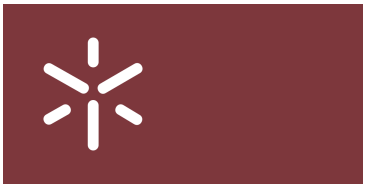

Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de Segurança de Sistemas Informáticos

Ano Letivo de 2024/2025

Cofre Seguro - Grupo 44

Afonso Pedreira
A104537

Fábio Magalhães
A104365

João Moura
A104534

Maio, 2025

SSI

Cofre Seguro - Grupo 44

Afonso Pedreira A104537

Fábio Magalhães A104365

João Moura A104534

Maio, 2025

Resumo

O projeto *Trivial Vault* consistiu no desenvolvimento de uma aplicação segura para armazenamento e partilha de ficheiros de texto, no contexto da unidade curricular de Segurança de Sistemas Informáticos. A aplicação foi concebida com o objetivo de garantir os princípios fundamentais da segurança da informação — nomeadamente, a **confidencialidade**, **integridade** e **autenticidade** dos dados — num ambiente colaborativo de utilizadores pertencentes a uma organização.

A aplicação é composta por dois componentes principais: um servidor, responsável por manter o estado global do sistema e gerir os pedidos dos utilizadores, e um cliente, que fornece uma interface em linha de comandos para interação com o serviço. A comunicação entre o cliente e o servidor é protegida através de conexões seguras, sendo utilizados certificados digitais X.509 emitidos por uma entidade certificadora confiável para autenticar utilizadores e assegurar a sua identidade.

Cada utilizador possui um cofre pessoal onde pode armazenar ficheiros, e tem ainda a possibilidade de criar grupos, gerir os seus membros e partilhar ficheiros com diferentes permissões de acesso. O sistema suporta funcionalidades como a partilha de ficheiros (com permissões de leitura ou escrita), revogação de acessos, substituição de conteúdos e gestão de ficheiros associados a grupos.

O desenvolvimento da solução implicou a implementação de mecanismos de criptografia assimétrica e simétrica para proteger os conteúdos armazenados, assegurando que nem mesmo o servidor tem acesso à informação confidencial dos utilizadores. Adicionalmente, foi adotado o formato **BSON** para a serialização eficiente das mensagens trocadas entre cliente e servidor, permitindo uma comunicação estruturada e com menor sobrecarga. O servidor incorpora ainda um **sistema de registo de eventos (logs)**, onde ficam armazenadas informações detalhadas sobre cada pedido ou tentativa de acesso, incluindo **data, hora, tipo de operação** e **identidade do utilizador** envolvido. Para reforçar os mecanismos de controlo de acessos, foi implementada uma **base de dados específica** que armazena a **lista de controlo de acessos (ACL)**, também estruturada em BSON, permitindo um acesso eficiente e seguro às permissões associadas a cada ficheiro ou grupo.

Índice

1. Introdução

Este trabalho foi desenvolvido no âmbito da unidade curricular de Segurança de Sistemas Informáticos, cujo objetivo consistiu na implementação de um sistema de gestão e partilha de ficheiros, garantindo a sua segurança através de encriptação, autenticação mútua com certificados digitais e controlo de acessos baseado em permissões.

2. Arquitetura do sistema

Antes de avançarmos com o desenvolvimento do projeto, foi essencial definir uma arquitetura coerente e alinhada com os princípios de segurança que pretendíamos assegurar. Assim, após várias discussões, o grupo chegou à conclusão de que a arquitetura apresentada a seguir oferece uma base sólida para implementar um sistema seguro:

2.1. Arquitetura Geral

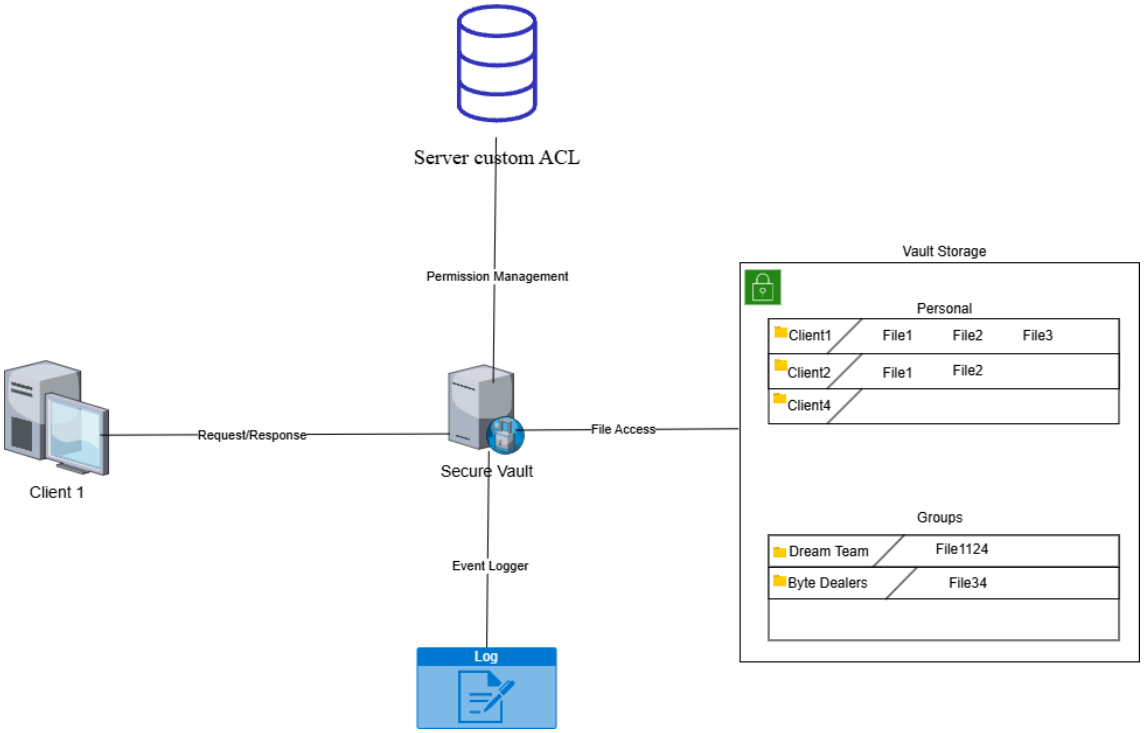

Figura 1: Arquitetura geral do Sistema

Como é possível observar na imagem anterior, a nossa arquitetura assenta num modelo cliente-servidor. A arquitetura apresentada nesta secção é geral, dado que os detalhes do processo de autenticação entre cliente e servidor serão explicados mais adiante.

O servidor, conforme descrito no enunciado, é responsável por gerir múltiplos utilizadores e, para isso, optámos por uma abordagem que inclui três componentes principais: uma base de dados associada à lista de controlo de acessos personalizada (*Server Custom ACL*), uma pasta local à qual apenas o servidor tem acesso para armazenamento de ficheiros de utilizadores e grupos (*Vault Storage*), e um sistema de registo de eventos (*Event Logger*) que permite ao administrador monitorizar toda a atividade no sistema. Este registo inclui tentativas de *login* (bem-sucedidas ou não), pedidos efetuados, bem como a data e hora de cada operação.

Após a fase inicial de autenticação — considerada a mais importante pelo grupo — o servidor assegura a existência de uma pasta dedicada ao cliente em *Vault Storage/Personal*, criando-a caso ainda não exista e, em seguida, verifica se o utilizador já está registado na *ACL* e, se necessário, adiciona uma entrada para ele. Só então é permitido ao cliente iniciar o envio de pedidos, os quais são processados pelo servidor, que consulta a *custom ACL* para verificar permissões, regista as ações no *log* de eventos e realiza as operações pretendidas pelo cliente. Assim, todos os pedidos do cliente passam obrigatoriamente pelo servidor, garantindo que cada ação é devidamente validada e processada segundo as regras de segurança estabelecidas.

2.1.1. Vault Storage

Tal como foi referido anteriormente, um dos componentes centrais do servidor é a pasta **Vault Storage**, que funciona como o repositório principal de todos os ficheiros geridos pelo sistema. Esta pasta armazena não só os ficheiros pessoais dos utilizadores, mas também os ficheiros pertencentes a grupos criados dinamicamente durante a utilização do sistema.

Esta pasta é de acesso exclusivo ao servidor e os ficheiros são referenciados internamente por identificadores únicos. Em nenhum momento os clientes têm acesso direto à estrutura interna desta pasta.

2.1.2. Custom ACL

O componente mais importante desta arquitetura é, sem dúvida, a lista de controlo de acesso personalizada, criada para gerir as permissões de cada utilizador, tanto para ficheiros pessoais, ficheiros partilhados como para ficheiros de grupo. Esta informação está armazenada num ficheiro `.bson` que representa a base de dados principal do sistema, e segue a seguinte estrutura:

```
"users": {
    "VAULT_CLI1": {
        "public_key": None,
        "personal_files": {
            "F1A3B5": {
                "name": "file_1.txt",
                "key": None
            },
            "D4E6F8": {
                "name": "file_2.txt",
                "key": None
            }
        },
        "shared_with_me": {
            "X9Y8Z7": {
                "owner": "VAULT_CLI2",
                "key": None,
                "perms": ["read"]
            }
        }
    }
},
"groups": {
    "id1": {
        "name": "grupo da CEGADA",
        "admin": "VAULT_CLI1",
        "members": {
            "VAULT_CLI1": ["read", "write"],
            "VAULT_CLI2": ["read"]
        },
        "files": {
            "F1A3B5": {
```

```
                "readers": {
                    "VAULT_CLI1": None,
                    "VAULT_CLI2": None
                }
            },
            "G7H8I9": {
                "readers": {
                    "VAULT_CLI1": None,
                    "VAULT_CLI2": None
                }
            }
        }
    }
}
```

Posteriormente, esta estrutura é serializada com `bson.dumps` e encriptada com uma chave derivada da chave privada do servidor utilizando o algoritmo HKDF com SHA-256. Esta derivação garante que apenas o servidor consegue aceder ou modificar a base de dados.

2.1.3. *Event Logger*

O registo de todas as ações (e tentativas de login) por parte dos utilizadores é essencial num servidor que lida com vários pedidos/utilizadores. Por isso, criámos um *event logger* que vai apontando tudo o que acontece: logins válidos, inválidos, pedidos de clientes, acessos suspeitos, etc.

Isto ajuda o administrador para perceber o que se está a passar no sistema — quem tentou entrar, quem conseguiu, quem não conseguiu. Também ajuda a identificar problemas ou comportamentos estranhos antes que se tornem graves. No fundo, funciona como uma espécie de caixa negra do servidor.

2.2. Arquitetura de Autenticação

Antes de o cliente poder fazer pedidos ao servidor, é necessário estabelecer uma comunicação segura, de forma a evitar possíveis ataques indesejados. Assim, o grupo considerou que esta seria a melhor estratégia a utilizar na fase de autenticação:

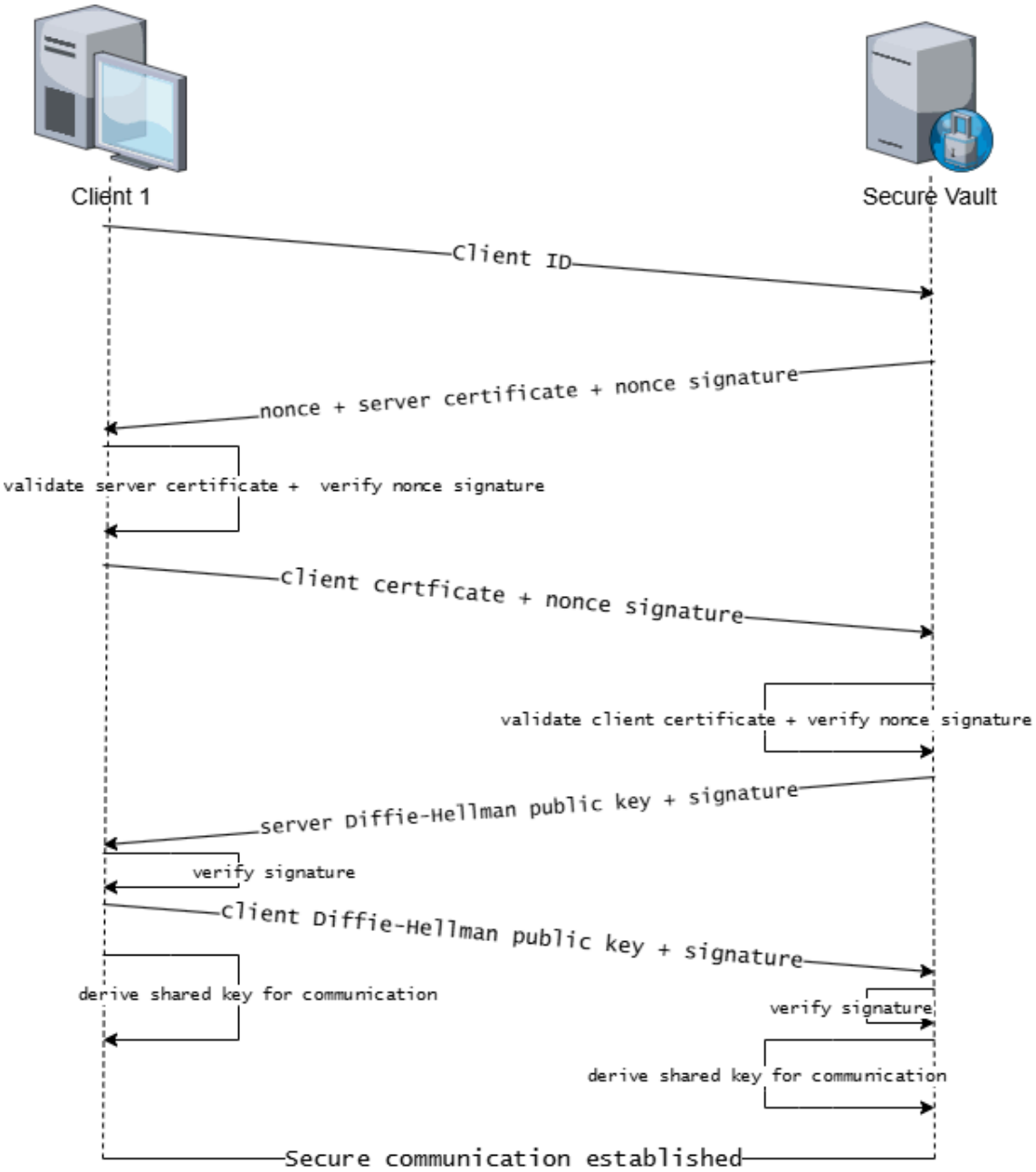

Figura 2: Arquitetura de autenticação

Sendo este o ponto crucial para garantir uma comunicação segura e conforme os princípios abordados ao longo do semestre, implementámos um mecanismo de autenticação mútua inspirado no protocolo *TLS*. Neste processo, o cliente inicia a sessão enviando apenas o seu ***ID***. O servidor, por sua vez, responde com um ***nonce*** assinado com a sua chave privada, acompanhado do seu certificado digital, assumindo assim a iniciativa de se identificar primeiro, evitando que o cliente envie dados sensíveis para uma entidade ainda não autenticada.

Depois de receber esta resposta, o cliente valida a assinatura do ***nonce*** com base no certificado do servidor, confirmando assim a sua identidade. Em seguida, assina o mesmo ***nonce*** com a sua

própria chave privada e envia de volta a assinatura juntamente com o seu certificado. O servidor valida então o certificado do cliente, garantindo que este foi emitido por uma entidade de confiança, e verifica a assinatura recebida.

A ***validação dos certificados*** é um passo essencial neste processo, pois é ela que nos permite confirmar que estamos efetivamente a comunicar com a entidade pretendida e não com alguém que se faz passar por outra. Esta etapa é fundamental para a ***autenticação mútua***, assegurando que ambos os lados da comunicação são legítimos.

Se todas as verificações forem bem-sucedidas, estabelece-se a confiança mútua e inicia-se a troca de chaves utilizando o protocolo ***Diffie-Hellman*** juntamente com a assinatura das mesmas. Isso permite gerar uma chave de sessão simétrica segura para todas as comunicações subsequentes, garantindo que todas as mensagens sejam encriptadas e que o conteúdo só possa ser visível por quem possuir a chave.

Embora este fluxo envolva várias trocas de informação, considerámos que era a abordagem mais segura e adequada para o contexto do projeto.

Este modelo garante ***confidencialidade***, ***integridade*** e ***autenticidade*** da comunicação desde o primeiro contacto, aproximando-se fortemente de um *TLS handshake*, mas com implementação manual não tão robusta.

3. Comunicação Cliente-Servidor

Após a fase de autenticação, o cliente pode enviar vários pedidos ao servidor. Assim, os comandos disponíveis são:

- `add <file-path>`
- `list [-u user-id | -g group-id]`
- `share <file-id> <user-id> <permission>`
- `delete <file-id>`
- `replace <file-id> <file-path>`
- `details <file-id>`
- `revoke <file-id> <user-id>`
- `read <file-id>`
- `group create <group name>`
- `group delete <group-id>`
- `group add-user <group-id> <user-id> <permissions>`
- `group delete-user <group-id> <user-id>`
- `group list`
- `group add <group-id> <file-path>`
- `exit`

A seguir, explicamos como o servidor lida com os pedidos recebidos.

3.1. `add`

Este comando permite que o cliente adicione um ficheiro ao seu cofre pessoal. O cliente começa por cifrar o conteúdo do ficheiro e, juntamente com este, envia a chave de cifragem cifrada com a sua chave pública. Este processo assegura que a chave seja armazenada de forma segura pelo servidor, sem a necessidade de o cliente guardá-la manualmente. A cifragem da chave com a chave pública assegura que apenas o cliente possa obter a chave de cifragem, ao decifrá-la com a sua chave privada.

Ao receber o pedido, o servidor verifica se o utilizador está registado na *ACL*. Se estiver, o ficheiro é então adicionado à pasta pessoal do cliente (`Vault/Personal/ClientX`) e o servidor atualiza a *ACL*, registando o *id* do ficheiro na secção de arquivos pessoais, bem como a chave associada ao mesmo. Este procedimento assegura que o servidor não tenha acesso ao conteúdo original do ficheiro, preservando a privacidade do utilizador.

Após armazenar corretamente todas as informações, o servidor envia ao cliente o *id* atribuído ao ficheiro que pretendia guardar no seu cofre pessoal.

3.2. `list`

Este comando permite ao cliente consultar os ficheiros aos quais tem acesso, juntamente com as respetivas permissões. Os ficheiros podem ser pessoais, partilhados por outros clientes ou perten-

centes a grupos, dependendo das opções especificadas pelo cliente (por exemplo, `-u <user-id>` para ficheiros partilhados e `-g <group-id>` para ficheiros de grupos).

Quando o servidor recebe este pedido, ele verifica as condições de acesso do cliente (se pretende ver todos os ficheiros, apenas os de um cliente específico ou de um grupo) e consulta a *ACL* para obter as informações necessárias, como permissões, nomes dos ficheiros e respetivos donos. Após reunir todos os dados, o servidor envia a resposta ao cliente, permitindo-lhe consultar as informações solicitadas.

3.3. `share`

Com este comando, o cliente pretende partilhar um ficheiro do seu cofre pessoal com outro utilizador do Cofre Seguro. Para tal, indica o *ID* do ficheiro, o *ID* do utilizador com quem deseja partilhar, bem como as permissões pretendidas (*read*,*write*). Após receber esta informação, o servidor valida a existência do ficheiro no cofre pessoal do cliente, a existência do utilizador de destino e verifica se este ainda não tem o ficheiro partilhado. Caso tudo esteja em ordem e as permissões incluam *read*, ocorre um passo adicional: o servidor envia ao cliente a chave pública do utilizador de destino, bem como a chave de cifragem do ficheiro, cifrada com a chave pública do próprio cliente. Assim, o cliente pode decifrar essa chave e reencriptá-la com a chave pública do utilizador com quem pretende partilhar o ficheiro. Por fim, essa chave é enviada de volta ao servidor, que a armazena na *ACL*, na entrada correspondente ao utilizador com quem o ficheiro será partilhado, dentro da secção destinada a ficheiros partilhados, concluindo assim o processo de partilha.

3.4. `delete`

Este comando permite ao cliente remover um ficheiro específico do sistema, dependendo do contexto em que se encontra: pessoal, partilhado ou de grupo.

- Se o ficheiro for pessoal, é removido completamente do sistema, e todos os utilizadores com quem estava partilhado perdem automaticamente o acesso.
- Se for um ficheiro partilhado, apenas a entrada referente ao acesso do cliente que executou o comando é removida da *ACL*, mantendo-se o ficheiro acessível para os restantes utilizadores autorizados.
- Caso se trate de um ficheiro de grupo, e o cliente seja administrador do grupo, o ficheiro é eliminado do sistema e todos os membros do grupo perdem o acesso.

Para isso, o cliente envia ao servidor o *ID* do ficheiro a remover. O servidor determina o tipo de ficheiro e executa as operações adequadas:
- remove o ficheiro da pasta correspondente (*personal* ou *group*);
- atualiza a *ACL*, eliminando as entradas relevantes, consoante o tipo de ficheiro e os utilizadores com acesso.

No final do processo, o servidor envia ao cliente uma resposta a indicar o sucesso ou insucesso da operação.

3.5. `replace`

Através do comando `replace`, o cliente pode substituir o conteúdo de um ficheiro já armazenado no servidor por um novo ficheiro que possui localmente. Para isso, o cliente encripta o novo conteúdo

e cifra a chave de cifragem com a sua chave pública, enviando ao servidor tanto o conteúdo como a chave cifrada.

O tratamento deste pedido depende do tipo de ficheiro:

- **Ficheiro pessoal**: O servidor atualiza diretamente o conteúdo do ficheiro e a entrada na *ACL* com a nova chave cifrada. Caso o ficheiro esteja partilhado com outros utilizadores, o servidor recolhe as suas chaves públicas e envia-as ao cliente, que deve cifrar a nova chave de conteúdo com cada uma dessas chaves públicas. As novas chaves cifradas são então armazenadas pelo servidor na *ACL*.
- **Ficheiro partilhado**: O cliente deve possuir permissões de escrita sobre o ficheiro. O servidor procede da mesma forma, recolhendo as chaves públicas de todos os utilizadores com acesso e solicitando ao cliente que cifre a nova chave de conteúdo para cada um deles.
- **Ficheiro de grupo**: O cliente deve igualmente ter permissões de escrita no grupo. O servidor identifica todos os membros com permissões de leitura, recolhe as respetivas chaves públicas e envia-as ao cliente para que possa cifrar a nova chave de conteúdo para cada um.

Após receber as chaves cifradas de volta, o servidor atualiza a *ACL* e confirma ao cliente o sucesso ou insucesso da operação.

3.6. `details`

O comando `details` permite ao cliente obter informações detalhadas sobre um ficheiro específico. Para isso, o cliente envia ao servidor o *ID* do ficheiro pretendido, e o servidor procede à verificação do tipo de ficheiro e das permissões associadas.

- *Ficheiro pessoal*: o servidor devolve todos os detalhes relevantes, como o nome do ficheiro, utilizadores com acesso e respetivas permissões.
- *Ficheiro partilhado*: o cliente recebe apenas o nome do ficheiro, o ID do proprietário e as permissões atribuídas.
- *Ficheiro de grupo*:
  - Se o cliente for o administrador do grupo, recebe a lista completa de membros com acesso e respetivas permissões.
  - Caso contrário, apenas recebe o nome do administrador do grupo e as permissões que lhe foram atribuídas.

Esta lógica assegura que apenas a informação estritamente necessária é revelada.

3.7. `revoke`

O comando `revoke` permite ao cliente remover o acesso de outro utilizador a um ficheiro do seu cofre pessoal. Para isso, o cliente envia ao servidor o *ID* do ficheiro e o *ID* do cliente cujas permissões deseja revogar.

O servidor verifica se o ficheiro pertence efetivamente ao cofre pessoal do cliente. Caso a verificação seja bem-sucedida, o servidor remove a entrada correspondente na *ACL* para que o utilizador indicado perca o acesso ao ficheiro.

Após a operação, o cliente é informado sobre o sucesso ou insucesso do processo.

3.8. `read`

O comando `read` permite que o cliente visualize o conteúdo de um ficheiro armazenado no servidor. Para isso, o cliente envia ao servidor o *ID* do ficheiro que deseja ler.

O servidor, ao receber o pedido, verifica na *ACL* se o ficheiro existe e se o cliente possui permissões de leitura sobre ele. Se as condições forem atendidas, o servidor envia de volta ao cliente o conteúdo do ficheiro, bem como a chave do ficheiro, que já está cifrada com a chave pública do cliente (não é cifrada novamente pelo servidor, pois já foi guardada dessa forma de forma a evitar que o servidor tenha acesso à chave).

O cliente, ao receber essa resposta, desencripta a chave do ficheiro utilizando a sua chave privada. Com a chave do ficheiro agora acessível, o cliente pode então proceder à desencriptação do conteúdo do ficheiro e visualizar o seu conteúdo.

3.9. `group create`

Através deste comando, o cliente pode criar um grupo no cofre do servidor. Para isso, basta indicar o nome pretendido para o grupo.

O servidor, ao receber essa informação, gera um novo *ID* para o grupo e cria uma nova pasta em `Vault/Groups` dedicada ao grupo. Além disso, a *ACL* é atualizada, adicionando o dono do grupo com permissões totais.

Após este processo, o servidor informa o cliente do *ID* atribuído ao grupo, completando a criação do mesmo.

3.10. `group delete`

Através deste comando, o cliente pode apagar um grupo, indicado pelo seu *ID*. No entanto, um grupo só pode ser eliminado pelo mesmo utilizador que o criou, ou seja, o dono do grupo.

O servidor, ao receber o pedido, faz a verificação de *ownership* . Se a verificação for bem-sucedida, o servidor procede com a remoção de todos os ficheiros pertencentes ao grupo e apaga a pasta principal do mesmo. A *ACL* também é atualizada, removendo toda a informação referente a esse grupo.

Após a conclusão do processo, o servidor informa o cliente sobre o sucesso ou insucesso da operação.

3.11. `group add-user`

O comando `group add-user` permite ao dono de um grupo adicionar novos utilizadores ao seu grupo, concedendo-lhes acesso a novos ficheiros. Para isso, o cliente deve especificar o *ID* do grupo, o *ID* do cliente a ser adicionado e as permissões que deseja atribuir a esse utilizador dentro do grupo.

Ao receber essa informação, o servidor verifica a *ownership* e caso a verificação seja bem-sucedida, o servidor atualiza a *ACL* na entrada do grupo, atribuindo as permissões definidas pelo dono para os ficheiros correspondentes.

Após completar o processo, o servidor envia ao cliente uma resposta informando o sucesso ou insucesso da operação.

3.12. `group delete-user`

Este comando permite ao cliente, caso seja administrador de um grupo, remover outro utilizador de um grupo. Para isso, o cliente deve especificar o *ID* do cliente a ser removido e o *ID* do grupo.

O servidor, ao receber a informação, verifica a ownership do grupo. Se o cliente for o dono do grupo, o servidor procede à remoção do utilizador, atualizando as permissões na *ACL* para garantir que o utilizador removido não tenha mais acesso aos ficheiros do grupo.

Após a operação, o servidor informa o cliente sobre o sucesso ou insucesso da remoção.

3.13. `group list`

Este comando permite ao cliente saber a quais grupos pertence e quais as permissões que possui em cada um deles. Para isso, o cliente envia apenas o pedido ao servidor, indicando que pretende consultar os grupos a que pertence.

O servidor, ao receber o pedido, consulta a *ACL* e reúne a informação necessária sobre os grupos associados ao cliente, incluindo o *ID* de cada grupo, o nome e as permissões atribuídas. Por fim, envia essa informação de volta ao cliente.

3.14. `group add`

Este comando permite ao cliente adicionar novos ficheiros a um grupo, desde que seja o dono do grupo ou tenha permissões de escrita. O processo segue uma lógica semelhante ao comando `add`, começando com a cifra do conteúdo do ficheiro. Juntamente com o ficheiro, o cliente envia a chave de cifragem cifrada com a sua chave pública.

O servidor, ao receber o pedido, verifica se o cliente tem permissões de escrita no grupo. Caso tenha, o servidor reúne as chaves públicas de todos os membros do grupo que possuem permissões de leitura. O cliente, então, cifra a chave do ficheiro com as chaves públicas de cada utilizador.

Depois de encriptar a chave com as respetivas chaves públicas, o cliente envia novamente a informação ao servidor. O servidor procede então ao armazenamento do ficheiro na pasta individual do grupo e à atualização da *ACL* no que toca ao grupo.

3.15. `exit`

Com este comando, o utilizador termina a comunicação com o servidor, fechando a conexão de forma segura.

4. Problemas e Desafios

Durante o desenvolvimento deste projeto, um dos principais desafios foi garantir que a comunicação fosse completamente segura. Após explorar diversas arquiteturas, conseguimos identificar uma solução que atendia às exigências do enunciado, garantindo **autenticidade, confidencialidade e integridade**.

Outro desafio significativo foi a serialização das mensagens, onde decidimos optar pelo formato *BSON*, pois ele proporcionava um processo mais simples e que no contexto deste projeto faria sentido.

Além disso, uma das dificuldades foi garantir a confidencialidade no acesso aos ficheiros, sejam eles partilhados, pessoais ou de grupos,sabendo que o servidor não podia ter acesso ao conteúdo, ou seja, ele tinha de estar cifrado.Foi necessário desenvolver uma estratégia resistente a possíveis ataques. Desta forma, a solução de encriptar a chave do ficheiro com a chave pública de quem tem acesso ao mesmo tornou-se um ponto crucial no projeto, pois permitiu atender também a esse requisito do enunciado.

5. Trabalho Futuro

Para trabalho futuro, seria interessante explorar a melhoria da eficiência da *ACL*, por exemplo, otimizando as verificações de permissões ou implementando um sistema mais escalável, que permita uma gestão de permissões mais flexível e dinâmica. Além disso, uma possível evolução seria a implementação de um sistema para gerar e gerir os próprios certificados usados pela aplicação, o que traria uma maior independência e um suporte maior de utilizadores, já que a implementação atual está limitada a apenas três utilizadores devido à utilização de três certificados .p12, o que restringe o potencial do sistema. Tal como já referido anteriormente, também faria sentido otimizar o processo de autenticação, reduzindo o número de trocas de informação necessárias e minimizando o *overhead* associado. Acreditamos que, com mais tempo de desenvolvimento, seria possível simplificar esta etapa sem comprometer a segurança, tornando o sistema mais eficiente e responsivo.

6. Conclusão

Com o desenvolvimento deste projeto, sentimos que conseguimos atingir um resultado sólido, aplicando de forma prática os conceitos abordados ao longo desta *UC*. A implementação de mecanismos de segurança, autenticação e gestão de permissões obrigou-nos a refletir bem sobre as decisões de arquitetura, o que contribuiu para uma melhor compreensão da importância destas componentes num sistema real.

No fim, consideramos que este projeto foi essencial para consolidar os conhecimentos teóricos, desafiando-nos a encontrar soluções reais para problemas concretos, o que se revelou uma experiência bastante enriquecedora.