Ano lectivo: 2020/2021

Segurança Informática e nas Organizações

Projeto 2 + 3

Criptografia Aplicada à Gestão de Direitos Digitais

Daniel Pinto 89341

Juan Lessa 91359

Índice

1. Confidencialidade e Integridade

1.1. Protocolo

Para garantir a confidencialidade e integridade das mensagens trocadas entre cliente e servidor utilizamos o seguinte esquema:

Ao estabelecerem conexão, cliente e servidor fazem uma troca de chaves Diffie-Hellman (através de um get para o endpoint */api/dh-parameters*, em seguida um post para o endpoint */api/dh-handshake)* para estabelecer um segredo em comum. Tais chaves DH são utilizadas apenas durante uma sessão.

Em seguida, o servidor envia uma lista com os algoritmos de symmetric cryptography, cipher modes e hash functions suportados (através de um get para o endpoint */api/cipher-suite*) para que o cliente escolha quais pretende utilizar na comunicação. O cliente escolhe aleatoriamente 1 algoritmo, 1 cipher mode e 1 hash function dentre as opções.

A partir desse ponto toda a comunicação entre cliente e servidor é criptografada usando o algoritmo e cipher mode escolhidos pelo cliente. A key utilizada para encriptar a comunicação será obtida derivando o segredo comum obtido com DH.

1.2. Cipher Suite

As seguintes cifras podem ser escolhidas pelo cliente para criptografar a comunicação:

Symmetric Cryptography:

- AES-128
- Triple DES
- ChaCha20

Cipher Mode:

- ECB
- CFB
- CBC
- OFB

Hash Function:

- SHA-256
- SHA-384
- SHA-512
- MD5
- BLAKE-2

Observação: caso o cliente escolha o ChaCha20 como algoritmo para criptografar a comunicação, o cipher mode será None, tendo em conta que o ChaCha20 não faz uso de cipher mode.

1.3. Métodos para Cifragem

Para realizar toda a cifragem de dados, desenvolvemos alguns métodos e armazenamos na pasta *utils*.

- symmetriccrypt.py

  Este ficheiro contém os métodos que criamos para encriptar mensagens, desencriptar mensagens, desencriptar ficheiros e gerar chaves (usando derivação).

- diffiehellman.py

  Este ficheiro contém os métodos que criamos para gerar chaves DH efêmeras e para calcular um segredo comum através das chaves.

- hashfunction.py

  Este ficheiro contém os métodos que criamos para gerar o hash de uma mensagem.

- rsa_utils.py

  Este ficheiro contém os métodos que criamos para gerar chaves, pares de chaves RSA, carregar chaves públicas ou privadas, assinar uma mensagem, verificar assinatura e encriptar ou desencriptar mensagens.

2. Autenticação e Isolamento

2.1. Autenticação Mútua com Cartão de Cidadão

Para a Realização da autenticação mútua foi proposto a utilização de uma PKI à nossa escolha, tendo sido o Cartão de Cidadão a escolha feita.

2.1.1. Protocolo

O protocolo começa com o cliente a gerar um nonce de 64 bits que de seguida é enviado para o servidor através de um post para o endpoint */api/server_auth*.

O servidor ao receber este nonce, assina-o com a chave privada do seu certificado e retorna-o ao cliente juntamente com o seu certificado e um novo nonce de 64 bits gerado pelo mesmo.

O cliente recebe o certificado retornado do servidor e constrói uma cadeia de certificação juntamente com a CA do servidor. De seguida valida a mesma aferindo se os certificados estão dentro do prazo de utilização, os seus estados de revocação, os seus propósitos e as assinaturas de cada certificado.

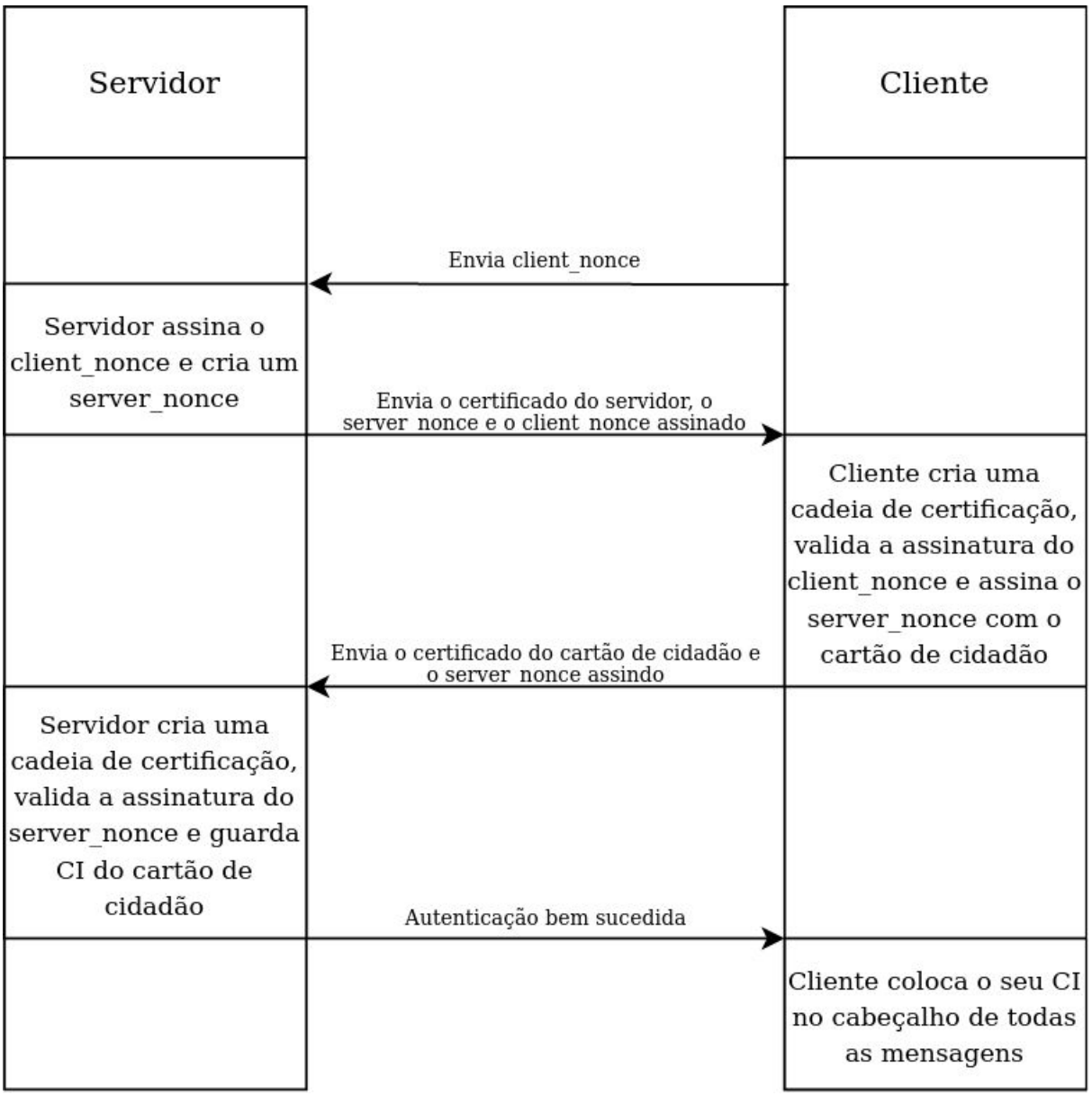
A assinatura do nonce gerado pelo cliente é validada e de seguida é iniciada uma sessão com o cartão de cidadão. Uma vez com a sessão iniciada, o cliente assina o nonce gerado pelo servidor e envia esse mesmo nonce juntamente com o certificado do cartão de cidadão para o servidor através de um post para o endpoint */api/client_auth.*

À semelhança do cliente, o servidor constrói uma cadeia de certificação com o certificado do cartão de cidadão e as CA's do mesmo, procedendo depois às validações.

Para verificações futuras, o servidor guarda o número de identificação civil do cliente e retorna ao cliente a confirmação ou rejeição da autenticação.

Consoante a resposta do servidor, o cliente passará a utilizar o seu número de identificação civil nos cabeçalhos de todas as mensagens.

2.1.2. Diagrama do Protocolo



2.2. Autenticar o Utilizador que Consome Conteúdos

Como referido anteriormente, no final do processo de autenticação mútua, o servidor guarda o número de identificação civil do utilizador e o cliente passa a comunicar com o servidor através de mensagens identificadas com esse mesmo número de identificação no header.

Na prática, no final da autenticação mútua, o servidor irá sempre verificar se as mensagens que lhe chegam são provenientes de um utilizador que está autorizado a consumir conteúdos.

Caso um utilizador não seja autorizado, o servidor devolve para o cliente um *status code* 401.

2.3. Autenticar o Conteúdo que é Consumido

A autenticação do conteúdo que é consumido é realizada através da assinatura do conteúdo com chaves RSA.

2.3.1. Protocolo

Imediatamente depois da autenticação mútua ser finalizada, inicia-se uma troca de chaves RSA.

O cliente começa por gerar um par de chaves RSA e guarda-as em ficheiros.

De seguida envia a chave pública gerada para o servidor através de um post para o endpoint */api/rsa_exchange.*

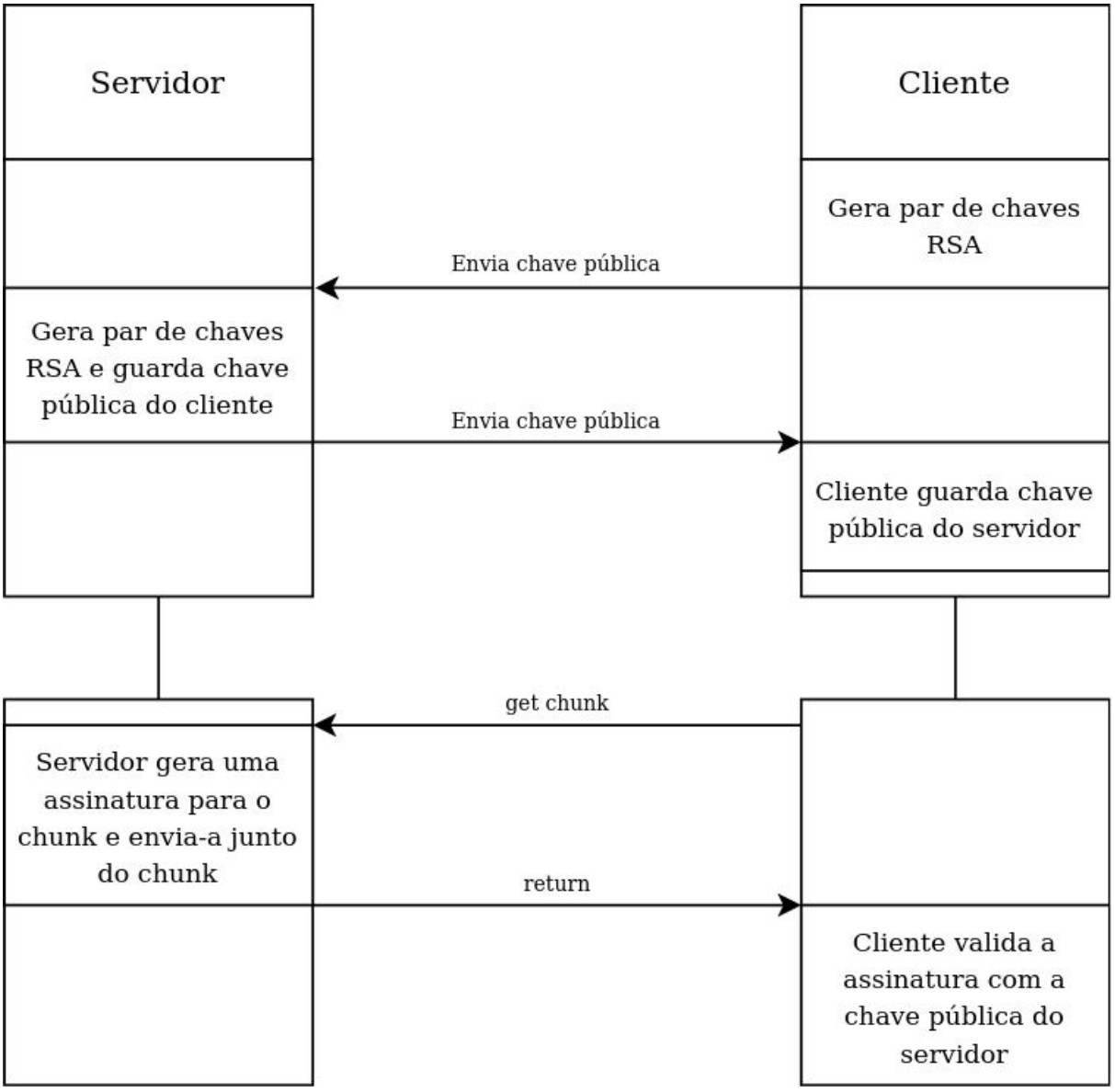
Assim como o cliente, o servidor gera um par de chaves RSA e guarda-as em ficheiros assim como a chave pública do cliente.

O Servidor retorna a sua chave pública para o cliente e este também a guarda num ficheiro.

Mais tarde, sempre que é feito um pedido get de um chunk do ficheiro áudio, o servidor irá gerar uma assinatura para esse chunk, com a sua chave privada, e irá juntá-la à mensagem a ser retornada para o cliente.

Uma vez com a resposta do servidor, o cliente irá validar a assinatura do chunk com a chave pública do servidor, podendo proceder ao processamento do chunk caso a assinatura seja válida.

2.3.2. Diagrama do Protocolo



2.4. Proteger o Conteúdo no Servidor

De maneira a proteger o conteúdo que está no servidor o mesmo foi previamente encriptado utilizando AES-128 e CBC.

De maneira a que o utilizador possa usufruir do conteúdo, quando é feita uma primeira chamada para o endpoint */api/download* o servidor irá desencriptar o ficheiro para um novo com o mesmo nome e adiciona ainda um caracter no final, ficando o ficheiro com um nome do tipo “.mp31”.

Durante todas as chamadas ao servidor, este irá ler do ficheiro desencriptado sendo que, quando o utilizador terminar de ouvir, é feito um get para o endpoint */api/finished*. A partir deste endpoint o servidor recebe instruções para eliminar o ficheiro desencriptado do sistema, restando apenas o ficheiro encriptado.