Searchable Secure Block Storage com OAS/OAMS

Relatório Técnico Resumido

HENRIQUE MONTEIRO (72928)
MADALENA FORTUNATO ALVES (75150)

Dezembro 2025

1 Introdução

Este projeto implementa um sistema de *Searchable Secure Block Storage* com três componentes principais:

- **OBSS** – servidor de armazenamento de blocos cifrados;
- **OAS** – *Authorization Server* responsável por registo, autenticação e emissão de tokens JWT;
- **OAMS** – serviço de gestão de partilhas e decisões de acesso (*CheckAccess*).

O cliente comunica com estes três serviços, suporta múltiplos utilizadores (secções) e implementa:

- cifragem de blocos com AES-GCM;
- pesquisa por palavras-chave via *search tokens* determinísticos (HMAC);
- registo e *login* com derivação de password através de PBKDF2;
- modelo de partilhas em que o OAMS decide, para cada `file_id`, se o acesso é permitido.

Todo o sistema usa um protocolo simples de **JSON por linha** sobre TCP, com funções partilhadas em `common/protocol.py`.

2 Arquitetura e Componentes

2.1 OBSS (Encrypted Block Storage)

O OBSS está em `server/server.py` e executa em `127.0.0.1:5500`. Responsabilidades:

- armazenar blocos cifrados em `server/storage/blocks/`;
- manter metadados cifrados em `server/storage/metadata/`:
  - `index.json`: `search_token` → lista de `file_id`;
  - `filemap.json`: `file_id` → lista de `block_id`.
- expor operações: `PUT`, `STORE_BLOCK`, `STORE_TOKEN`, `SEARCH`, `GET` e `LIST_BLOCKS`.

Os metadados são cifrados com AES-GCM usando uma **meta key** em `server/keys/key.txt`. A leitura/escrita é protegida com `RLock` e *atomic write* (`.tmp` + `os.replace`) para suportar concorrência.

2.2 OAS (Authorization Server)

O OAS está em `oas/oas_server.py` e executa em `127.0.0.1:6000`. Funções principais:

- **CreateRegistration** – registo de utilizadores com:
  - chave pública Ed25519 (`pub_key`, base64);
  - **hpw_record** derivado com PBKDF2 no cliente;
  - atributos (`attrs`), opcionalmente em hash (ex.: email).
- **AuthStart/AuthResponse/AuthResult** – autenticação baseada em challenge–response com assinatura Ed25519 e conhecimento de password;
- emissão de **JWT** com Ed25519 (EdDSA), com `sub` = anonymous id, `scope` e `exp` curto;
- **GetMyRegistration**, **ModifyRegistration**, **DeleteRegistration** com base em token JWT.

O estado de utilizadores é guardado em `oas/data/users_db.json`: `anon_id` → { `pub_key`, `pwd_hash`, `attrs`, `status`, `created_at` }.

2.3 OAMS (Sharing & Access Management)

O OAMS está em `oams/oams_server.py` e executa em `127.0.0.1:6001`. Funcionalidades:

- **CreateSharingRegistration**: regista partilhas de ficheiros;
- **DeleteSharingRegistration**: revoga partilhas existentes;
- **CheckAccess**: decide `allowed=True/False` para um par `(file_id, permission)`.

Cada partilha é guardada em `oams/data/shares_db.json` com:

- `share_id` (UUID);
- `file_id` (identificador cifrado do ficheiro);
- `owner_fpr`, `authorized_fpr` = SHA-256 da `pub_key` em base64;
- `permissions` (por exemplo `["obss:get","obss:search"]`);
- `status` (`"active"` ou `"revoked"`).

O OAMS valida tokens JWT do OAS (com `verify_oas_jwt`), verifica `sub`, `scope` e decide se o fingerprint do chamador coincide com `owner_fpr` ou `authorized_fpr`.

2.4 Cliente e Multi-secção

O cliente em `client/client.py` usa:

- `client/crypto_client.py` – chaves Ed25519, PBKDF2 local, assinaturas, verificação de respostas e JWT do OAS;
- `client/obss_gateway.py` – cifragem de blocos, tokens de pesquisa, file IDs cifrados, índice local.

A variável de ambiente `CLIENT_SECAO` (ex.: `userA`, `userB`) define a "conta" ativa. Para cada secção:

- `client/keys/<SECAO>/priv_key.pem`;
- `client/keys/<SECAO>/pub_key.b64`;
- `client/keys/<SECAO>/hpw_meta.txt` (hpw_record associado);
- `client/keys/<SECAO>/oas_token.txt`;
- `client/index/<SECAO>/client_index.json` (mapa nome → blocos, keywords, file_id).

Chaves simétricas globais do cliente:

- `client/keys/enc_key.txt` – MASTER_KEY (AES-GCM para blocos);
- `client/keys/mac_key.txt` – SEARCH_KEY (HMAC para tokens de pesquisa e cifragem de `file_id`).

3 Criptografia e Segurança

3.1 Cifragem de blocos e metadados

Blocos de ficheiros (cliente/servidor)

- AES-GCM com MASTER_KEY (256 bits);
- nonce:
  - se `DEDUP=ON`: `nonce = SHA-256(plaintext)[:12]`;
  - caso contrário: nonce aleatório de 96 bits.
- `block_id` = SHA-256(ciphertext).

No OBSS, cada bloco é guardado como JSON com `nonce`, `ciphertext`, `tag`. Os metadados `index.json` e `filemap.json` são cifrados com AES-GCM e uma meta key distinta.

Searchable encryption

- **Token de pesquisa**:

$$\texttt{search\_token} = \text{HMAC\_SHA256}(\texttt{SEARCH\_KEY}, \texttt{keyword})$$

- Índice no servidor: `search_token` → lista de `file_id` cifrados;
- O `file_id` é obtido com:

$$\texttt{enc\_file\_id} = \text{AES-GCM}(\texttt{SEARCH\_KEY}, \texttt{filename})$$

O OBSS só vê tokens e `file_id` cifrados; não conhece nomes de ficheiros nem keywords em claro.

3.2 Derivação de Passwords com PBKDF2 (Cliente)

Foi reforçada a derivação de passwords no cliente, substituindo SHA-256 simples por **PBKDF2-HMAC-SHA256**, em linha com o requisito NFR-4 (*uso de PBKDF2, bcrypt ou Argon2*) e recomendações OWASP.

Parâmetros definidos no cliente

- `PBKDF2_ITERATIONS = 100_000`;
- `PBKDF2_HASH_NAME = "sha256"`;
- `PBKDF2_KEY_LENGTH = 32` (256 bits).

Funções principais

- `compute_hpw(password, salt, iterations)`:

$$\texttt{hpw} = \text{PBKDF2-HMAC-SHA256}(\texttt{password}, \texttt{salt}, \texttt{iterations})$$

- `make_hpw_record(password, iterations)` gera um **hpw_record** com formato:

```
"pbkdf2_sha256$<iterations>$<hex_salt>$<hex_hpw>
```

  onde:

  - `salt` tem 16 bytes;
  - `hpw` é o resultado PBKDF2 (32 bytes).

- `parse_hpw_record(hpw_record)` devolve `(algorithm, iterations, salt, hpw_bytes)`;
- `verify_hpw_local(password, hpw_record)` permite verificar, no cliente, se uma password corresponde ao `hpw_record`, usando `hmac.compare_digest`.

Desta forma, o cliente nunca guarda a password em claro; apenas guarda o `hpw_record` associado à secção em `hpw_meta.txt`.

3.3 Hash de Password no Servidor (OAS)

No OAS, o campo `pwd_hash` armazenado em `users_db.json` é um **hash forte** derivado do `hpw_record` recebido do cliente:

$$\texttt{stored\_pwd\_hash} = \text{PBKDF2-HMAC-SHA256}(\texttt{hpw\_record}, \texttt{salt\_OAS} \| \texttt{PEPPER}, \texttt{iters})$$

Formato:

`"pbkdf2_sha256$iters$salt_hex$hash_hex`

Assim, a password do utilizador é protegida por duas camadas de PBKDF2: uma no cliente (salt específico por secção) e outra no servidor (salt + pepper).

3.4 Assinaturas e JWT

- Todas as chaves de utilizador são pares Ed25519 gerados no cliente;
- O cliente assina:
  - pedidos de `CreateRegistration`, `AuthStart`, `AuthResponse`;
  - pedidos ao OAMS (`CreateSharingRegistration`, etc.);
- O OAS e o OAMS assinam respostas relevantes com as suas chaves privadas Ed25519;
- Tokens JWT são emitidos pelo OAS com:
  - `alg="EdDSA"`, `typ="JWT"`;
  - `sub` = `pub_key` em base64;
  - `iat`, `exp` (curto, 2 minutos), `jti`, `nonce`;
  - `scope`, por exemplo `"obss:read obss:share"`.

O OBSS não interpreta tokens diretamente; sempre que precisa de uma decisão de autorização consulta o OAMS com `CheckAccess`.

4 Fluxos Principais

4.1 Registo & Login

Registo (CreateRegistration)

1. Cliente gera par de chaves Ed25519 para a secção;
2. Cliente deriva `hpw_record` com PBKDF2 (100k iterações) e guarda-o localmente;
3. Cliente envia `CreateRegistration` com `pub_key`, `pwd_hash = hpw_record` e `attrs`, assinando o body;
4. OAS verifica a assinatura, deriva `stored_pwd_hash` com PBKDF2+pepper e grava em `users_db.json`.

Login (AuthStart / AuthResponse / AuthResult)

1. Cliente envia `AuthStart` com `pub_key`, `ctx` e assinatura;
2. OAS responde com `AuthChallenge` (`challenge_id`, `nonce`, `ts`, `ctx`);
3. Cliente reconstrói `hpw` a partir do `hpw_record` e password, constrói `m = "nonce_b64|ts|ctx|hpw_hex"` e assina `m`;
4. Cliente envia `AuthResponse` com `sig_m` e `pw_proof = hpw_hex`;
5. OAS verifica `sig_m`, verifica `pw_proof` contra `stored_pwd_hash` e, se ok, emite JWT em `AuthResult`.

4.2 Armazenamento e Pesquisa no OBSS

PUT (upload)

- Cliente lê o ficheiro por blocos, cifra cada bloco com AES-GCM e envia `STORE_BLOCK`;
- Cliente calcula `enc_file_id = encrypt_file_id(filename)`;
- Para cada keyword, calcula `search_token` (HMAC) e envia `STORE_TOKEN` com `token`, `file_id` e `block_ids`;
- Índice local da secção é atualizado em `client_index.json`.

SEARCH e GET com OAMS

- Cliente envia `SEARCH` com `search_token` e JWT;
- OBSS consulta `index.json` e, para cada `file_id`, chama `CheckAccess(file_id, "obss:search")` no OAMS;
- Apenas `file_id` autorizados são devolvidos;
- `GET` por keyword usa o mesmo filtro, mas envia também os blocos cifrados do ficheiro.

Este script cria múltimos *threads* a fazer uploads simultâneos de ficheiros de teste, exercitando o acesso concorrente a `index.json` e `filemap.json`, bem como a deduplicação de blocos.

5 Conclusão e Trabalho Futuro

O sistema desenvolvido cumpre os objetivos principais do enunciado:

- armazenamento de ficheiros em blocos cifrados com **AES-GCM**, garantindo confidencialidade e integridade;
- suporte para **searchable encryption** via tokens determinísticos (HMAC-SHA256) sobre keywords;
- autenticação forte com **challenge-response** e confirmação de password baseada em `hpw` e **PBKDF2**;

- emissão e validação de **tokens JWT** com **Ed25519**, usados pelo OBSS e pelo OAMS;
- mecanismo de **partilha** centrado no OAMS, que aplica políticas de acesso por ficheiro sem expor chaves públicas em claro;
- **persistência de estado** em todos os componentes.

Como trabalho futuro, poderiam ser exploradas as seguintes extensões:

- proteção adicional dos ficheiros JSON de OAS e OAMS (e.g., cifragem em repouso);
- introdução de **TLS** entre todos os serviços, aproximando o sistema de um ambiente de produção;
- implementação de listas de revogação para `jti` dos tokens JWT e *logging* estruturado para auditoria;
- suporte a múltiplos OAMS e OBSS replicados, com tolerância a falhas;
- implementação de uma interface gráfica ou API REST sobre o cliente de linha de comandos.

Este projeto demonstra, de forma integrada, a aplicação de conceitos de criptografia, autenticação, autorização e segurança de sistemas distribuídos, alinhados com os tópicos da unidade curricular de Segurança e Sistemas de Computadores.