An Approach to Oblivious, Decentralized IAM Integrated with a Searchable, Secure, Encrypted Block-Storage Service

Relatório Técnico do Projeto 2

Autores: Tomás Alexandre 73213, Nicolae Iachimovschi 73381
Unidade Curricular: Segurança e Sistemas de Computadores
Professor: Henrique João Lopes Domingos.
Ano Letivo: 2025/2026

Dezembro , 2025

1. Introdução

Este projeto tem como objetivo estender o sistema de armazenamento seguro desenvolvido no Projeto 1, introduzindo um modelo multi-servidor oblivious que separa autenticação, gestão de acessos e armazenamento de dados.

O sistema passa a ser composto por três serviços distintos:

- **OAS – Oblivious Auth Server**, responsável pela gestão de identidades e emissão de tokens de autenticação (JWT);
- **OAMS – Oblivious Access Management Server**, responsável por listas de controlo de acesso (ACLs) e partilhas de chaves de ficheiro entre utilizadores;
- **OBSS – Oblivious Block Storage Server**, responsável por armazenar blocos de ficheiros cifrados e realizar pesquisas sobre dados cifrados.

A implementação foi realizada em Java, utilizando TLS 1.2/1.3, AES-GCM, ECDSA (ES256), PBKDF2 e SHA-256, cumprindo os requisitos funcionais e de segurança definidos no enunciado do Projeto 2.

2. Arquitetura Geral do Sistema

O sistema segue uma arquitetura modular composta por vários componentes, cada um com responsabilidades bem definidas no processo de autenticação, gestão de acessos e armazenamento seguro:

- **BlockStorageServer.java (OBSS) -** Servidor de blocos cifrados. Armazena os blocos de ficheiro, metadados de associação entre *docId*, *blockId* e *tokens* de pesquisa, e responde a comandos remotos como *STORE_BLOCK*, *GET_BLOCK*, *GET_DOC_BLOCKS* e *SEARCH*, sempre sobre uma ligação *TLS*. O servidor nunca vê dados em claro nem palavras-chave originais, apenas hashes e identificadores.
- **ObliviousAuthServer.java (OAS) -** Responsável pela gestão de utilizadores e pela emissão de tokens JWT. Mantém um registo persistente de utilizadores com chave pública EC (P-256) e respetiva *fingerprint*; password protegida com PBKDF2; e atributos associados. Expõe comandos *CREATE_REG*, *MODIFY_REG*, *DELETE_REG*, *AUTH_START* e *AUTH_FINISH*, todos assinados digitalmente pelo utilizador e validados no servidor.
- **ObliviousAccessServer.java (OAMS) -** Responsável pela gestão de ACLs por documento. Mantém, para cada docId, uma lista de entradas contendo o *fingerprint* do dono; o *fingerprint* do utilizador com quem foi partilhado; o

conjunto de permissões (GET, SEARCH, SHARE); e o *blob* cifrado com a chave de ficheiro (quando partilhada). Expõe comandos *CREATE_SHARE*, *DELETE_SHARE* e *CHECK_ACCESS*, todos assinados pelo cliente e validados no servidor.

- **Project2Client.java** - Cliente interativo com menu textual, que centraliza toda a lógica do lado do utilizador: criação e armazenamento local da identidade (par de chaves EC + *fingerprint*); registo, autenticação e gestão de conta no OAS; upload, download e pesquisa de ficheiros no OBSS, usando sempre tokens JWT válidos; e a comunicação com o OAMS para partilha, revogação e validação de acessos, bem como para obtenção da chave de ficheiro quando o acesso é permitido.
- **JwtUtils.java -** Módulo auxiliar responsável pela criação e validação de tokens JWT assinados com ES256, responsável pela geração de pares de chaves EC (P-256); criação de tokens com *claims* padrão (iss, sub, exp, iat, jti) e com o campo de scope; verificação de assinatura, validade temporal e *issuer*; e cálculo da *fingerprint* da chave pública com SHA-256.
- **CryptoConfig.java -** Gere a configuração criptográfica do sistema, tal como no Projeto 1.

3. Segurança e Criptografia Implementada

O sistema foi desenhado para garantir confidencialidade, integridade, autenticação, controlo de acesso e privacidade das pesquisas, seguindo um modelo de adversário "honest-but-curious", onde os servidores executam corretamente o protocolo, mas tentam obter o máximo de informação a partir dos dados observados.

1. **Confidencialidade** - Os ficheiros são sempre cifrados no cliente antes de serem enviados:
    a. Utilização de AES-GCM 256 bits, garantindo cifragem e autenticação integrada.
    b. Cada bloco recebe um IV aleatório, evitando repetições.
    c. A chave de ficheiro (*fileKey*) é gerada no cliente e nunca é armazenada em claro nos servidores.
    d. As bases de dados internas do OAS e OAMS são cifradas com **AES-GCM**, protegendo passwords, identidades e ACLs.
    e. PBKDF2-HMAC-SHA256 com 200 000 iterações e *salt* de 16 bytes, dificultando ataques offline.
2. **Integridade e Autencicidade** - A integridade dos dados e das comunicações é garantida através de vários mecanismos:
    a. **Tags GCM -** qualquer alteração num bloco impede a sua decifragem.

   b. **Assinaturas ECDSA (ES256) -** todos os pedidos críticos enviados ao OAS/OAMS são assinados pelo cliente; todas as respostas dos servidores são assinadas e verificadas pelo cliente.
   c. **Proteção contra replay -** pedidos incluem timestamp e são rejeitados se excederem a tolerância de 5 minutos.
3. **Privacidade das Pesquisas -** As keywords nunca são enviadas em claro:
   a. Cada keyword é convertida num hash SHA-256, funcionando como "token de pesquisa" opaco.
   b. O OBSS apenas recebe e armazena estes tokens, não podendo reconstituir a keyword original.
4. **Autenticação e Controlo de Acesso -** O sistema separa autenticação (OAS) de autorização (OAMS):
   a. O utilizador autentica-se no OAS através de um processo de desafio-resposta, provando a posse da chave privada e o conhecimento da password.
   b. Em caso de sucesso, o OAS emite um JWT ES256 com permissões (obss:get, obss:search, obss:share).
   c. O OAMS valida estes tokens e gere ACLs por documento.
5. **Comunicação Segura -** Todas as comunicações cliente-servidor utilizam TLS 1.2/1.3, certificados armazenados em *keystore* e *truststore*.

4. Descrição Detalhada das Operações

A implementação cumpre integralmente as operações exigidas no enunciado, tanto no lado cliente como no servidor, garantindo segurança, persistência e capacidade de pesquisa sobre dados cifrados.

4.1. Comando CREATE_REG

1. O cliente gera um par de chaves EC (P-256) e calcula a *fingerprint* SHA-256 da chave pública.
2. O utilizador escolhe uma password local.
3. O cliente prepara o pedido CREATE_REG contendo a chave pública codificada, a password em claro, o número de atributos e o timestamp atual.
4. Todos estes campos são assinados pelo cliente com a sua chave privada EC.
5. O OAS valida a assinatura e o timestamp, deriva a password com PBKDF2, calcula a *fingerprint* e guarda o novo utilizador na base de dados cifrada.
6. O servidor responde com "OK" assinado, que o cliente verifica antes de considerar o registo concluído.

4.2. Comando MODIFY_REG

1. O cliente envia a chave pública, a password atual, a nova password, o conjunto de novos atributos, timestamp e assinatura.
2. O OAS verifica a assinatura, recupera o utilizador pela *fingerprint* e valida a password atual usando PBKDF2.
3. Se for correta, gera novo *salt* e *hash* para a password e substitui os atributos.
4. A base de dados cifrada é atualizada e é devolvida uma resposta "OK" assinada.

4.3. Comando DELETE_REG

1. O cliente envia chave pública, password, timestamp e assinatura.
2. O OAS valida assinatura, password e timestamp.
3. Em caso de sucesso, remove o utilizador, apaga o desafio ativo, e devolve "OK" assinado.
4. O cliente apaga também o ficheiro local user.keys e invalida o token JWT em memória.

4.4. Comando AUTH

1. O cliente envia a sua chave pública ao OAS.
2. O OAS verifica se o utilizador existe e envia de volta um **nonce** (desafio) assinado, garantindo que veio do servidor legítimo.
3. O cliente envia a **password** e assina o desafio recebido.
4. O OAS valida a password, a assinatura e o nonce.
5. Se estiver tudo correto, o servidor emite um **token JWT** com as permissões do utilizador.
6. O cliente valida a assinatura do token e passa a usá-lo em todas as operações seguintes.

4.5. Comando UPLOAD_FILE

1. O utilizador escolhe um ficheiro e fornece keywords.
2. O cliente gera um docId, uma chave de ficheiro (fileKey) e transforma cada keyword em hash SHA-256.
3. O ficheiro é lido em blocos e para cada bloco é criado um blockId; o bloco é cifrado com AES-GCM e o cliente envia ao OBSS.
4. O OBSS armazena os blocos e devolve "OK" (ou "OK_DUP" se já existir).
5. Após o upload, o cliente envia ao OAMS o comando CREATE_SHARE para criar a ACL do documento, com permissões iniciais GET, SEARCH e SHARE, incluindo a chave do ficheiro.
6. O OAMS valida e guarda a ACL cifrada.

4.6. Comando DOWNLOAD_FILE

1. O utilizador indica o docId pretendido.
2. O cliente invoca CHECK_ACCESS no OAMS, enviando o token JWT, o docId, o pedido de permissão "GET" e uma assinatura sobre o pedido.
3. Se o acesso for permitido, o OAMS devolve "OK" assinado e o *key blob* com a chave do ficheiro (no dono é a chave original; nos partilhados é a mesma chave recebida na criação da partilha).
4. O cliente constrói a fileKey a partir do *blob* e contacta o OBSS
5. Os blocos em claro são escritos sequencialmente num ficheiro local.
6. Se algum bloco falhar (tag inválida ou erro de permissão), a operação é abortada para preservar a integridade.

4.7. Comando SEARCH_FILE

1. O cliente primeiro verifica, através do OAMS, se o token JWT inclui o *scope*.
2. O utilizador introduz uma keyword.
3. O cliente envia ao OBSS o comando SEARCH com o JWT e hashedKw.
4. O OBSS compara o hash com os tokens armazenados e devolve uma lista de docId candidatos.
5. Para cada docId devolvido, o cliente invoca novamente CHECK_ACCESS no OAMS.
6. Desta forma, o OBSS não sabe para que utilizador está a pesquisar e o OAMS não vê as keywords, apenas docIds, mantendo o modelo oblivious.

4.8. Comando SHARE_FILE

1. O dono do documento conhece o docId e obtém a fileKey através de CHECK_ACCESS.
2. No menu do cliente, indica o docId, o *fingerprint* do utilizador destinatário e o conjunto de permissões (GET, SEARCH, SHARE).
3. O cliente envia ao OAMS um pedido CREATE_SHARE com JWT atual; docId; *fingerprint* do destinatário; permissões; *key blob* e timestamp.
4. O OAMS valida o token.
5. Atualiza a ACL do docId, removendo previamente entradas duplicadas para o mesmo par dono/destinatário, e guarda as novas ACLs cifradas.
6. Devolve "OK" assinado ao cliente.

4.9. Comando DELETE_FILE

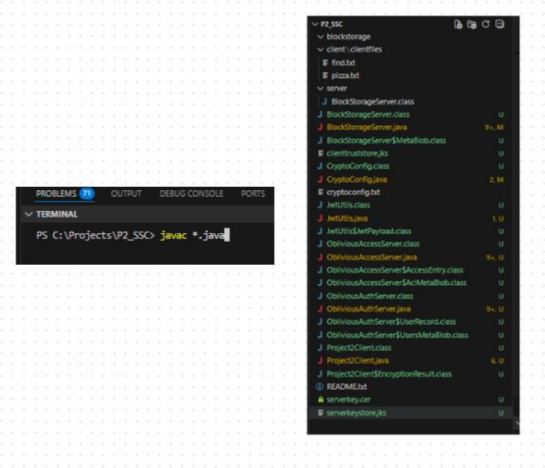
1. O dono indica o docId e o *fingerprint* anteriormente partilhado.
2. O cliente envia DELETE_SHARE com JWT, docId, grantee, timestamp, chave pública e assinatura.

3. O OAMS valida todos os elementos, remove a entrada de ACL correspondente e atualiza o ficheiro cifrado de metadados.
4. A resposta "OK" assinada confirma que o destinatário deixou de ter acesso ao ficheiro.

5. Simulação Prática

Nesta secção descrevemos uma sequência de passos para demonstrar o funcionamento do sistema Oblivious IAM + Secure Block Storage, desde a configuração inicial até à partilha de ficheiros entre dois utilizadores.

1. **Compilação do Projeto**
   a. *javac *.java*



2. **Execução dos Servidores -** Os servidores são iniciados com (cada um em um terminal individual)

   a. *java ObliviousAuthServer*

```
PS C:\Projects\P2_SSC> java ObliviousAuthServer
Config carregada de: cryptoconfig.txt
Chave de assinatura OAS gerada: oas_signing_keypair.bin
OAS iniciado: utilizadores carregados = 0
ObliviousAuthServer (OAS) a escutar em TLS na porta 6000
```

   b. *java ObliviousAccessServer*

```
PS C:\Projects\P2_SSC> java ObliviousAccessServer
Config carregada de: cryptoconfig.txt
Gerado novo par de chaves OAMS em oams_signing_keypair.bin
OAMS iniciado: docIds com ACLs = 0
ObliviousAccessServer (OAMS) a escutar em TLS na porta 7000
```

   c. *java BlockStorageServer*

```
PS C:\Projects\P2_SSC> java BlockStorageServer
Config carregada de: cryptoconfig.txt
[OBSS Stats]
 - Blocos Ã?nicos: 0
 - Documentos: 0
 - Keywords: 0
Secure BlockStorageServer (TLS) a escutar na porta 5000
```

3. Execução dos Clientes Interativos

a. *Cliente 1 (Ana)*

```
PS C:\Projects\P2_SSC> java "-Dp2.userKeyFile=alice.keys" Project2Client
>>
[Init] Chaves pÃºblicas dos servidores carregadas.

========== PROJECT 2 SECURE CLIENT ==========
User: NÃ£o registado/carregado
Auth: NÃƒO

1. Create Identity & Register (OAS)
2. Authenticate (OAS)
3. Manage Account (Modify/Delete)
----------------------------------
4. Upload File (Encrypted -> OBSS + OAMS)
5. Download File (Decrypt <- OBSS + OAMS)
6. Search (Blind Search)
----------------------------------
7. Share File (OAMS)
8. Revoke Share (OAMS)
0. Exit
OpÃ§Ã£o:
```

b. *Cliente 2 (Bob)*

```
PS C:\Projects\P2_SSC> java "-Dp2.userKeyFile=bob.keys" Project2Client
>>
[Init] Chaves pÃºblicas dos servidores carregadas.

========== PROJECT 2 SECURE CLIENT ==========
User: NÃ£o registado/carregado
Auth: NÃƒO

1. Create Identity & Register (OAS)
2. Authenticate (OAS)
3. Manage Account (Modify/Delete)
----------------------------------
4. Upload File (Encrypted -> OBSS + OAMS)
5. Download File (Decrypt <- OBSS + OAMS)
6. Search (Blind Search)
----------------------------------
7. Share File (OAMS)
8. Revoke Share (OAMS)
0. Exit
OpÃ§Ã£o:
```

4. Registo do Cliente

a. Cliente 1 (Ana)

```
1. Create Identity & Register (OAS)
2. Authenticate (OAS)
3. Manage Account (Modify/Delete)
-----------------------------------
4. Upload File (Encrypted -> OBSS + OAMS)
5. Download File (Decrypt <- OBSS + OAMS)
6. Search (Blind Search)
-----------------------------------
7. Share File (OAMS)
8. Revoke Share (OAMS)
0. Exit
OpÃ§Ã£o: 1

== CRIAR IDENTIDADE ==
Identidade guardada em disco.
Nova identidade criada.
Fingerprint (User FP): 3556edade2e0dd09b8f265114bee5ef99fe8d09a92704bfbcbfefa17b37fcbfe
Escolha uma Password: 12345
Servidor: OK (Assinatura Validada)
Identidade criada e salva em 'alice.keys'.
```

a. Cliente 2 (Bob)

```
1. Create Identity & Register (OAS)
2. Authenticate (OAS)
3. Manage Account (Modify/Delete)
-----------------------------------
4. Upload File (Encrypted -> OBSS + OAMS)
5. Download File (Decrypt <- OBSS + OAMS)
6. Search (Blind Search)
-----------------------------------
7. Share File (OAMS)
8. Revoke Share (OAMS)
0. Exit
OpÃ§Ã£o: 1

== CRIAR IDENTIDADE ==
Identidade guardada em disco.
Nova identidade criada.
Fingerprint (User FP): d821d8364fe673c5cc70fe1c594c3fe96b142d227b86a9237afa8115a4396e79
Escolha uma Password: 123
Servidor: OK (Assinatura Validada)
Identidade criada e salva em 'bob.keys'.
```

5. Autenticação do Cliente

a. Cliente 1 (Ana)

```
OpÃ§Ã£o: 2
Password: 12345
Autenticado com sucesso.
```

a. Cliente 2 (Bob)

```
OpÃ§Ã£o: 2
Password: 123
Autenticado com sucesso.
```

6. **Upload do Ficheiro –** Terminal da Ana

```
OpÃ§Ã£o: 4
Caminho do ficheiro: C:\Projects\P2_SSC\client\clientfiles\pizza.txt
Keywords (separadas por vÃ rgula): comida
A encriptar e enviar docID: 509119ba-5399-4caf-930d-fecc8cb53c51
Servidor: OK (Assinatura Validada)
Ficheiro guardado com sucesso! DocID: 509119ba-5399-4caf-930d-fecc8cb53c51
```

7. **Partilhar Ficheiro -** Terminal da Ana

```
OpÃ§Ã£o: 7
DocID: 509119ba-5399-4caf-930d-fecc8cb53c51
Grantee FP: d821d8364fe673c5cc70fe1c594c3fe96b142d227b86a9237afa8115a4396e79
Perms (ex: GET SEARCH): GET SEARCH
Servidor: OK (Assinatura Validada)
Partilha criada com sucesso para d821d8364fe673c5cc70fe1c594c3fe96b142d227b86a9237afa8115a
4396e79
```

8. **Download do Ficheiro –** Terminal do Bob

```
OpÃ§Ã£o: 5
DocID: 509119ba-5399-4caf-930d-fecc8cb53c51
Ficheiro decifrado e salvo: downloaded_509119ba-5399-4caf-930d-fecc8cb53c51.bin
```

9. **Pesquisa por Keyword –** Terminal do Bob

```
OpÃ§Ã£o: 6
Keyword: comida
Resultados com acesso (apÃ³s ACL OAMS):
 - 509119ba-5399-4caf-930d-fecc8cb53c51
```

6. Conclusão

Em suma, o sistema Oblivious IAM + Secure Block Storage foi implementado com sucesso, integrando OAS, OAMS e OBSS para suportar registo/autenticação, gestão de partilhas e armazenamento cifrado de ficheiros, garantindo confidencialidade, integridade e autenticidade com AES-GCM, PBKDF2, ECDSA, SHA-256 e TLS. A solução cumpre os requisitos técnicos e de segurança definidos no enunciado e demonstra um fluxo completo de gestão de identidades, permissões e ficheiros.