2/rick.txt to 127.0.0.1:5000 LogLevel: 20
```

Img 4.2 - Inicialização do Cliente, especificando o ficheiro rick.txt

```
2019-11-17 17:51:47 pop-os root[14390] INFO Send: {'type': 'NEGOTIATION_REQ', 'algorithms': {'symmetric_ciphers': ['AES', 'ChaCha20', '3DES'], 'cipher_modes': ['GCM', 'None', 'ECB', 'CBC'], 'digest_algorithms': ['SHA256', 'SHA512', 'BLAKE2']}}
2019-11-17 17:51:47 pop-os root[14372] INFO Connection from ('127.0.0.1', 34270)
2019-11-17 17:51:47 pop-os root[14372] INFO Send: {'type': 'NEGOTIATION_REP', 'algorithms': {'symmetric_cipher': 'AES', 'cipher_mode': 'GCM', 'digest_algorithm': 'SHA256'}}
2019-11-17 17:51:47 pop-os root[14390] INFO Send: {'type': 'DH_INIT', 'parameters': {'p': 10815929782036807544484905835844725310109419197769841816030453134952545640419933192920894544637285746636300736595840665664905874508710239544976210372982483, 'g': 2, 'public_key': '-----BEGIN PUBLIC KEY-----\nmIGbMFMGCSqGSIb3DQEDATBGAKEAzoMp3AxsR3Qp9GaoPpYc9chC3oyctWGPnsga\nyoCuk157lety8bUC3Sj3Bvw58XDuNS5asY7Zq8nGwUSSWG320wIBAgNEAAJBAMZb\nFedx5HH/aXgtrpf6kpB8hbM0BSyFdRumngE7uOmp3nS/YLWVYT\nVnugzBZJaQSwCP\nesp05xP9/wFk7Q0IM3w\n\n-----END PUBLIC KEY-----\n'}}
2019-11-17 17:51:47 pop-os root[14372] INFO Send: {'type': 'DH_SERVER_KEY', 'key': '-----BEGIN PUBLIC KEY-----\nmIGbMFMGCSqGSIb3DQEDATBGAKEAzoMp3AxsR3Qp9GaoPpYc9chC3oyctWGPnsga\nyoCuk157lety8bUC3Sj3Bvw58XDuNS5asY7Zq8nGwUSSWG320wIBAgNEAAJBAMZb\nFedx5HH/aXgtrpf6kpB8hbM0BSyFdRumngE7uOmp3nS/YLWVYT\nVnugzBZJaQSwCP\nesp05xP9/wFk7Q0IM3w\n\n-----END PUBLIC KEY-----\n'}}
2019-11-17 17:51:47 pop-os root[14390] INFO My Shared Key: b'\x00\x17\x98[\xb7,\xaf\x1e\xc7\xca\xcai\xe7\xdc\xe0\xd46\xaaS\xa19\xd9G[\xc0\xa8\x10d\xa0\x10\x1b\xb9'
2019-11-17 17:51:47 pop-os root[14372] INFO My Shared Key: b'\x00\x17\x98[\xb7,\xaf\x1e\xc7\xca\xcai\xe7\xdc\xe0\xd46\xaaS\xa19\xd9G[\xc0\xa8\x10d\xa0\x10\x1b\xb9'
```

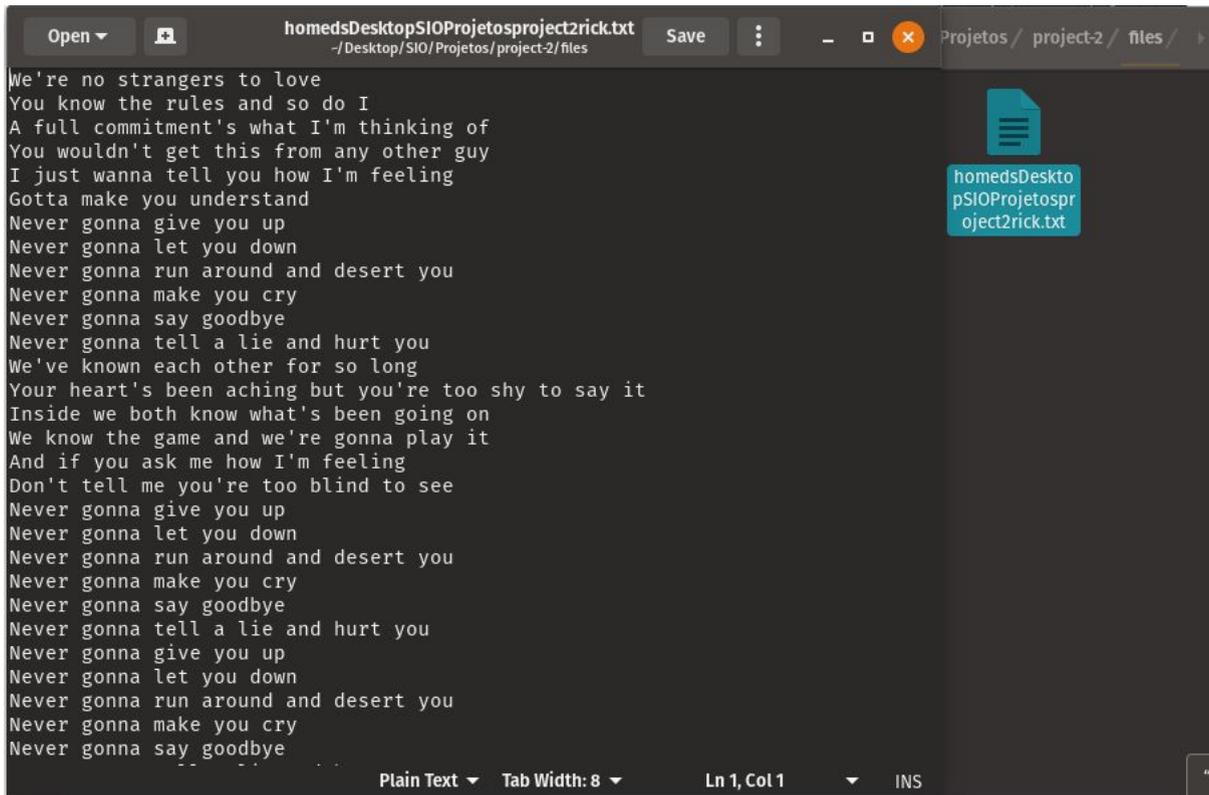
Img 4.3 - Envio de um **NEGOTIATION_REQ** do cliente (terminal da esquerda) para o servidor (terminal da direita) que responde com um **NEGOTIATION_REP**. Depois disto o cliente envia um **DH_INIT** dando início à negociação do segredo partilhado. O servidor responde com um **DH_SERVER_KEY** completando então o algoritmo de Diffie-Hellman. Depois disto, tanto o cliente como o servidor vão possuir uma mesma Chave Partilhada igual.

```
2019-11-17 17:51:47 pop-os root[14390] INFO Send: {'type': 'SECURE_X', 'payload': 'kbOTU1EluhPA0-SbdJ2xrhW3ymj21aR+5SQ/h9NMEipqCWI3ku/Mi-Crte8WFEMc5h4I2F5YM0DYT
: 'Fq+kupls6qH672Xc6ixaerOX6U4P1y8fA2LWcoqKBZfThdWkYA2ip+hfZp+j54kv10G/dp+f2eR9jqb/
FkVgX06ccj+gKsg=', 'mac': 'nj6GPH8BfN27k+cZ2n1g+L7pZV516NrdgXaZb+qRSg=', 'iv': '
h2i+I9rRXHj2LsLj', 'nonce': None, 'tag': 'GrOwEMp71KbjjH6HMYFqw='}
2019-11-17 17:51:47 pop-os root[14372] INFO File open
2019-11-17 17:51:47 pop-os root[14372] INFO Send: {'type': 'OK'}
2019-11-17 17:51:47 pop-os root[14390] INFO Channel open
```

Img 4.4 - Envio de uma mensagem de **OPEN**, encriptada e encapsulada numa mensagem **SECURE_X**, pelo cliente (terminal da esquerda). O servidor vai então responder com uma mensagem do tipo **OK**, efetivamente permitindo então o estabelecimento de um canal de comunicação entre estes dois. Note-se que é também neste passo que o servidor cria o ficheiro onde irão ser guardados os Chunks enviados.

```
2019-11-17 17:51:47 pop-os root[14390] INFO Transferring Chunk
2019-11-17 17:51:47 pop-os root[14390] INFO Send: {'type': 'SECURE_X', 'payload': 'CGdoNurNeMxyVDMKhlmdmWu6pexdRwKfpxlIwBgWklSeVzKjyM9o5pt0Zu0Ukp0x8AdU2EbSGzDd6tE/kTPDP8X52W/75408jwEq87+wf
Uts6oXIe8AMcYNT+fx2gw1/pfBvDx9T62CVgQh062oozpsYfWx5NEK1YBHQNAE7Imkwm3Cctt+qAM6NPEVQgz1no2EZx9LS6XH4ERA7yV5Fj49HATXQmj77j+0aa0a6g336AIfctM2BET04r8WkKx07Hk03K5Exp8gHHNgOQFwgjYwWkVICMvYgtT
Nlgw/7fJ8rwd/Reg7jw7pbu+wgx+mp2eukW0Fwchpy9Zb9600c2ne7Kfsv0Q45J4hu31aPjkt+9jWRmC3WUxq1Ibt2tXKRDXPY013e6Z78QVidpDcJR8smhD2S34bv2o2/VyfmhIhh+x+bg/s+kl6hhKHU882jxPL7kTukM2TGKRLSCLDdAno
zUUSwnqcG8dP60T/Zar1B87yQae6w6SddnVK41jxvH5qhlE5dX8Yw8q3R8YGHuLmRQfy09Y6Y5UWDCVf/b8fYhfwyG/fbUjap1XqeQrF5PjtgINTJU52bfrLZAS10H41xVI+WcQJDI48AJkunv/9Uvd5R8GdXgrAovteuSmByeSm6lV6l+vw6rX7Qq
5nTvw4xeN7OHY2WnjBp/Nrcus80uk6vRwgdC9p0pVpXTeFa0Ytd+LXu6SngGH/mst+mb8jcsEqPtX5Fw0qAbY2S7Za9ghdnc/6jZMkKw0ewq83232bUd+04h0upbXf+Lg8d8N2jLVj8XTTm2BT++FvKChmY1Ab79/Wmp0P8Xb8eD0F823LK
h9o+RuMwv+nMUV5YRNaSwKELhBAhw2inwEXFUXrw7FaouXBfKUSMnnOQULNcLWyuS66aFb1FVSA8q4v+oofqA3jSEAW8TEFuPNbVd4CWLrgLsu5VlWzITi9tN8VHBIHTPZVWzCvLoSnLqC6zKlg3uH7sQF0EBYZLNay9JaXa1PgP6gM6bjOALWRU1D
VogdP9NmcFMZxc0oSeb10TtKggssnh9/orS+9sk/Q9nr1m5/vYkbbk+t3dny8JyCly/XS2z8zMI80A6wrC5T8Jp1NXULUj695q48SgJF9UA2WEqZLBoArut9RE+mJryzgVdK+sOIskadE4F5FMZUEANXU151ReaOmfnKHInENVC13cpuSE7VdGEowyh
orZ096sa1bkFp+FFTSxFLJk8REwr/fgaJ3y27skAHU06Udnfku2U0PLrRMP0gC-g/E-enn04cGcasNybRPNXpYU0Z6TXb9F4y3Upa03PLmB20tV78bgX/op11JyjcFvXBNGERNzMKExF/h504fXmy8xWooBWhafAmvZfgo75mWjLlHtW36X75q3B6
F0BKuEFy6gP5j18Ve4VZfcbMputz4Syf7D21IqW1I/JDK/m4TX7WFnZj7atE0cXknJnZrM3UxP82fFqNKWS9HNX4aWZa9Xpxv1SLK66u056q1NuhW97TC9BE1dayLtyALH2RjPzdBf5jQnkySv1rmE75SDH7ceU/wRkV44Jww-WrSdiFvQoGy555oK
PpcJd00fEK981gmbAT3T6T5zXJG600H+0sdf43rRBruytQ3Yj3tejl+xbSMkZg3wbht7ZGVPsaw6D0Z385FT5UjU5nhigVThf1D1LaPGJuAcc/0FFr8YpKyMdvIm+m1D1ulCdpmhnanpI16Jr+SIg6DWRpDC95GUKYlbpXsvdtkJ5/Y3TaUdx5vGZKn
aTi1Uvge3cKpI2Cqyg0dGpt0lsonuac4/60QHVAc5aI8MzgzAMt9IOVZRV0aQ=', 'mac': 'opoEYJnt4X6u056f0qu+eU5Bw0kjzTH0/mLykjb1/68=', 'iv': '51o483hvtMUy+Y7', 'nonce': None, 'tag': 'WBPY9d9QKRY5NKe3GAGI
Bg='}
2019-11-17 17:51:47 pop-os root[14390] INFO Transferring Chunk
2019-11-17 17:51:47 pop-os root[14390] INFO Send: {'type': 'SECURE_X', 'payload': 'VnopTNYj9pheiG8TkEvU5n1u2LBlfX3tGF0AgLAPF0ZewV6RnYGLKiz+It3M60tng6eadQkU08grCbwp8ehzXe9kVzQuZ0z4rpjyA0z
cH6dNxwZLqZErVsoAsqRd744H3ooGhXXZFC2DX7T9tN8xygySPycHCvPIGTyGob29Bk00fgXsUkhfibtCtAZD9+oD/tuEzFNs0ZCgmm2+y3cF05xjckSpz+Lsc4niKAgu11v+FY/k05wkRejw8MkG28nyX1Q30SboGmU5lrcqMXGgW6FJ363uImb
Byrs3RwM33a8u5as9/59BFqIJD8NegLlp4+jyJe2AgIV+/t444WT81nAdZn3kJGrHRwLHhSn2AmHdG6e2a0ZUHTcm38EcpKGZ94QLtMcAtk25+Ym87F6LOMNFYJ01b9brglrFeHvMURD2NmfvRQ33oA8Gvq6r9Ibt5mWpVt0WHcLDvFukr7XddFDs
BdSt056GithVtCVBnQntef7f1J7CfwpuvHioK1mpPaUyk4s1721xYXtW3nuRnCLiBnxWect-fymZQ8UQprf0XVWkvd7pvgrSP5v6jDdZdJwLWuUc/LR2X1XJLEBfBT0yGxJhIcpmRDZVhtsqw7IV0sA800f7mW4LrdnFRLC/tbaahDjs3BUzclRK
K1h7mdNGu48nct2DHCCp16eKf2xv8pCyobekRpmvFdJPUTSf/jwxhLhPaMNHK33u8PazKeHoSPrgT5n6jhon/+rH9Va0Iap2aogggX0VhgzZBSAFDcbkg-sp5UnHa+3R0up39Z+xi0y2/cdgZpSx68GHZmq/t1V4/0vL6UDADz3NIJ7nxQEn9eY
uGegJEt2Za592Zv2AgCruaa5qmH/0HUMW2NC832C3S8UH9P4Yv9oGvLerbu7y9z0r053D7zVLDzh4RevmXxehFVtMaPseVKG++nPuKpc1vdHrJQBRZncW/reNCKbdopM3YWhIlg0GkR5YDrNiuPIp5H3c2en1yJqonaNlKuVvCiFyrtt
hMSXRNfZk013Wx/y7BoLht9HLL/FEZ10o322B1VUg+LsLCZt3EbvXVabPkvIdrUv6639cP4++nVRZhiZa7Zu-7ZFL0xzy+buubP5KCRH6UR6pFNsW7SFPZYVksdq6gY1PqegvU159TK1HhigH51VBGSUX//cFmWkDuJGLbcRRdM14bpB5Qy2
atXx6jPpoyZdCfA2U//ZLOXBwg64j7pshfraCW/geUeKlhhbvLqymM4NkuThnQ4CL/ngeRS383k-RVKNShH70wLTVnVeJdWGFuxaX0TYWPCP4+66KM/RF7FQkAhwcR0Yz5NE++DRFVFNQBA1HeVfPkab324NukV4LNV9c77A0ASD2i1BzN
GLEAFQ3Ct67QwnG5dW106hrw6uVHF3VTK0ddjzTfLgbtXIMLdUGaXorFwJv8F1M2IdS2J1b/8hoPc9p5XfWd0JvJaWfU', 'mac': 'oLIIM1XBTCGK6Dw3R80vYtRLaaDwnECbS9HAUTOwo=', 'iv': 'BU05H1756ceN5h7', 'nonce'
: None, 'tag': 'd9P48sdhoZjo0lB1f4afVw='}
2019-11-17 17:51:47 pop-os root[14390] INFO Send: {'type': 'SECURE_X', 'payload': 'dQe3E6d+ZjbPuxszN0e74+sD10RNRwTYFtL9SwoSxwS6S29iIrmvLP9bPwP/LJINCImAdXy/4XAT5Jcs2YH0lnZfP51aZKRzL4aIjyH1L
K/+kY1ew8nkGyfyQ8gcbz5Lk1z535m8Zy3DLP0T/vMK4g0ErgTqJ5UJ3hnpz=', 'mac': 'Dok02V1L5YX71KH26WdGx+kuYmmd4pY5F5E8U50FI=', 'iv': 'QXcXQKfL2bq1f5zG', 'nonce': None, 'tag': 'gF8Yq+Hcf1gRk8RdBd
ebrg='}
2019-11-17 17:51:47 pop-os root[14390] INFO File transferred. Closing transport
2019-11-17 17:51:47 pop-os root[14390] INFO The server closed the connection
```

Img 4.5 - Envio dos vários chunks do texto contido no ficheiro *rick.txt* do cliente para o servidor. Neste caso o cliente partiu o texto em dois Chunks. Após o envio dos chunks - incluindo mensagens do tipo **DATA**, encriptadas e encapsuladas em mensagens do **SECURE_X** - pelo cliente, este envia uma mensagem do tipo **CLOSE**, encriptada e encapsulada numa mensagem do tipo **SECURE_X**, visto ter efetivamente enviado todo o ficheiro pretendendo assim terminar a comunicação.



Img 4.6 - Ficheiro guardado pelo servidor no diretório files

Note-se que:

- Para facilitar a explicação das mensagens trocadas durante a execução dos programas, mudou-se o logger utilizado no método send de **INFO** para **DEBUG**. Numa execução normal não são apresentados os loggers das mensagens trocadas.
- Embora o ficheiro usado no exemplo foi do tipo .txt, o programa também foi testado com outros formatos (nomeadamente, pdf, mp3 e mp4), tendo obtido os mesmos resultados.

5. Conclusão

Chegando então ao término deste trabalho, podemos concluir que fomos capazes de cumprir todos os objetivos estabelecidos pelos docentes da disciplina, tendo aprofundado os conhecimentos de Criptografia utilizando Chaves Simétricas e negociação de um Segredo Partilhado, lecionados nas aulas teóricas.

6. Bibliografia e Referências

Para a realização deste trabalho, foi-nos crucial as informações fornecidas pelas seguintes fontes:

- Conjunto de Slides 3 da disciplina de Segurança Informática e nas Organizações
- Documentação Oficial do CryptographyIO
 - <https://cryptography.io/en/latest/>
- Secret Key Exchange (Diffie-Hellman) - Computerphile
 - <https://www.youtube.com/watch?v=NmM9HA2MOGI>