deti
universidade de aveiro
departamento de eletrónica,
telecomunicações e informática

Relatório do Segundo Projeto

SecureShare

Segurança Informática e nas Organizações

Professores:
João Barraca (jpbarraca@ua.pt)
Alfredo Matos (alfredo.matos@ua.pt)

Trabalho Realizado Por:
Luís Simão Dias Tojal nº119636
José Pedro da Costa Bagagem nº120141

Índice

1 Introdução

O projeto **SecureShare** foi desenvolvido no âmbito da unidade curricular de Segurança Informática e nas Organizações (SIO), com o objetivo de criar uma plataforma que demonstre a aplicação prática de conceitos avançados de segurança informática. Este sistema implementa um modelo de segurança multi-nível (Multi-Level Security - MLS) baseado no modelo Bell-LaPadula, combinando controlo de acesso baseado em roles (RBAC) com encriptação end-to-end e auditoria através de hash chains.

O **SecureShare** foi concebido com os seguintes objetivos principais:

- **Encriptação de ficheiros End-to-End:** Implementar um sistema onde todos os ficheiros são encriptados no cliente antes do upload, garantindo que o servidor nunca tem acesso ao conteúdo desencriptado. Utiliza-se encriptação híbrida (AES-256-GCM e RSA-4096).
- **Controlo de Acesso Multi-Nível:** Implementar o modelo Bell-LaPadula com quatro níveis de clearance (UNCLASSIFIED, CONFIDENTIAL, SECRET, TOP_SECRET) e categorias de segurança representadas por departamentos organizacionais, garantindo:
  - *No Read Up:* Utilizadores só podem ler ficheiros com classificação igual ou inferior à sua clearance.
  - *No Write Down:* Utilizadores só podem criar ficheiros com classificação igual ou superior à sua clearance.
- **Controle de Acesso:** Implementar um sistema Role-Based Access Control com cinco roles distintos (Administrator, Security Officer, Trusted Officer, Standard User, Auditor), cada um com permissões específicas e claramente delimitadas.
- **Auditoria Tamper-Proof:** Criar um sistema de auditoria baseado em hash chains que permite verificar a integridade completa do histórico de operações, garantindo que qualquer tentativa de modificação ou eliminação de registos seja detetável.
- **Partilha Segura e Flexível:** Suportar dois modos de partilha:
  - *User-Specific:* Ficheiros partilhados com utilizadores específicos, com chaves encriptadas individualmente para cada destinatário.
  - *Public:* Links públicos com a chave de desencriptação no URL fragment, que nunca é enviado ao servidor.

2 Arquitetura do Sistema e Tecnologias Utilizadas

O **SecureShare** foi desenvolvido seguindo uma arquitetura monolítica com separação clara de responsabilidades entre cliente e servidor. O projeto está dividido em backend e apresenta uma cli para o cliente poder interagir como o sistema.

- **API RESTful:** Desenvolvida em Python com recurso ao framework FastAPI, a escolha desta tecnologia deve-se ao facto de já trabalharmos com Python ao longo das aulas e, adicionalmente, por o FastAPI simplificar significativamente o processo de criação de uma API. Este framework disponibiliza várias funcionalidades úteis, como a validação automática de dados através do Pydantic e a geração imediata de documentação dos endpoints, tornando o desenvolvimento mais eficiente e organizado.
- **Base de Dados:** Desenvolvida em SQLite, uma vez que esta tecnologia permite criar uma base de dados relacional leve, rápida e de fácil integração. Para o ORM, optámos pelo SQLModel, que não só simplifica a criação de tabelas e a execução de queries, como também implementa medidas de segurança importantes, tais como a proteção contra SQL injection através de queries parametrizadas.
- **Autenticação:** Utilizámos a biblioteca PyJWT, que permite a utilização de JWT (JSON Web Tokens) e facilita o processo de verificação de login. Embora o sistema não seja totalmente stateless, é feita uma verificação para garantir que o token não foi previamente revogado.
- **CLI:** Utilizámos Python, por ser uma linguagem simples de utilizar neste processo.

- **Deployment:** O backend e a base de dados estão empacotados em containers Docker distintos. A comunicação entre cliente e servidor é realizada através de HTTPS, garantindo a confidencialidade e integridade dos dados em trânsito. Isto previne ataques do tipo man-in-the-middle e eavesdropping. Para o ambiente de desenvolvimento, foram utilizados certificados autoassinados, permitindo simular o ambiente de produção com TLS sem necessidade de uma autoridade de certificação real. Para implementar o HTTPS, optámos por configurar o Nginx como reverse proxy, terminando o TLS antes do FastAPI.

3 Autenticação

O processo de autenticação é subdividido em 3 partes: Criação, Ativação e Login.

- **Criação:** Apenas o administrador terá permissão para criar contas, assegurando que não sejam registadas contas por indivíduos externos à organização e permitindo a identificação precisa do titular de cada conta. O administrador terá de definir o nome de utilizador, facilitando a associação inequívoca da conta à respetiva pessoa. O sistema, por sua vez, gera automaticamente uma palavra-passe de uso único (one-time password), de forma aleatória, evitando que o administrador tenha de criar manualmente a palavra-passe e prevenindo a utilização de palavras-passe facilmente memorizáveis.
- **Ativação:** Seguidamente, o titular da conta recém-criada deverá inserir o seu nome e a palavra-passe de uso único (one-time password). Será também necessário definir uma nova palavra-passe, que passará a ser utilizada no login. O sistema exige que esta palavra-passe tenha mais de 8 caracteres, incluindo pelo menos uma letra maiúscula, uma letra minúscula e um dígito, garantindo assim que não sejam utilizadas palavras-passe previsíveis. No CLI, caso a ativação seja concluída com sucesso, serão gerados os pares de chaves: a chave pública será armazenada no servidor, sem qualquer encriptação, e a chave privada será guardada em formato binário, após ter sido derivada a partir da palavra-passe.
- **Login:** O utilizador envia o seu nome e palavra-passe e, após a validação da sua identidade, são devolvidas as suas informações, incluindo o identificador (ID) e todos os JWTs associados, como o token de sessão, os Role Tokens e os Clearance Tokens. A partir do login, o CLI envia os tokens necessários para cada endpoint, sendo o servidor responsável por verificar a assinatura e a data de expiração de cada um.

3.1 Endpoints da API

Os endpoints ligados a autenticação são:

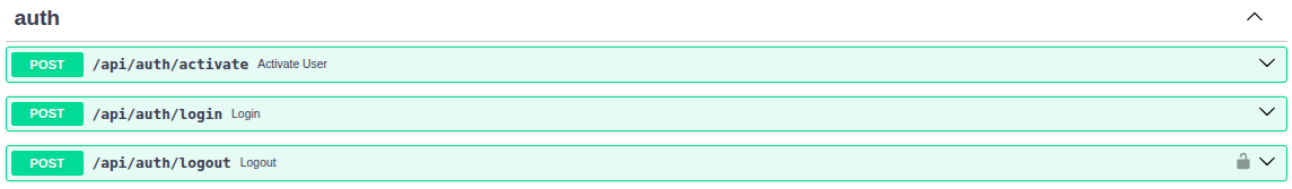

Figura 1: Endpoints da API para autenticação.

3.2 Token de sessão

Optámos por utilizar JSON Web Tokens (JWT). Assim, quando o utilizador faz login, o servidor gera um token contendo os seus dados e assina-o com a sua chave privada através do algoritmo RSA-256. A partir desse momento, sempre que for feito um pedido a um endpoint que exija autenticação, o token é enviado e o servidor valida apenas a assinatura utilizando a respetiva chave pública.

Estes tokens têm um tempo de vida de 15 minutos. Após esse período, o utilizador terá de iniciar sessão novamente para continuar a utilizar o sistema. Esta tempo de vida é essencial, pois, caso o token seja comprometido, o atacante terá apenas uma janela de tempo muito reduzida para causar danos na conta do utilizador.

Ao fazer logout, decidimos que o sistema irá guardar na base de dados o token que foi revogado. Desta forma, garantimos que, mesmo que alguém consiga obter esse token após o utilizador terminar a sessão, não conseguirá aceder aos ficheiros aos quais o utilizador tinha acesso.

3.3 Encriptação de palavras-passe

Durante a ativação de uma conta, o utilizador envia a sua palavra-passe ao servidor através de HTTPS. Assim que a recebe, o servidor procede de imediato à sua encriptação, garantindo que, mesmo que alguém consiga aceder aos dados, não conseguirá recuperar as palavras-passe.

O algoritmo utilizado para este processo é o Argon2id, escolhido por ser atualmente um dos mais robustos. Para além de exigir uma grande quantidade de memória, também limita a utilização de GPU, o que reduz significativamente a eficácia de ataques de brute-force. Mesmo que um atacante obtivesse todos os hashes, o tempo necessário para descobrir uma palavra-passe seria extremamente elevado. Para além disso, também evita ataques de Rainbow tables, uma vez que o algoritmo utiliza sempre salts únicos, impedindo a reutilização de tabelas pré-calculadas.

3.4 Chaves Assimétricas

Neste projeto, as chaves geradas são sempre do tipo RSA, uma vez que este padrão é amplamente conhecido, testado e considerado seguro. Para evitar que o utilizador perca o acesso à sua conta caso extravie a sua chave privada, é armazenado no servidor um blob binário cifrado, derivado diretamente da palavra-passe do utilizador.

O processo funciona da seguinte forma: primeiro, é gerada uma chave de criptografia AES a partir da palavra-passe, utilizando o algoritmo Argon2id com um salt aleatório. Este procedimento garante que a chave resultante é suficientemente longa, robusta e imprevisível, algo que uma palavra-passe normal não assegura. Em seguida, a chave privada RSA (no formato PEM) é cifrada com AES em modo GCM, utilizando um nonce também gerado aleatoriamente. O modo GCM fornece tanto confidencialidade como integridade dos dados, produzindo igualmente um tag de autenticação.

Por fim, o salt, o nonce, o ciphertext e o tag são concatenados para formar um único blob binário, que é posteriormente codificado em Base64 e armazenado no servidor. Assim, apenas o utilizador que conheça a sua palavra-passe poderá, mais tarde, derivar novamente a chave AES e recuperar a sua chave privada a partir do blob guardado.

3.5 Logging de Operações

Todas as operações sobre departamentos são registadas no log de auditoria:

- `CREATE_USER` – Criação da conta
- `ACTIVATE_USER` – Ativação da conta
- `LOGIN_USER` – Log In numa conta
- `UPDATE_VAULT` – Atualiza a blob com da chave privada
- `UPDATE_USER_INFO` – Mudanças nos dados da conta

4 Controlo de Acesso Baseado em Roles (RBAC)

Implementou-se um sistema de Controlo de Acesso Baseado em Papéis (RBAC - Role-Based Access Control) para definir as permissões administrativas e de gestão no sistema. Considerou-se que um utilizador poderá ter mais do que um role.

O sistema define cinco papéis principais, cada um com um conjunto específico de permissões:

Role	**Descrição e Responsabilidades Principais**
ADMINISTRATOR	Gere a aplicação, nomeia SECURITY_OFFICERS e cria departamentos.
SECURITY_OFFICER	Define políticas de segurança, emite e revoga (*clearance tokens*).
TRUSTED_OFFICER	Papel elevado que pode **contornar as regras do Bell-LaPadula** para leitura/escrita, mediante justificação e registo em *log*, sendo que este cargo está associado a um ou mais departamentos, só podendo contornar as regras se todos os departamentos do ficheiro fazerem parte da lista de departamentos onde este é TRUSTED_OFFICER.
STANDARD_USER	Papel padrão. Pode carregar e descarregar ficheiros, obedecendo rigorosamente às políticas de segurança.
AUDITOR	Acede e valida o registo de interações (*audit log*), que é assegurado por uma cadeia de *hashes*.

Table 1: Papéis (Roles) do RBAC no Projeto SecureShare

4.1 Verificação de Roles

Optou-se por realizar a verificação de roles através de tokens específicos de autorização, distintos dos tokens de sessão. Estes também são JWTs mas apresentam um tempo de vida bastante superior aos tokens de sessão.

Estes tokens são criados por um ADMINISTRATOR ou por um SECURITY_OFFICER e são assinados com a sua respetiva chave privada.

Desta forma, mesmo que alguém obtenha acesso indevido à conta de um utilizador com estas permissões, não conseguirá atribuir novos roles a outras contas sem possuir igualmente a chave privada necessária para assinar os tokens.

No lado do cliente, o CLI seleciona automaticamente o role mais adequado para cada operação. Quando o token chega ao servidor, este lê no header o ID do utilizador que o assinou, obtém a sua chave pública e verifica a assinatura, garantindo assim a autenticidade e a integridade do token.

4.2 Revogação de Roles

A revogação de um role também é suportada para pessoas com o papel de SECURITY_OFFICER. Quando um role é revogado, é criado um novo registo contendo o ID do respetivo token. Nas interações seguintes, o servidor consulta a base de dados e verifica que o token em causa foi revogado, impedindo assim a sua utilização.

4.3 Logging de Operações

Todas as operações sobre departamentos são registadas no log de auditoria:

- `REVOKE_ROLE` – Token que foi revocado
- `ADD_ROLE` – Role adicionado

5 Segurança Multi-Nível (MLS)

Implementou-se um modelo de segurança multi-nível baseado no modelo Bell-LaPadula, que combina níveis hierárquicos de classificação com categorias não-hierárquicas (departamentos) para controlar a confidencialidade da informação.

5.1 Níveis de Classificação

O sistema define quatro níveis de *clearance* hierárquicos, ordenados do menor para o maior nível de acesso:

1. **UNCLASSIFIED**
2. **CONFIDENTIAL**
3. **SECRET**
4. **TOP_SECRET**

Cada ficheiro carregado no sistema deve ser associado a um nível de classificação. Da mesma forma, cada utilizador pode possuir *clearance tokens* assinados criptograficamente que define o seu nível de acesso.

5.2 Departamentos como Categorias de Segurança

Para além dos níveis hierárquicos, o sistema utiliza departamentos como *labels* de segurança não-hierárquicos. Um ficheiro pode estar associado a um ou mais departamentos (por exemplo, FINANCE, HR, ENGINEERING), e um utilizador só pode aceder a ficheiros que pertençam a departamentos para os quais possui autorização.

A associação entre ficheiros e departamentos é gerida através da tabela `FileDepartment`, enquanto que as autorizações dos utilizadores são representadas pela tabela `ClearanceDepartment`, que associa os *clearance tokens* aos respetivos departamentos.

5.3 Modelo Bell-LaPadula: Regras de Acesso

O sistema implementa as duas propriedades fundamentais do modelo Bell-LaPadula:

5.3.1 Simple Security Property (No Read Up)

Um utilizador só pode ler um ficheiro se:

- O seu nível de *clearance* for **maior ou igual** ao nível de classificação do ficheiro
- Os departamentos do ficheiro forem um **subconjunto** dos departamentos autorizados do utilizador

$$\text{Level}(\text{Subject}) \geq \text{Level}(\text{Object}) \land \text{Departments}(\text{Subject}) \supseteq \text{Departments}(\text{Object}) \tag{1}$$

Na implementação, durante o download de um ficheiro, o sistema verifica para cada departamento associado ao ficheiro se o utilizador possui um *clearance token* ativo e válido com nível de acesso suficiente. Caso contrário, é registada uma violação MLS no log de auditoria.

5.3.2 No Write Down

Um utilizador só pode carregar um ficheiro se:

- O seu nível de *clearance* for **menor ou igual** ao nível de classificação atribuído ao ficheiro
- Os departamentos autorizados do utilizador forem um **subconjunto** dos departamentos do ficheiro

$$\text{Level}(\text{Subject}) \leq \text{Level}(\text{Object}) \land \text{Departments}(\text{Subject}) \subseteq \text{Departments}(\text{Object}) \tag{2}$$

Esta regra impede que utilizadores com alto nível de acesso escrevam informação classificada em ficheiros de baixa classificação, o que poderia resultar em fugas de informação.

5.4 Clearance Tokens

Os *clearance tokens* são objetos assinados criptograficamente emitidos pelo *Security Officer*. Cada token contém:

- Identificador do utilizador
- Nível de *clearance*
- Data de emissão e expiração
- Estado (ACTIVE, REVOKED, EXPIRED)
- Assinatura digital do emissor
- Lista de departamentos associados

Os tokens têm um tempo de vida limitado e podem ser revogados pelo *Security Officer* através da criação de objetos de revogação armazenados no servidor.

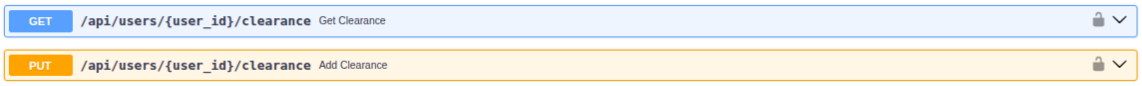

Figura 2: Endpoints da API para atualizar as clearances.

5.5 Trusted Officer Bypass

O sistema prevê uma exceção controlada às regras Bell-LaPadula para utilizadores com o papel de **TRUSTED_OFFICER**. Estes utilizadores podem ultrapassar as restrições de leitura e escrita desde que forneçam uma justificação válida.Optámos ainda por implementar a relação do **TRUSTED_OFFICER** com os departamentos, de forma que o bypass só possa ser efetuado se o utilizador tiver o papel de Trusted Officer em ambos os departamentos envolvidos.

Quando um *Trusted Officer* realiza uma operação de bypass:

- Deve obrigatoriamente fornecer uma *reason* explicando a necessidade do bypass
- A operação é registada no log de auditoria com a ação `SECURITY_OFFICER_BYPASS_UPLOAD`
- A justificação é armazenada juntamente com o ficheiro para futura auditoria

Esta funcionalidade permite flexibilidade operacional em situações excepcionais, mantendo a rastreabilidade e responsabilização através do sistema de logging.

5.6 Logging de Violações MLS

Todas as tentativas de acesso que violem as políticas MLS são registadas no log de auditoria com a ação `MLS_VIOLATION`. O log inclui:

- Identificação do utilizador
- Ficheiro a que tentou aceder
- Motivo da negação (clearance insuficiente, departamento não autorizado, etc.)
- *Timestamp* da tentativa

Este mecanismo permite que os auditores identifiquem tentativas de acesso não autorizado e potenciais violações de segurança.

6 Sistema de Transferência de Ficheiros

O sistema SecureShare implementa um mecanismo robusto de transferência de ficheiros com encriptação end-to-end, garantindo que o servidor nunca tem acesso ao conteúdo em texto limpo.

6.1 Encriptação Híbrida

O sistema utiliza um esquema de **encriptação híbrida** que combina criptografia simétrica e assimétrica para otimizar segurança e desempenho:

6.1.1 Chave Simétrica (AES)

Para cada ficheiro carregado, o cliente gera uma **chave AES aleatória** (File Key) que é utilizada para encriptar o conteúdo do ficheiro. A encriptação simétrica é ideal para ficheiros grandes devido à sua eficiência computacional.

O ficheiro encriptado é acompanhado pelos seguintes parâmetros criptográficos:

- `aes_iv` – Vector de inicialização (IV) único para cada encriptação
- `aes_tag` – Tag de autenticação para verificação de integridade (modo GCM)

6.1.2 Chave Assimétrica (RSA/ECC)

A chave AES é protegida através de **encriptação assimétrica**. Para ficheiros privados, o cliente encripta a chave AES com a sua própria chave pública antes de a enviar para o servidor. Desta forma:

- O servidor armazena apenas a chave AES encriptada (`symetric_key_encrypted`)
- Apenas o proprietário, com a sua chave privada, pode desencriptar a chave AES
- O conteúdo do ficheiro permanece inacessível ao servidor

6.2 Upload de Ficheiros

O processo de upload segue o fluxo ilustrado nos endpoints da API:

1. O cliente gera uma chave AES aleatória
2. O cliente encripta o ficheiro localmente com a chave AES (modo GCM para integridade)
3. Para ficheiros privados, o cliente encripta a chave AES com a sua chave pública
4. O cliente envia para `POST /api/transfers/` o ficheiro encriptado, IV, tag, chave encriptada, nível de classificação e lista de departamentos
5. O servidor valida as permissões MLS e armazena o ficheiro

O servidor atribui automaticamente um **UID único** (UUID v4) a cada ficheiro, garantindo identificadores não previsíveis para os URLs de download.

6.3 Controlo de Acesso no Upload (No Write Down)

Conforme descrito na secção MLS, o upload está sujeito à propriedade do modelo Bell-LaPadula. Um utilizador só pode carregar ficheiros com nível de classificação **igual ou superior** ao seu próprio nível de *clearance* para os departamentos selecionados.

Para situações excepcionais, utilizadores com o papel de **Trusted Officer** podem ultrapassar estas restrições mediante justificação obrigatória, que é registada no log de auditoria.

6.4 Partilha Pública (Organizacional)

O sistema suporta dois modos de partilha: **privada** e **pública**. É importante clarificar que o termo "público" não significa acessível a qualquer pessoa na Internet, mas sim a qualquer utilizador **dentro da organização** que possua as permissões MLS adequadas.

6.4.1 Diferença entre Partilha Privada e Pública

- **Partilha Privada** (`is_private = true`): O proprietário deve especificar explicitamente cada utilizador com quem pretende partilhar o ficheiro. A chave AES é encriptada individualmente com a chave pública de cada destinatário.

- **Partilha Pública** (`is_private = false`): O proprietário não necessita de definir destinatários específicos. Qualquer utilizador autenticado da organização com *clearance* e departamentos adequados pode aceder ao ficheiro.

6.4.2 Verificação MLS em Partilhas Públicas

Mesmo em ficheiros públicos, o sistema **continua a aplicar todas as verificações MLS**. Antes de permitir o download, o servidor valida:

- Se o utilizador possui um *clearance token* ativo com nível igual ou superior à classificação do ficheiro
- Se o utilizador está autorizado em todos os departamentos associados ao ficheiro

Assim, um ficheiro "público" com classificação `SECRET` no departamento FINANCE só será acessível a utilizadores com *clearance* `SECRET` ou superior e autorização para o departamento FINANCE.

6.4.3 Gestão de Chaves em Partilhas Públicas

Para ficheiros públicos, a chave AES pode ser transmitida de duas formas:

1. **No fragmento da URL**: A chave de desencriptação é incluída no fragmento da URL (após o símbolo `#`), por exemplo: `/download/{uid}#<chave_base64>`. O fragmento da URL nunca é enviado ao servidor, mantendo a chave inacessível ao backend.
2. **Sem encriptação da chave**: Como não há destinatário específico, a chave AES não é encriptada com nenhuma chave pública. O campo `symetric_key_encrypted` pode permanecer nulo, e a proteção do ficheiro depende exclusivamente do controlo de acesso MLS do servidor.

6.4.4 Vantagens da Partilha Pública

- **Simplicidade**: Não é necessário conhecer antecipadamente todos os destinatários
- **Flexibilidade**: Novos utilizadores com permissões adequadas podem aceder automaticamente
- **Segurança mantida**: As políticas MLS garantem que apenas utilizadores autorizados acedem ao conteúdo

Este modelo é ideal para partilhar documentos com toda uma equipa ou departamento, sem necessidade de gerir individualmente cada acesso.

6.4.5 Justificação da Abordagem

Esta implementação foi escolhida em detrimento de uma partilha verdadeiramente aberta a toda a organização (sem verificações MLS) por razões de segurança fundamentais. Numa abordagem completamente aberta, todos os ficheiros públicos teriam obrigatoriamente de possuir o nível de classificação mais baixo (`UNCLASSIFIED`), uma vez que qualquer utilizador poderia aceder. Isto criaria dois problemas críticos: primeiro, devido à propriedade *No Write Down*, apenas utilizadores com *clearance* `UNCLASSIFIED` conseguiriam criar ficheiros públicos, limitando severamente a utilidade da funcionalidade; segundo, se ignorássemos a regra *No Write Down* para permitir partilhas públicas, estaríamos a conceder um bypass generalizado que permitiria a qualquer utilizador mal-intencionado classificar informação secreta como `UNCLASSIFIED`, garantindo acesso universal à mesma. Ao manter as verificações MLS ativas em partilhas públicas, evitamos ambos os problemas e garantimos que a classificação do ficheiro continua a ser respeitada.

6.5 Download de Ficheiros

O download através de `GET /api/download/{file_uid}` inclui verificações MLS (No Read Up) e retorna:

- O ficheiro encriptado (blob binário)
- Headers HTTP com a chave encriptada, IV e tag para desencriptação no cliente

O cliente deve então desencriptar a chave AES com a sua chave privada e usar essa chave para desencriptar o conteúdo do ficheiro.

6.6 Partilha de Ficheiros

O sistema permite partilhar ficheiros privados com outros utilizadores através do endpoint `POST /api/transfers/share/`. O processo de partilha funciona da seguinte forma:

1. O proprietário desencripta a chave AES do ficheiro usando a sua chave privada
2. O proprietário obtém a chave pública do destinatário através da API
3. O proprietário re-encripta a chave AES com a chave pública do destinatário
4. A nova chave encriptada é enviada ao servidor juntamente com o `file_uid` e `user_share_id`

As chaves partilhadas são armazenadas na tabela `EncryptedFileKeys`, que associa cada ficheiro a múltiplos destinatários, cada um com a sua própria versão da chave AES encriptada.

Este modelo permite flexibilidade na gestão de acessos sem necessidade de re-encriptar o ficheiro original.

6.7 Endpoints da API

Nesta figura estão representados todos os endpoints relacionados com a transferência de ficheiros.

Figura 3: Endpoints da API para transferência de ficheiros.

6.8 Expiração Automática

Todos os ficheiros têm uma **data de expiração** configurável (por defeito, 3 dias após o upload). Após esta data, os ficheiros são automaticamente eliminados do sistema, garantindo que dados sensíveis não permanecem armazenados indefinidamente.

7 Sistema de Auditoria (Audit Logs)

O sistema SecureShare implementa um mecanismo robusto de auditoria baseado em *hash chain*, garantindo a integridade e imutabilidade dos registos de todas as operações realizadas na plataforma.

7.1 Estrutura dos Logs

Cada entrada de log contém os seguintes campos:

- `action` – Tipo de ação realizada (e.g., `UPLOAD`, `DOWNLOAD`, `MLS_VIOLATION`)
- `time_stamp` – Data e hora da operação em UTC
- `description` – Descrição detalhada da operação
- `user_id` – Identificador do utilizador que realizou a ação
- `previous_hash` – Hash da entrada anterior na cadeia
- `current_hash` – Hash da entrada atual
- `check_by` – Identificador do auditor que validou a entrada (opcional)
- `check_at` – Data da validação (opcional)

- `signature` – Assinatura digital do auditor (opcional)

7.2 Hash Chain

Para garantir a integridade dos logs, cada entrada é ligada à anterior através de uma *hash chain*. O hash de cada entrada é calculado da seguinte forma:

```
hash_input = action | timestamp | description | user_id | previous_hash
current_hash = SHA256(hash_input)
```

Esta estrutura garante que qualquer alteração a uma entrada anterior invalida todos os hashes subsequentes, tornando impossível modificar o histórico sem deteção.

7.3 Controlo de Acesso

O acesso aos logs de auditoria é **exclusivo a utilizadores com o papel de AUDITOR**. Antes de qualquer operação sobre os logs, o sistema verifica se o utilizador possui um `RoleToken` ativo com a role `AUDITOR`. Tentativas de acesso não autorizado são registadas no próprio log com a ação `CHECK_AUDITOR_FAIL`.

7.4 Verificação da Integridade

Quando um auditor acede aos logs através de `GET /api/audit/log`, o sistema:

1. Recupera todos os registos ordenados cronologicamente
2. Percorre a cadeia verificando que cada `previous_hash` corresponde ao `current_hash` da entrada anterior
3. Recalcula o hash de cada entrada para confirmar que não foi alterada
4. Retorna a lista de logs e, separadamente, a lista de logs que foram detetados como alterados (`tampered`)

7.5 Assinatura de Logs pelo Auditor

O auditor pode assinar uma entrada de log através de `PUT /api/audit/validate`, criando um ponto de verificação na cadeia. O processo funciona da seguinte forma:

1. O auditor assina o `current_hash` da entrada com a sua **chave privada**
2. Submete a assinatura (codificada em Base64) juntamente com o `log_id`
3. O servidor recupera a **chave pública** do auditor armazenada no sistema
4. O servidor verifica a assinatura utilizando o algoritmo RSA-PSS com SHA-256
5. Antes de aceitar a assinatura, o servidor valida a integridade de toda a cadeia de logs até à entrada em questão
6. Se tudo estiver válido, a assinatura, o identificador do auditor e o *timestamp* são armazenados na entrada

7.6 Verificação de Assinaturas

As assinaturas dos auditores permitem verificar posteriormente que:

- O log foi validado por um auditor específico numa determinada data
- A cadeia de logs até esse ponto estava íntegra no momento da assinatura
- A entrada não foi alterada desde a assinatura (qualquer modificação invalidaria a assinatura)

Este mecanismo cria pontos de confiança na cadeia, permitindo auditorias incrementais sem necessidade de verificar toda a história desde o início.

7.7 Ações Registadas

O sistema regista automaticamente todas as operações relevantes, incluindo:

- Uploads e downloads de ficheiros (`UPLOAD`, `DOWNLOAD_SUCCESS`)
- Violações de políticas MLS (`MLS_VIOLATION`)
- Tentativas de acesso negadas (`UPLOAD_DENIED`, `DOWNLOAD_FAILED`)
- Operações de bypass por Trusted Officers (`SECURITY_OFFICER_BYPASS_UPLOAD`)
- Partilhas de ficheiros (`FILE_SHARED`)
- Eliminação de ficheiros (`FILE_DELETED`)
- Acessos aos logs de auditoria (`RETRIEVE_LOGS_SUCCESS`)
- Deteção de logs alterados (`RETRIEVE_LOGS_WERE_ALTERED`)

8 Gestão de Departamentos

Os departamentos constituem um elemento central do modelo de segurança do SecureShare, funcionando como *labels* de segurança não-hierárquicos que complementam os níveis de classificação no controlo de acesso.

8.1 Definição e Propósito

Um departamento representa uma unidade organizacional (e.g., FINANCE, HR, ENGINEERING) que agrupa utilizadores e ficheiros com base na sua área funcional. No contexto do modelo MLS, os departamentos atuam como categorias de segurança que restringem o acesso lateral entre diferentes áreas da organização.

8.2 Estrutura do Departamento

Cada departamento contém os seguintes atributos:

- `id` – Identificador único do departamento
- `name` – Nome do departamento (único no sistema)
- `created_at` – Data e hora de criação
- `created_by` – Identificador do administrador que criou o departamento

8.3 Controlo de Acesso

A gestão de departamentos é **exclusiva a utilizadores com a role ADMINISTRATOR**. Antes de qualquer operação de criação, listagem ou eliminação, o sistema verifica se o utilizador possui um `RoleToken` ativo com a role `ADMINISTRATOR`. Tentativas de acesso não autorizado são registadas no log de auditoria com a ação `CHECK_ADMIN_FAIL`.

8.4 Associações no Sistema

Os departamentos estão associados a duas entidades principais no sistema:

8.4.1 Associação com Clearance Tokens

Através da tabela `ClearanceDepartment`, os *clearance tokens* são associados aos departamentos para os quais concedem acesso. Esta relação muitos-para-muitos permite que um único token autorize acesso a múltiplos departamentos.

8.4.2 Associação com Ficheiros

Através da tabela `FileDepartment`, cada ficheiro pode estar associado a um ou mais departamentos. Esta associação determina quais utilizadores (com base nos seus *clearance tokens*) podem aceder ao ficheiro, conforme as regras do modelo Bell-LaPadula.

8.5 Endpoints da API

A gestão de departamentos é realizada através dos seguintes endpoints:

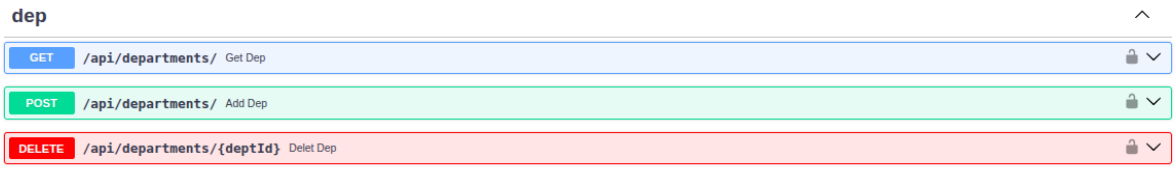

Figura 4: Endpoints da API para gestão de departamentos.

8.6 Validações

O sistema implementa as seguintes validações na gestão de departamentos:

- **Unicidade do nome**: Não é permitido criar departamentos com nomes duplicados. Tentativas de duplicação são registadas com a ação `DEPARTMENT_DUPLICATE`.
- **Existência na eliminação**: A eliminação de um departamento inexistente é registada com a ação `DEPARTMENT_UNKNOWN`.

8.7 Logging de Operações

Todas as operações sobre departamentos são registadas no log de auditoria:

- `DEPARTMENTS_ACCESSED` – Listagem de departamentos
- `DEPARTMENT_CREATED` – Criação de um novo departamento
- `DEPARTMENT_DELETED` – Eliminação de um departamento
- `DEPARTMENT_DUPLICATE` – Tentativa de criar departamento duplicado
- `DEPARTMENT_UNKNOWN` – Tentativa de eliminar departamento inexistente

9 Conclusões

9.1 Objetivos Alcançados

O desenvolvimento do **SecureShare** permitiu concretizar com sucesso os principais objetivos definidos no enunciado do projeto:

- **Encriptação End-to-End**: Implementação completa de um sistema híbrido de encriptação (AES-256-GCM + RSA) que garante que o servidor nunca tem acesso ao conteúdo dos ficheiros em texto limpo. Toda a encriptação e desencriptação ocorre exclusivamente no lado do cliente.
- **Modelo Bell-LaPadula**: Aplicação funcional das propriedades *No Read Up* e *No Write Down*, com os quatro níveis de classificação (UNCLASSIFIED, CONFIDENTIAL, SECRET, TOP_SECRET) e integração com departamentos como *labels* de segurança.
- **Sistema RBAC**: Hierarquia completa de roles (Administrator, Security Officer, Trusted Officer, User, Auditor) com tokens assinados digitalmente e mecanismo de revogação.
- **Auditoria com Hash Chain**: Sistema de logging imutável que permite detetar qualquer tentativa de adulteração do histórico, com assinatura criptográfica por auditores.

- **Gestão Segura de Credenciais**: Utilização de Argon2id para hash de passwords, armazenamento seguro de chaves privadas encriptadas (vault), e sistema de one-time passwords para ativação de contas.

9.2 Limitações Conhecidas

O sistema atual apresenta algumas limitações que devem ser consideradas:

- **Escalabilidade**: A utilização de SQLite limita a capacidade de processamento concorrente. Para ambientes de produção com elevado volume de utilizadores, seria necessário migrar para PostgreSQL ou similar.
- **Certificados Self-Signed**: Em ambiente de desenvolvimento são utilizados certificados auto-assinados. Uma implementação de produção requer certificados emitidos por uma CA reconhecida.
- **Recuperação de Palavra-Passe**: Não existe qualquer mecanismo de recuperação caso um utilizador se esqueça da sua palavra-passe; se isso acontecer, perderá permanentemente todo o acesso de que dispunha.
- **Conflitos com o Modelo Bell-LaPadula com o uso de vários clearance tokens**: Como um utilizador pode possuir mais do que um clearance token, isso permite-lhe aceder a ficheiros com um nível de acesso elevado e, posteriormente, publicá-los recorrendo a outro clearance token com um nível de acesso inferior ou pertencente a outro departamento. Esta situação abre a possibilidade de ocorrerem write-downs, permitindo que potenciais whistleblowers divulguem informação sensível ou classificada. Optou-se por manter este comportamento na implementação final, uma vez que foi interpretado como estando de acordo com o solicitado no guião, devendo constituir um ponto de atenção para o SECURITY OFFICER aquando da atribuição das clearances.
- **Interface de Utilizador**: A CLI, embora funcional, poderia ser complementada com uma interface web para maior usabilidade.

9.3 Reflexão Final

O projeto **SecureShare** demonstrou ser um exercício valioso na aplicação prática de conceitos teóricos de segurança informática. A implementação de um sistema real que combina encriptação end-to-end, controlo de acesso multi-nível e auditoria criptográfica permitiu compreender profundamente os desafios e trade-offs envolvidos no desenvolvimento de software seguro.

A experiência adquirida reforça a importância de uma abordagem *security-by-design*, onde os mecanismos de segurança são considerados desde a fase inicial de arquitetura, em vez de serem adicionados posteriormente como camadas superficiais.

O resultado final é um sistema funcional que, apesar das limitações, demonstra a viabilidade de implementar controlos de segurança robustos numa aplicação de partilha de ficheiros, garantindo confidencialidade e integridade.

10 Referências

Recurso:	URL
Documentação sqlmodel	https://sqlmodel.tiangolo.com/
Documentação fastApi	https://fastapi.tiangolo.com/
Bell lapadula	https://www.vokke.com.au/permission-systems-and-access-controls-the-bell-lapadula-model/
cryptography	https://cryptography.io/en/latest/
hashing password	https://www.geeksforgeeks.org/python/how-to-hash-passwords-in-python/
argon2	https://en.wikipedia.org/wiki/Argon2
website da disciplina	https://sweet.ua.pt/jpbarraca/course/sio/