Universidade do Minho

Mestrado em Engenharia Informática

Engenharia de segurança

Prática 2 - PD1

Grupo 1

Rui Carlos Azevedo Carvalho - PG47633
Daniel Barbosa Miranda - PG47123
Ana Luísa Lira Tomé Carneiro - PG46983

Contents

Introdução

O objetivo deste projeto é disponibilizar um serviço de cofre digital, garantindo que os documentos lá depositados sejam exclusivamente acedidos pelos seus titulares. Desta forma foi necessário a implementação de um cofre digital que permitisse depositar documentos que pudessem ser acedidos pelas entidades identificadas pelo depositante. Caso o depositante identifique que o documento só pode ser acedido por várias pessoas em conjunto, então é necessário a implementação do esquema de Shamir que permite a partilha de um segredo por um conjunto de pessoas. Além da funcionalidade de depósito é necessário que o cofre digital também implemente um método que permita ao utilizador retirar o documento do cofre desde que este tenha acesso autorizado ao documento.

Este projeto foi implementado em Python como recurso às bibliotecas Pycryptodome e Cryptograhpy que foram utilizadas para implementação de algoritmos de cifra simétrica e assimétrica, protocolos de troca e partilha de chaves, funções de *hash* criptográficas, entre outros.

Implementação do Problema

Este problema baseou-se em duas entidades principais: o **cliente** que representa todas as funcionalidades a serem implementadas do lado do utilizador do cofre e o **servidor** que consiste em todos os métodos necessários para implementar o cofre digital. A comunicação entre estas duas entidades inicia-se com o protocolo Diffie-Hellman para que seja estabelecida uma chave de sessão partilhada entre estas duas entidades. Esta chave será utilizada para a cifra e decifra dos pedidos e respostas enviados entre o cliente e o servidor. Quando um cliente pretende depositar um documento este envia para o servidor o documento a armazenar e a sua chave pública. Esta chave pública que foi gerada a partir do algoritmo RSA será utilizada para cifrar a chave utilizada para cifrar o documento no cofre digital. O servidor ao receber este pedido vai armazenar o documento cifrado no sistema de armazenamento enviando de volta ao cliente a chave que decifrar o documento armazenado juntamente com a *hash* do documento. Estes parâmetros serão utilizados para que o cliente, posteriormente, consiga retirar o documento do cofre.

De seguida apresentamos a explicação de cada funcionalidade deste problema assim como a explicação da implementação do algoritmo de Diffie-Hellman e do sistema de armazenamento. No final, apresentamos o modo de utilização do programa implementado.

2.1 Criação de um canal seguro entre cliente e servidor

Após a aplicação do cliente estabelecer uma conexão com o servidor através de um socket TCP, é realizado um processo que torna o canal de comunicação entre estas duas entidades um canal seguro.

Para isto, tanto a aplicação do cliente como a aplicação do servidor possuem chaves públicas e privadas associadas ao esquema criptográfico Ed448, que é um esquema de curvas elípticas que utiliza o EcDSA para realizar assinaturas. Numa primeira instância, e após se estabelecer a ligação, as duas entidades troca as suas chaves públicas, sendo que, estas serão utilizadas para assinar os pacotes trocados no acordo de uma chave, que é secreta e partilhada pelo cliente e servidor, de modo a garantir que estes pacotes são enviados pelo agente correto (servidor ou cliente) e não uma outra entidade que possa querer realizar um ataque.

Já para o processo de acordo de uma chave entre ambos, é utilizado o esquema Diffie-Hellman com a curva elíptica "curve448". Desta forma, o cliente envia um pacote com a sua chave pública, sendo esta assinada e verificada na receção pelo servidor. O servidor faz o mesmo mas no sentido contrário. De seguida, ambos derivam uma chave comum

através da aplicação do seu segredo privado (chave privada) e aquilo que receberam da outra entidade, sendo a chave derivada uma chave com 32 bytes, para ser utilizada na cifra das mensagens com recurso ao algoritmo AES no modo GCM.

Por fim, ambas as entidades confirmam a chave que foi derivada trocando entre ambos o hash da chave derivada, sendo este comparado com o hash da chave gerada. Assim ambos confirmam que a chave que derivaram é a mesma e as mensagens trocadas passam a ser cifradas com esta. A figura abaixo demonstra as trocas de chaves que existem entre as duas entidades:

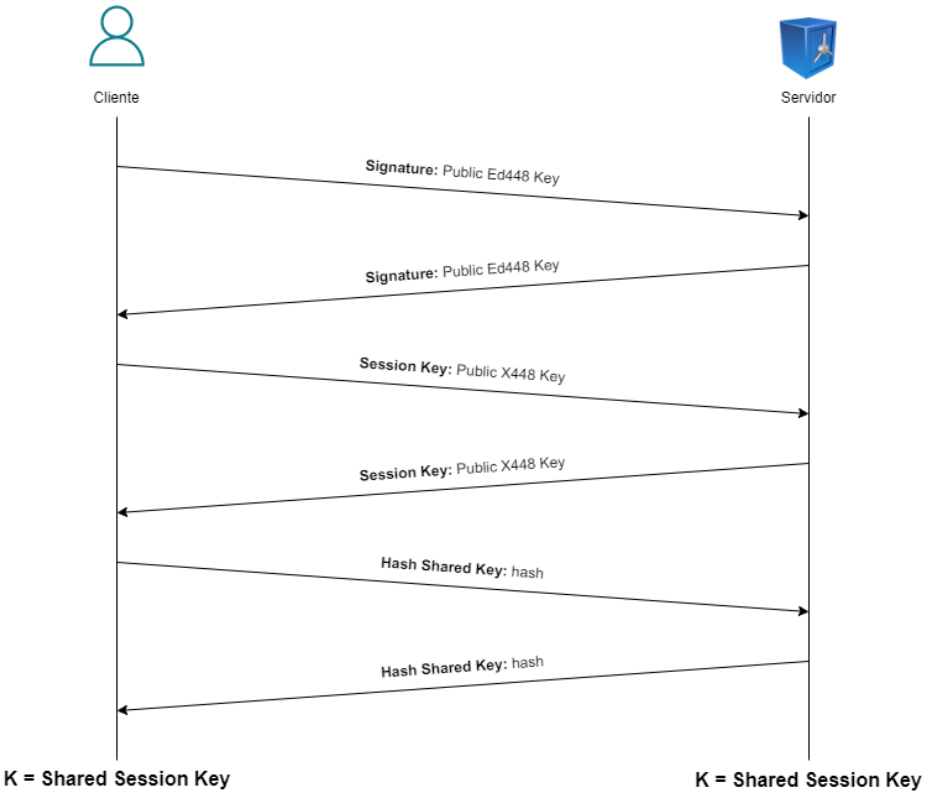

Figure 2.1: Diagrama do estabelecimento de um canal seguro entre cliente e servidor

2.2 Sistema de Armazenamento

O sistema de armazenamento do servidor está divido, essencialmente, em duas partes. A primeira que trata de criar um novo registo para um novo ficheiro que o cliente pretenda depositar e outra para armazenar pedidos de obtenção de um ficheiro que necessite de um mínimo número de pessoas para o obter.

De facto, o sistema do servidor implementa uma base de dados SQLlite3 na qual cada linha representa um registo de um ficheiro. Este registo é composto pelo hash do ficheiro, o valor de n e m (caso seja aplicável), nounce e a tag obtidas no processo de cifra do ficheiro. Na tabela abaixo é possível verificar um exemplo do estado da base de dados com dois ficheiros registos, sendo que, o primero representa um depósito de um ficheiro que pode ser acedido apenas pela pessoa que o depositou (valores de n e m nulos) e o segundo necessita de um mínimo de 2 pessoas para ser obtido, sendo a chave de cifra repartida em 5 chaves secretas:

fileHash	n	m	nounce	tag
c2151asdfasdckjasdifj	NULL	NULL	b'\xa\fdabs'	b'\xa15\asd'
c2151asdfasdckjasdifj	2	5	b'\xa\sdgfqwe'	b'\asd\rwerasd'

A segunda estrutura de dados utilizada no sistema do servidor é uma que é utilizada para os ficheiros que necessitem de mais do que uma chave para ser obtido. Neste caso utilizou-se um dicionário que armazena para faz corresponder a um determinado hash de um ficheiro uma lista de pares (índice, chave) e um contador. Desta forma, quando um utilizador pretende obter um ficheiro tanto o índice como a chave fornecida são armazenadas nesta lista de pares até que o número mínimo de pessoas seja atingido sendo aplicada a combinação das chaves através do algoritmo de shamir e o ficheiro é posteriormente fornecido a todos os clientes cujo índice e chave estavam nesta lista. Caso ainda não se tenha atingido o número mínimo de pessoas então a thread que foi criada para tratar o pedido do cliente no lado do servidor fica bloqueada e apenas será ativada quando um dos próximos clientes que pretendem aceder ao mesmo ficheiro indicarem mais um uma chave secreta.

2.3 Funcionalidades

2.3.1 Depositar Documento

Esta funcionalidade consiste em depositar um documento que só será acedido pelo o depositante. Para isso é necessário que o utilizador indique a sua chave pública gerada através do algoritmo RSA e o documento que pretende armazenar. De seguida apresenta-se o diagrama de sequência representativo da comunicação estabelecida entre o cliente e o servidor para implementar a funcionalidade.

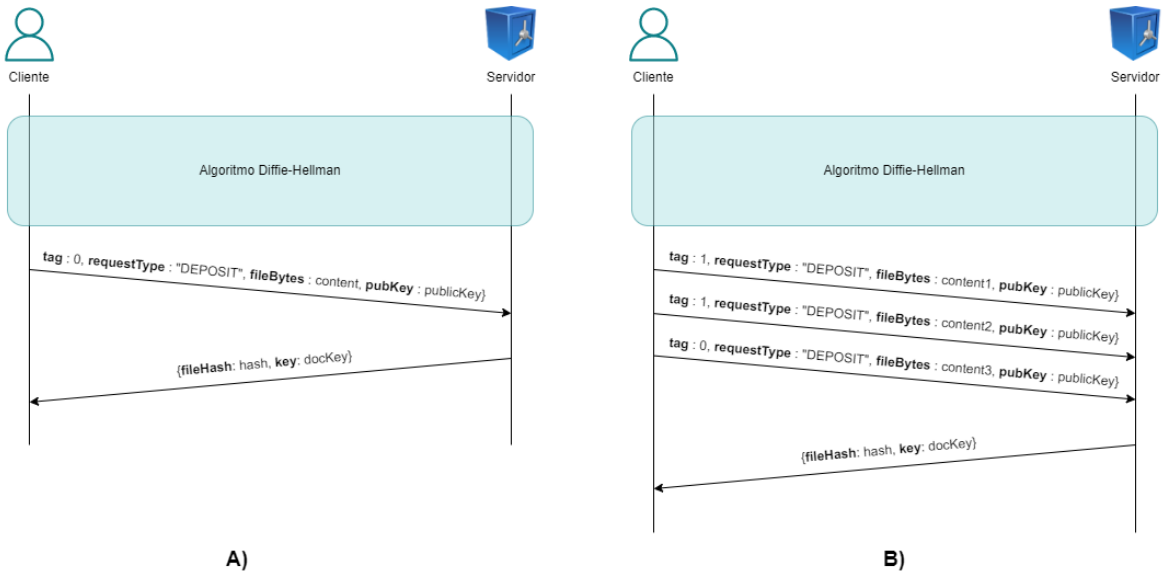

Figure 2.2: Pacotes trocados entre o cliente e o servidor quando pretendemos depositar A) um ficheiro pequeno ou B) um ficheiro grande

Cliente - *depositHandler*

A função *depositHandler* começa por receber a chave pública do utilizador e o documento que pretendemos armazenar. Como o *socket* da comunicação entre o cliente e o servidor só consegue enviar 4096 bytes por cada pacote é necessário verificar se o documento que pretendemos armazenar terá de ser fragmentado em vários pacotes ou se podemos enviar o conteúdo do documento num só pacote. Para isso verifica-se se o documento tem um tamanho maior que 3000 bytes dando cerca de 1096 bytes de espaço para o cabeçalho da mensagem. Caso o documento tenha um tamanho menor que 3000 bytes então todo o conteúdo é enviado num só pacote ao servidor tal como está representado na figura 2.3.1 A). Caso contrário, através da função *sendFile*, fragmenta-se o conteúdo do documento em blocos de 2000 bytes permitindo desta forma enviar o conteúdo em vários pacotes. Para que o servidor tenha conhecimento da quantidade de pacotes enviados com o documento fragmentado é enviado no pacote uma *tag* que terá valor 1 caso ainda existam pacotes a serem enviados com o documento e terá valor 0 caso seja o último pacote com o documento fragmentado, tal como vemos na figura 2.3.1 B).

É de notar que a função *send* vai não só enviar os pacotes pelo *socket* como também vai cifrar todas as mensagens enviadas do cliente para o servidor utilizando a chave da sessão partilhada entre o servidor e o cliente. Da mesma formam, todas as resposta obtidas do servidor foram cifradas com a chave da sessão, sendo os pacotes que chegam ao cliente decifrados com a mesma chave na função *receive*.

O cliente quando receber a resposta do servidor vai receber a *hash* do documento que foi armazenada assim como a chave que decifra o documento cifrada com a sua chave pública que foi enviada. Esta informação será apresentada ao utilizador que, posteriormente, será usada para que este tenha acesso ao ficheiro.

Servidor - *depositHandler*

No lado do servidor, este recebe o pedido e gera uma chave que será utilizada para cifrar o ficheiro através do algoritmo simétrico `AES` no modo `EAX`, dando origem a um *ciphertext*, um valor *nounce* e uma *tag*. Ao *ciphertext* é aplicado uma função de *hash* (SHA256) de forma a gerar a hash do documento que será armazenada juntamente com o ficheiro cifrado, o valor *nounce* e a tag no sistema de armazenamento.

Por fim, cifra-se, com `PKCS OAEP`, a chave utilizada para cifrar o ficheiro utilizando a chave pública fornecida pelo cliente e, envia-se o resultado dessa operação em conjunto com a *hash* gerada ao cliente que solicitou o pedido.

2.3.2 Depositar Documento para múltiplos utilizadores

Esta funcionalidade consiste em depositar um documento que será acedido por um conjunto de utilizadores. Para isso é necessário que o utilizador indique as chaves públicas de todos os utilizadores que podem aceder ao documento, documento que pretende armazenar, o número de utilizadores que podem aceder ao documento e o número de utilizadores necessários para aceder ao documento. De seguida apresenta-se o diagrama de sequência representativo da comunicação estabelecida entre o cliente e o servidor para implementar a funcionalidade.

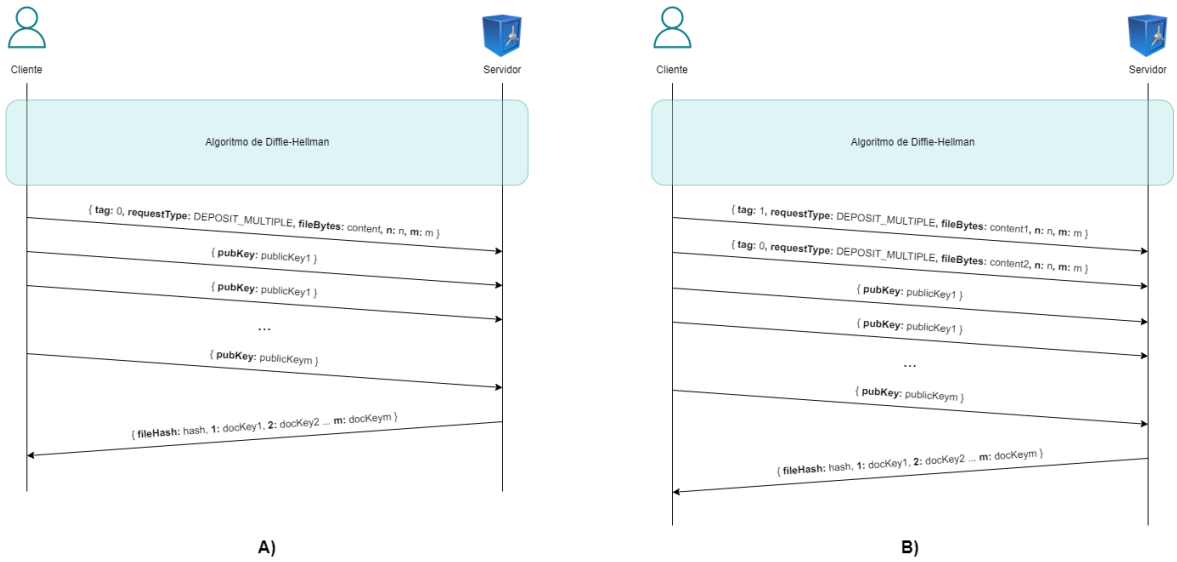

Figure 2.3: Pacotes trocados entre o cliente e o servidor quando pretendemos depositar A) um ficheiro pequeno ou B) um ficheiro grande que seja acedido por um conjunto de utilizadores

Cliente - *depositMultipleHandler*

A função *depositMultipleHandler* começa por receber as chaves públicas dos utilizadores que poderão aceder ao documento, o documento que pretendemos armazenar, o número de pessoas que podem aceder ao documento e o conjunto de utilizadores que são necessários para aceder ao ficheiro. Tal como acontece no *depositHandler* também nesta função é necessário verificar o tamanho do documento e aplicar as mesmas funções para que todo o documento, fragmentado ou não, chegue ao servidor tal como está representado na figura 2.3.2 A) e B). De seguida é necessário enviar ao servidor as várias chaves públicas dos utilizadores que puderam aceder ao ficheiro.

O cliente quando receber a resposta do servidor vai receber a *hash* do documento que foi armazenada assim como as chaves, que vão permitir decifrar o documento, de todos os utilizadores que tem acesso ao ficheiro. As chaves foram todas cifradas com a sua chave pública de cada utilizador. Esta informação será apresentada ao utilizador que, posteriormente, será usada para que este tenha acesso ao ficheiro.

Servidor - *depositMultipleHandler*

Esta funcionalidade é análoga ao *depositHandler*, distinguindo-se, primeiramente, pelo facto de que, em conjunto com o ficheiro cifrado é guardado no sistema de armazenamento o *quorum* (`n`) e o número de partes (`m`).

Numa segunda instância, aplica-se o algoritmo de partilha de segredo de *Shamir*, à chave utilizada na cifra do ficheiro, utilizando como parâmetros o *quorum* e o número de partes, sendo o seu resultado os pares das partes e os seus respetivos índices. Cada uma dessas partes é cifrada com `PKCS OAEP`, utilizando as diferentes chaves públicas recebidas pelo cliente. Por fim, cada par `(indíce, parte cifrada)` é adicionado numa lista, sendo essa lista e o *hash* enviados ao cliente que requisitou o pedido.

2.3.3 Obter Documento

Esta funcionalidade consiste em aceder a um documento que já foi depositado no cofre. Para isso é necessário que o utilizador indique a sua chave privada para aceder ao documento, a *hash* do documento que pretende obter e o caminho a onde quer armazenar o documento obtido. De seguida apresenta-se o diagrama de sequência representativo da comunicação estabelecida entre o cliente e o servidor para implementar a funcionalidade.

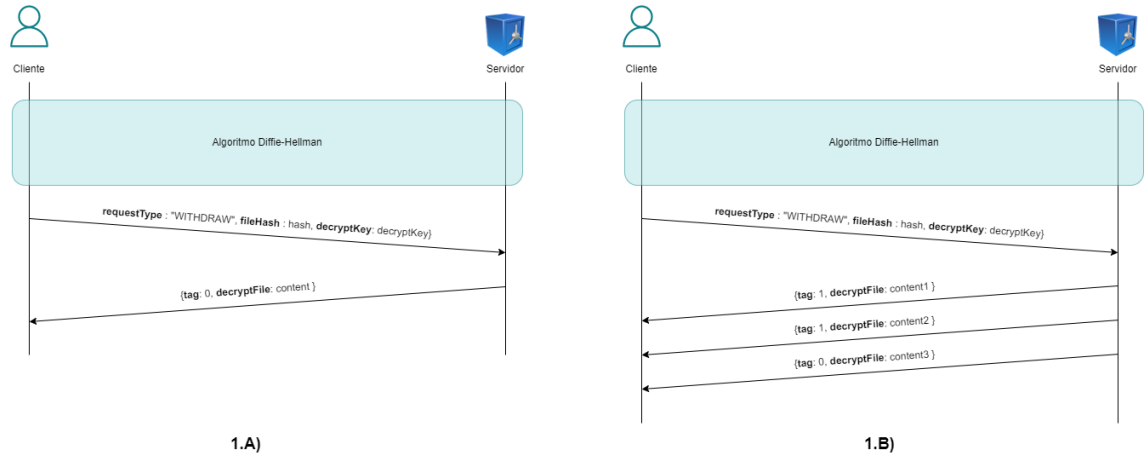

Figure 2.4: Pacotes trocados entre o cliente e o servidor quando pretendemos aceder a um 1.A) ficheiro pequeno ou a um 1.B) ficheiro grande que seja acedido por um só utilizado.

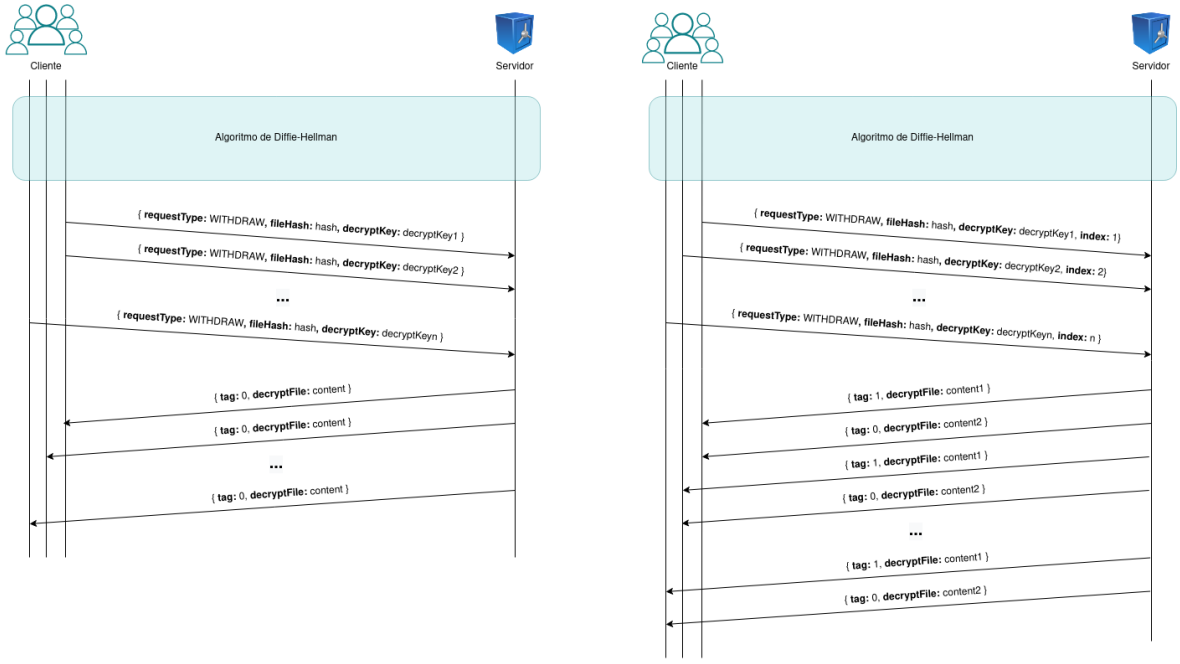

Figure 2.5: Pacotes trocados entre o cliente e o servidor quando pretendemos aceder a um 2.A) ficheiro pequeno ou a um 2.B) ficheiro grande que seja acedido por um conjunto de utilizadores

Cliente - *withdrawHandler* e *withdrawMultipleHandler*

A função *withdrawHandler* começa por receber a chave que decifra o documento num só ficheiro, a *hash* do documento que queremos aceder e o caminho onde armazenamos o documento obtido. Já a *withdrawMultipleHandler* necessita também de receber o índice a que essa chave corresponde.

Tal como podemos ver na figura 2.3.3 1.A) podemos ver que após enviarmos o pedido para o servidor este vai nos responder com o documento decifrado que será armazenado no caminho indicado como parâmetro. Caso o documento decifrado seja maior que 3000 bytes então o servidor enviar vários pacotes com o conteúdo fragmentado. Esse conteúdo será depois concatenado no lado do cliente para assim obter o conteúdo original (2.3.3 1.B)). Caso contrário, o servidor vai enviar um único pacote com o documento decifrado (2.3.3 1.A)).

Segundo o esquema de Shamir para que o servidor consiga decifrar e aceder ao documento pedido é necessários que n clientes (com o n determinado na função *depositMultipleHandler*) enviem a sua chave que consegue decifrar o documento. Tal como podemos ver na figura 2.3.3 depois dos n clientes enviarem a sua chave (e o respetivo índice da chave) o servidor consegue responder com o documento decifrado. Caso o documento decifrado seja maior que 3000 bytes então o servidor enviar vários pacotes com o conteúdo fragmentado. Esse conteúdo será depois concatenado no lado do cliente para assim obter o conteúdo original (2.3.3 2.B)). Caso contrário, o servidor vai enviar um único pacote com o documento decifrado (2.3.3 2.A)).

Servidor - *withdrawHandler* e *withdrawMultipleHandler*

Do lado do servidor a obtenção simples de um ficheiro, isto é, que só precise uma chave para ser decifrado é iniciada pela obtenção do conteúdo do ficheiro cifrado através do hash que foi enviado pelo cliente. Obviamente, se o ficheiro não existir é enviado para o cliente uma mensagem de erro. De seguida, decifra-se o conteúdo do ficheiro e envia-se de volta ao cliente. Caso o ficheiro tenho uma tamanho superior a 3000 bytes então é aplicada a fragmentação tal como está descrita no cliente.

Caso seja um ficheiro que necessite de n chaves, então utiliza-se a estrutura de dados indicada no ponto referente ao sistema de armazenamento do sistema que irá guardar o índice e a chave que este utilizador indicou. A conexão irá ficar bloqueada até que se atinga o número mínimo de pessoas para decifrar este ficheiro.

2.4 Modo de Utilização

De forma a utilizar o programa devemos começar por correr o servidor através do comando:

```
python3 vault-d.py
```

De seguida, corremos o cliente através do seguinte comando:

```
python3 vault-c.py -d publicKey -f filePath
```

Com este comando depositamos o documento *filePath* que só pode ser acedido pelo o depositante utilizando a chave pública do utilizador gerada a partir do algoritmo RSA que se encontra no ficheiro *publicKey*.

Para gerar este ficheiro com a chave RSA utiliza-se os seguintes comandos:

```
openssl genrsa -out [Chave privada] 2048
openssl rsa -in [Chave privada] -pubout -out [Chave publica]
```

O primeiro gera uma chave privada RSA com 2048 bits que será armazenada no ficheiro indicado em [Chave privada]. Já o segundo gera a chave pública a partir da privada.

O comando do vault-c.py apresentado devolve a *hash* do ficheiro e a chave *docKey* que decifra o documento cifrada com a chave pública do utilizador. Caso queiramos depositar um documento que possa ser acedido por um conjunto de utilizadores então utilizamos o comando:

```
python3 vault-c.py -f filePath -s n m -d pubKey1 pubKey2 ...
```

Neste comando podemos ver que o documento *filepath* poderá ser acedido por m utilizadores contudo será necessário n entidades para conseguir decifrar o documento. As chaves públicas dos m utilizadores estão, respetivamente, nos ficheiros *pubKey1*, *pubKey2*, ... , *publicKeym*. O comando devolve a *hash* do ficheiro e as chaves *docKey* que decifram o documento cifradas com a chave pública de cada utilizador.

As chaves que provém do servidor encontram-se cifradas e como tal é necessário decifrá-las através do seguinte comando:

```
openssl rsautl -decrypt -in [Chave cifrada] -oaep -out [
   Output] -inkey [Chave privada]
```

Neste a [chave cifrada] é aquela que é enviada pelo serivodr, o [Output] é o ficheiro onde será guardada a chave decifrada e a [Chave privada] é o *path* da chave privada associada à chave pública que foi enviada para o servidor no momento de depósito do ficheiro.

Finalmente, para aceder ao documento armazenado quer seja por um conjunto de utilizadores quer seja só por uma entidade, utilizamos o comando:

```
python3 vault-c.py -w decryptKey -h fileHash -f filePath
```

Com este comando o utilizador esta pedir ao programa que aceda ao documento que tenha a *hash fileHash* e que este documento seja decifrado utilizando a chave presente no ficheiro *decryptKey*. Esta chave representa a chave *docKey* só que já decifrada com a chave privada do utilizador.

No caso de o documento ser acedido por um conjunto de utilizadores é necessário que pelo menos n clientes enviem o pedido de acesso ao documento utilizando o comando abaixo para que o servidor consiga decifrar o documento. No final, o programa vai armazenar o documento retirado do cofre digital no ficheiro *filePath*, fazendo com que todos os utilizadores que pediram acesso ao documento consigam aceder a este. Por fim, caso um pedido não atinga o número de chaves mínimo em 60 segundos dá-se um *timeout* e o utilizador recebe uma mensagem de erro. Por outro lado, caso o utilizador indique um índice já existente (ou seja alguém já enviou uma chave com esse índice) o pedido é ignorado e o utilizador recebe essa indicação.

```
python3 vault-c.py -w decryptKey -i index -h fileHash -f
   filePath
```

Nota: Alternativamente à utilização dos comandos openssl indicados é possível utilizar os scripts que estão na diretoria "src/client", sendo que, o scritp "createKeys.sh" é utilizado da seguinte forma:

```
bash createKeys.sh 1 5
```

Através deste comando serão criadas 10 chaves, sendo que 5 são privadas e as outras são as públicas que foram criadas através destas privadas. As chaves privadas serão armazenadas em ficheiros denominados "privateKeyi.pem" sendo i o índice correspondente à chave e as públicas em "publicKeysi.crt".

Já script "decryptKey.sh" apenas encapsula o comando openssl utilizado para decifrar uma chave proveniente do servidor e é utilizado da seguinte forma:

```
bash decryptKey.sh [Chave cifrada] [Output] [Chave privada]
```