Projeto 2: Comunicações Seguras

Universidade de Aveiro
Segurança Informática e Nas Organizações 2019/2020

Trabalho realizado por:

- 88808 - João Miguel Nunes de Medeiros e Vasconcelos
- 88886 - Tiago Carvalho Mendes

Novembro de 2019

1. Introdução

O presente documento tem como principal objetivo descrever detalhadamente a solução desenvolvida tendo em conta os objetivos propostos para o segundo projeto da unidade curricular de Segurança Informática e Nas Organizações da Universidade de Aveiro, considerando o seu planeamento, desenho, implementação e validação tendo em conta o código fornecido como base para o trabalho.

No guião de apresentação deste segundo projeto, era pedido o *planeamento*, *desenho*, *implementação* e *validação* de um protocolo que permita a **comunicação segura** (confidencial e íntegra) entre dois pontos, nomeadamente **um cliente e um servidor** através de uma ligação por **sockets TCP**, em Python.

2. Planeamento

2.1 Objetivos do trabalho

De modo a planear a solução a desenvolver, é necessário considerar **os seguintes aspetos**, presentes no guião de apresentação do projeto:

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
    - e) Suporte de pelo menos duas cifras simétricas (ex: AES e Salsa20)
    - f) Dois modos de cifra (ex: CBC e GCM)
    - g) Dois algoritmos de síntese (ex: SHA-256 e SHA-512)
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.

Outros aspetos a considerar são **os seguintes**:

- **Implementação de funções genéricas** de cifra/decifra/cálculo de um MAC/verificação de um MAC de textos
- **Criação de novos tipos de mensagens** a enviar, incluindo as mensagens já existentes dentro do conteúdo destas novas mensagens (num formato cifrado e íntegro).

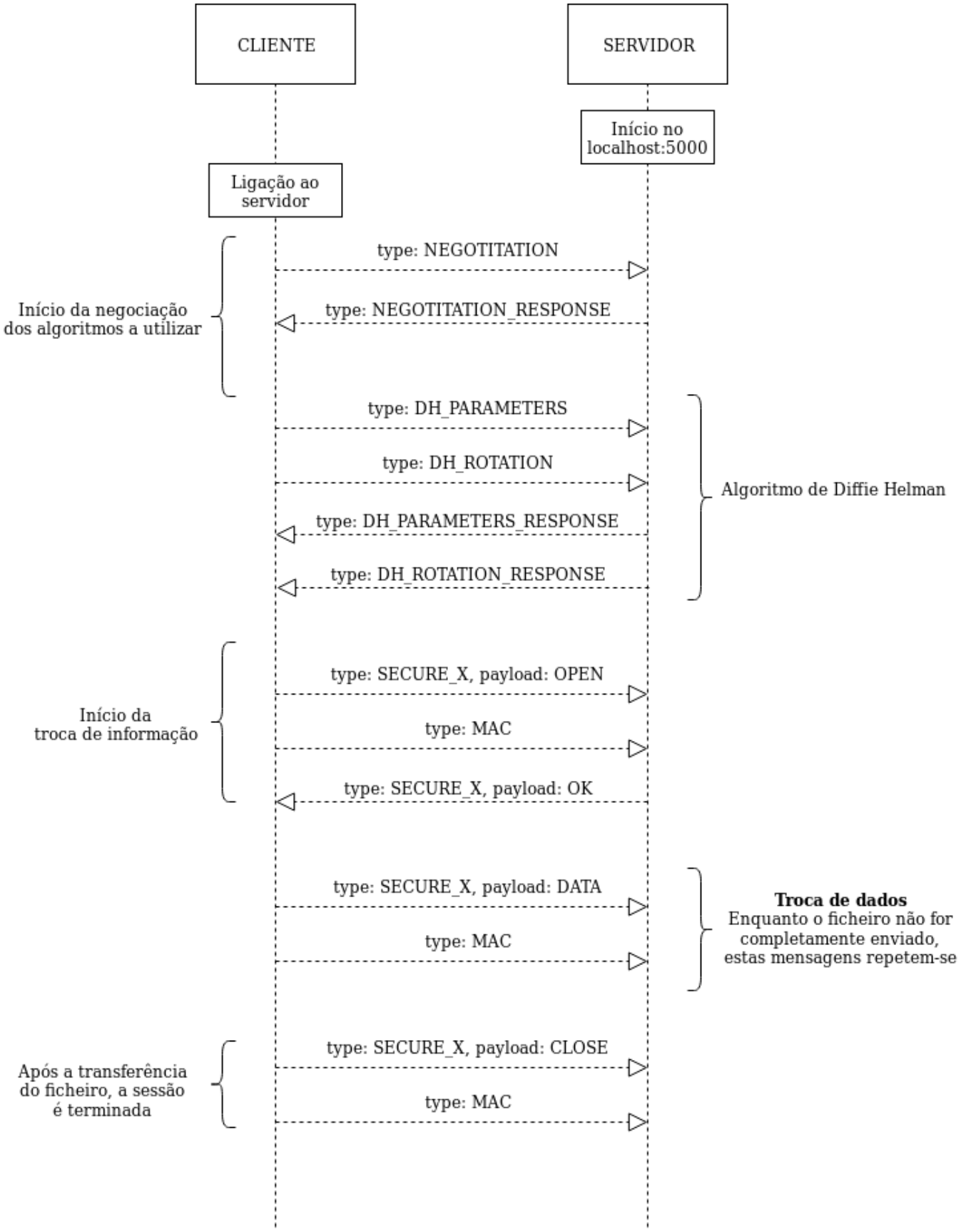
2.2 Fluxo de troca de mensagens

Para cumprir os objetivos pretendidos com a realização deste projeto, primeiro definimos qual seria o **fluxo de troca de mensagens**, que de seguida iremos explicar. Este fluxo está dividido em 5 fases distintas:

1. Início da negociação dos algoritmos a utilizar
2. Incorporação do algoritmo de **Diffie Hellman**
3. Início da troca de informação segura através de uma **mensagem *OPEN*** cifrada.
4. Envio de pedaços (*chunks*) de um ficheiro através de várias **mensagens *DATA*** cifradas.
5. Término da sessão após a transferência completa do ficheiro através de uma **mensagem *CLOSE*** cifrada.

Nota: De realçar que as mensagens cifradas são seguidamente acompanhadas de uma **mensagem do tipo MAC**, com o intuito de controlar a integridade das mesmas.

De seguida, apresenta-se um **diagrama de sequências UML**, ilustrando todas as mensagens trocadas entre o *cliente* e o *servidor*:



3. Negociação dos algoritmos utilizados

A sessão entre o *cliente* e o *servidor* inicia-se com a negociação do **algoritmo de cifra**, **modo de cifra** e **função de síntese** a utilizar. Para tal, o cliente informa o servidor dos algoritmos que possui através de uma mensagem do tipo `NEGOTIATION`:

```
algorithms = dict()
algorithms['symetric_ciphers'] = self.symetric_ciphers
algorithms['chiper_modes'] = self.cipher_modes
algorithms['digest'] = self.digest

message = {'type': 'NEGOTIATION', 'algorithms': algorithms}
self._send(message)
```

O *servidor*, ao receber e processar esta mensagem, verifica quais os algoritmos deste conjunto que tem disponíveis e informa o *cliente* através de uma mensagem do tipo `NEGOTIATION_RESPONSE` quais os algoritmos escolhidos:

```
chosen_algorithms = dict()
chosen_algorithms['symetric_ciphers'] = self.crypto.symmetric_cipher
chosen_algorithms['chiper_modes'] = self.crypto.cipher_mode
chosen_algorithms['digest'] = self.crypto.digest
message = {'type': 'NEGOTIATION_RESPONSE','chosen_algorithms':
chosen_algorithms}
self._send(message)
```

Nota: A variável `self.crypto` é um objeto da classe `Crypto`, desenvolvida por nós, com todo o processamento criptográfico da nossa solução.

Após receber a mensagem com os algoritmos a utilizar durante a sessão, o *cliente* termina a etapa de **negociação de algoritmos** e dá início ao processo de **troca de chaves** através do algoritmo de **Diffie Hellman**.

De seguida seguem-se capturas de ecrã do funcionamento desta etapa, tanto no *cliente* como no *servidor*.

![cliente] ![servidor]

4. Troca de chaves utilizando o algoritmo Diffie Hellman

No seguimento do ponto anterior, o *cliente* inicia o processo de **troca de chaves** através do algoritmo de **Diffie Hellman**.

NÃO TERMINADO

5. Confidencialidade

Após a troca de chaves descrita no ponto anterior, o *cliente* dá início à troca de informação através do envio de uma mensagem do tipo `OPEN`:

```
message = {'type': 'OPEN', 'file_name': self.file_name}
```

No entanto, esta forma de enviar a mensagem **não é de todo segura.** Portanto, e visto que se pode proceder à **encriptação** e **desencriptação** através das chaves simétricas partilhadas, o *cliente* irá enviar uma nova mensagem do tipo `SECURE_X`, que irá ter como `'payload'` a mensagem do tipo `OPEN` encriptada:

```
secure_message = {'type': 'SECURE_X', 'payload': None}
payload = json.dumps(message).encode()
criptogram = self.crypto.file_encryption(payload)
secure_message['payload'] = base64.b64encode(criptogram).decode()
self._send(message)
```

Nota: Com o intuito de simplificar a explicação, este pedaço de código foi adaptado, não estando rigorosamente igual ao da solução entregue.

A função `self.crypto.file_encryption(payload)`, semelhante à desenvolvida nas aulas práticas da unidade curricular, encripta um conjunto de **bytes** segundo o *algoritmo de cifra simétrica* e o *modo de cifra* escolhidos no *processo de negociação*, corrigindo o **block_size** do último bloco através de um **padding,** retornando por fim o criptograma.

O *servidor*, ao receber a mensagem do tipo `SECURE_X`, guarda o conteúdo do campo `'payload'` na variável `self.encrypted_data` (que será útil na transferência do ficheiro, como explicado mais à frente). De seguida, e **só após** confirmar a **integridade da mensagem** (também explicado mais à frente), o *servidor* desencripta o `'payload'` e processa a mensagem, neste caso do tipo `OPEN`, com o código já fornecido (podendo ser também do tipo `DATA` ou `CLOSE`).

Esta lógica de encriptação e desencriptação de mensagens está implementada tanto no *cliente* como no *servidor*. Assim, pode-se garantir a **confidencialidade** das mensagens trocadas.

6. Controlo de integridade