Searchable Secure Encrypted Block Storage Service

Relatório Técnico do Projeto 1

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

Outubro, 2025

1. Introdução

Este projeto tem como objetivo o desenvolvimento de um sistema cliente-servidor seguro de que permite armazenar ficheiros de forma cifrada, persistente e pesquisável através de palavras-chave. O sistema garante **confidencialidade**, **integridade** e **autenticidade** dos blocos armazenados e suporta *searchable encryption*, permitindo que o servidor execute pesquisas sobre dados cifrados sem conhecer o seu conteúdo.

A implementação foi realizada em **Java**, utilizando **TCP/TLS** para a comunicação segura entre o cliente e o servidor, e cumpre os requisitos funcionais e de segurança definidos no enunciado do projeto.

2. Arquitetura Geral do Sistema

O sistema segue uma arquitetura modular composta por quatro componentes principais, cada um com responsabilidades bem definidas no processo de armazenamento, pesquisa e recuperação segura de dados cifrados:

- **BlockStorageServer.java -** Responsável por armazenar blocos cifrados, manter os meta dados de forma cifrada e responder a comandos remotos através de uma ligação segura TLS. Implementa a lógica de *deduplicação*, associação entre identificadores de documento e blocos, e a gestão de *tokens* de pesquisa *searchable tokens* sem nunca aceder ao conteúdo original dos ficheiros.
- **BlockStorageClient.java -** Cliente interativo com menu textual que permite ao utilizador executar operações como *PUT*, *GET*, *LIST*, *SEARCH* e *CHECK INTEGRITY*. É responsável por dividir os ficheiros em blocos, cifrá-los com AES-GCM, criar *tags* de autenticação e enviar os metadados e *tokens* de pesquisa ao servidor de forma segura.
- **ClTest.java -** Cliente de teste automático compatível com os comandos exigidos no enunciado. Permite testar diretamente as funcionalidades essenciais do sistema (armazenamento, pesquisa e verificação de integridade), sem o menu interativo, facilitando a demonstração e validação dos requisitos de segurança e funcionalidade.
- **CryptoConfig.java** - gere toda a configuração criptográfica do sistema, incluindo o algoritmo de cifra (*AES/GCM/NoPadding*), tamanho das chaves, algoritmo *HMAC* utilizado para os *tokens* de pesquisa e tamanho dos blocos de dados. Lê as definições a partir do ficheiro (*cryptoconfig.txt*), permitindo ajustar parâmetros criptográficos sem necessidade de alterar o código-fonte.

3. Segurança e Criptografia Implementada

O sistema garante as propriedades de **confidencialidade**, **integridade** e **autenticidade** dos dados armazenados, seguindo um modelo de adversário "*honest-but-curious*", onde o servidor pode aceder aos ficheiros cifrados, mas não deve ser capaz de interpretar o seu conteúdo.

1. **Confidencialidade** - Todos os blocos são cifrados com o algoritmo AES-GCM, utilizando chaves de 256 *bits* geradas no cliente. O modo GCM, além de cifrar, também fornece autenticação integrada, impedindo modificações não detetadas. As chaves são armazenadas localmente em ficheiros cifrados (**client_key.enc**, **client_kw_key.enc**, **client_auth.enc**), protegidas por password e derivadas através de PBKDF2 com 200 000 iterações, garantindo resistência a ataques de força bruta.
2. **Integridade e Autencicidade** - Para reforçar a integridade e autenticidade de cada bloco, é criado um hash (HMAC SHA-256) com base no conteúdo cifrado. O cliente valida o HMAC e a *tag* GCM durante a reconstrução dos ficheiros ou durante o comando *CHECKINTEGRITY*, assegurando que nenhum bloco foi alterado.
3. **Pesquisa por Palavras-Chave** - Realizada através de tokens HMAC determinísticos, criados localmente no cliente a partir das *keywords* fornecidas, sendo armazenados e associados aos identificadores de documento. Desta forma, o servidor pode realizar buscas sem conhecer as *keywords* originais, garantindo privacidade das consultas.
4. **Comunicação Segura** - Todas as comunicações entre cliente e servidor decorrem através de TLS v1.3, com certificados assinados (**serverkeystore.jks**, **clienttruststore.jks**). Assim, as mensagens trocadas (blocos, metadados, *tokens*) são protegidas contra interceção e adulteração durante o transporte.

4. Segurança e Criptografia Implementada

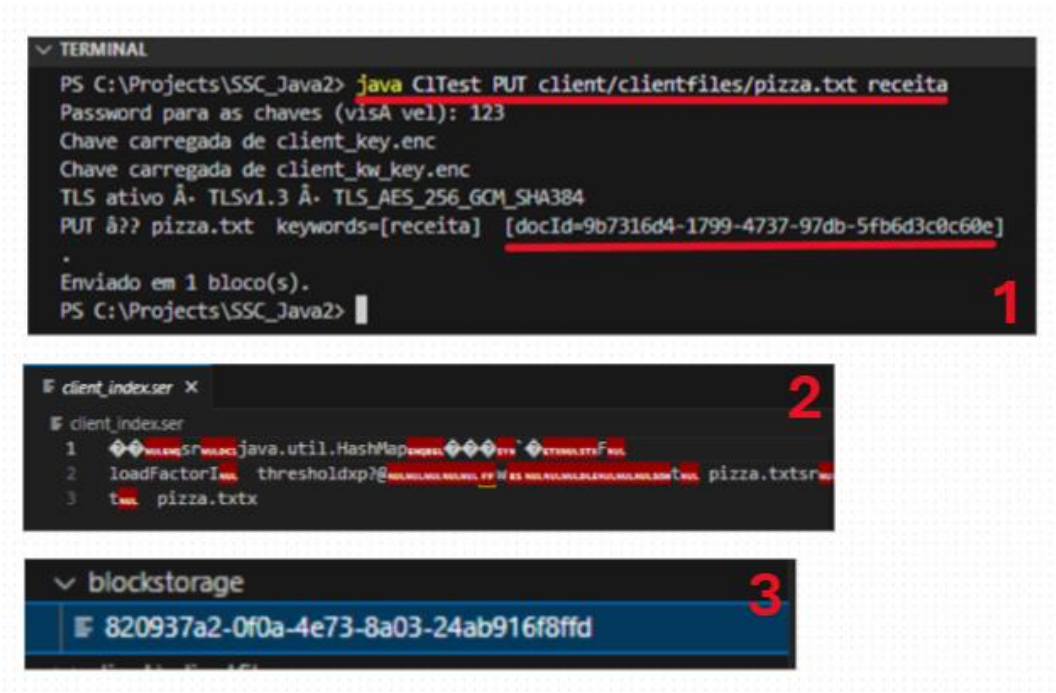
A implementação cumpre integralmente as operações exigidas no enunciado, tanto no lado cliente como no servidor, garantindo segurança, persistência e capacidade de pesquisa sobre dados cifrados.

4.1. Comando PUT

O comando **PUT** é responsável por enviar ficheiros locais do cliente para o servidor, de forma cifrada e autenticada.

1. O ficheiro é fragmentado em blocos de tamanho fixo (configurável no ficheiro *cryptoconfig.txt,* por defeito 4096 bytes).
2. Cada bloco recebe um UUID, que serve de referência no servidor.
3. Cada bloco é cifrado com uma chave simétrica AES de 256 bits.

4. Além da *tag* GCM, é calculado um HMAC-SHA256 sobre o bloco cifrado, utilizando a chave de autenticação (*client_auth.enc*), e assim garantindo autenticação mesmo após o armazenamento prolongado.
5. O cliente cria **tokens HMAC** a partir das *keywords* fornecidas.
6. O cliente envia, via **TLS**, cada bloco cifrado e respetiva *tag*, com o tamanho do bloco, os dados cifrados, a *tag* do HMAC o ID do documento e os *tokens*.
7. O servidor calcula um *hash* SHA-256 do conteúdo cifrado e evita armazenar blocos duplicados, otimizando o espaço em disco.
8. O cliente regista o mapeamento entre o nome do ficheiro, o docId e os blocos no ficheiro *client_index.ser*.
9. Por fim, o ficheiro é armazenado de forma totalmente cifrada e autenticada, permanecendo ilegível mesmo para o administrador do servidor.

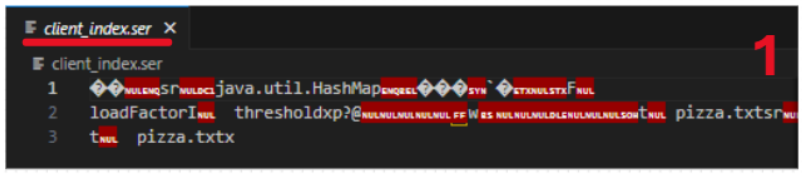


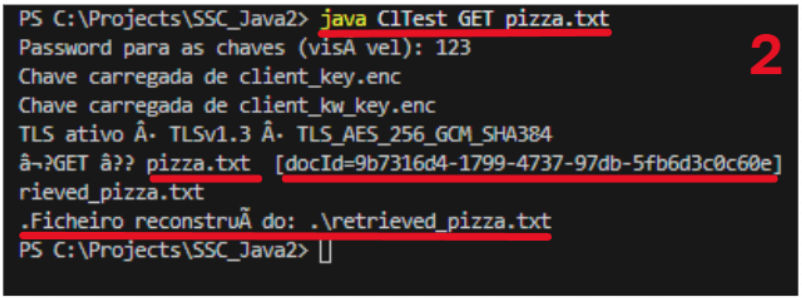
4.2. Comando GET

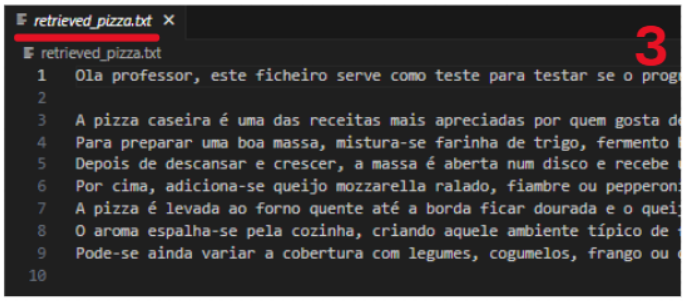
O comando **GET** permite recuperar um ficheiro previamente enviado, garantindo que a integridade dos blocos é verificada antes da reconstrução.

1. O cliente lê o índice local (*client_index.ser*) e identifica o docId e os blocos correspondentes.
2. Para cada bloco, o cliente envia o pedido *GET_BLOCK* ao servidor.
3. O servidor devolve o bloco cifrado e a respetiva *tag* de autenticação.
4. O HMAC é verificado.
5. O bloco é decifrado com AES-GCM, validando a *tag* interna de integridade.

6. Caso algum bloco apresente erro (tag inválida, alteração ou perda), a reconstrução é abortada para garantir integridade total.
7. Após verificação bem-sucedida de todos os blocos, o cliente reconstrói o ficheiro original, gravando-o localmente com o prefixo *retrieved_*.



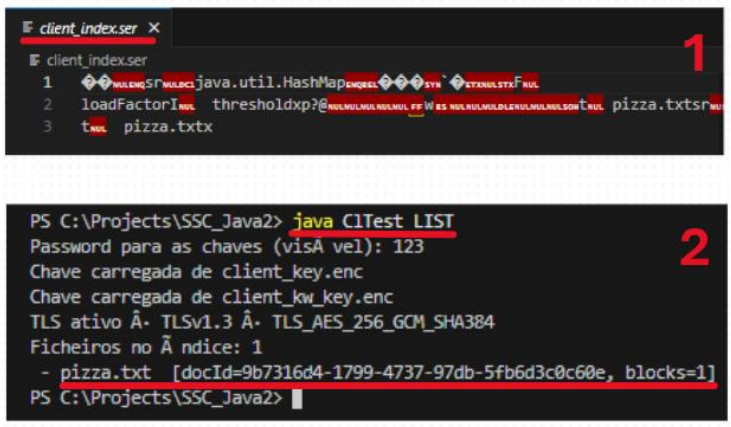




4.3. Comando LIST

O comando **LIST** permite ao utilizador listar todos os ficheiros que foram enviados e estão registados no índice local.

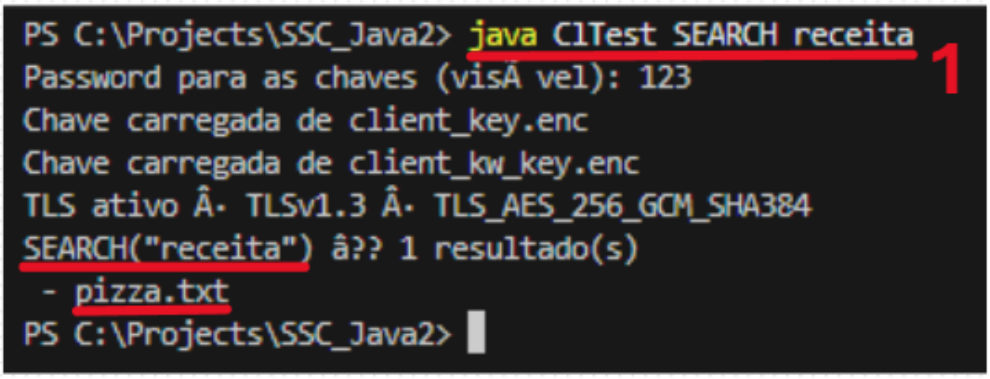
1. O comando lê ficheiro *client_index.ser* e apresenta, para cada entrada o nome do ficheiro original, o indentificador do documento, e o número de blocos associados.



4.4. Comando SEARCH

O comando **SEARCH** permite ao utilizador procurar ficheiros associados a uma determinada keyword, sem que o servidor tenha de conhecer o termo original.

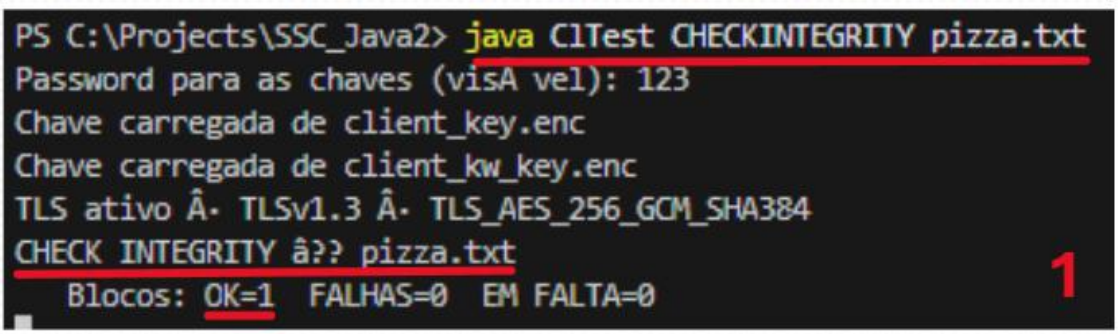
1. O cliente cria um *token* de pesquisa com HMAC-SHA256 sobre a *keyword*.
2. O *token* é enviado ao servidor através do comando SEARCH.
3. O servidor compara o *token* com os tokens armazenados.
4. O servidor devolve uma lista de *docIds* correspondentes.
5. O cliente mapeia os *docIds* para nomes de ficheiros conhecidos e apresenta os resultados.



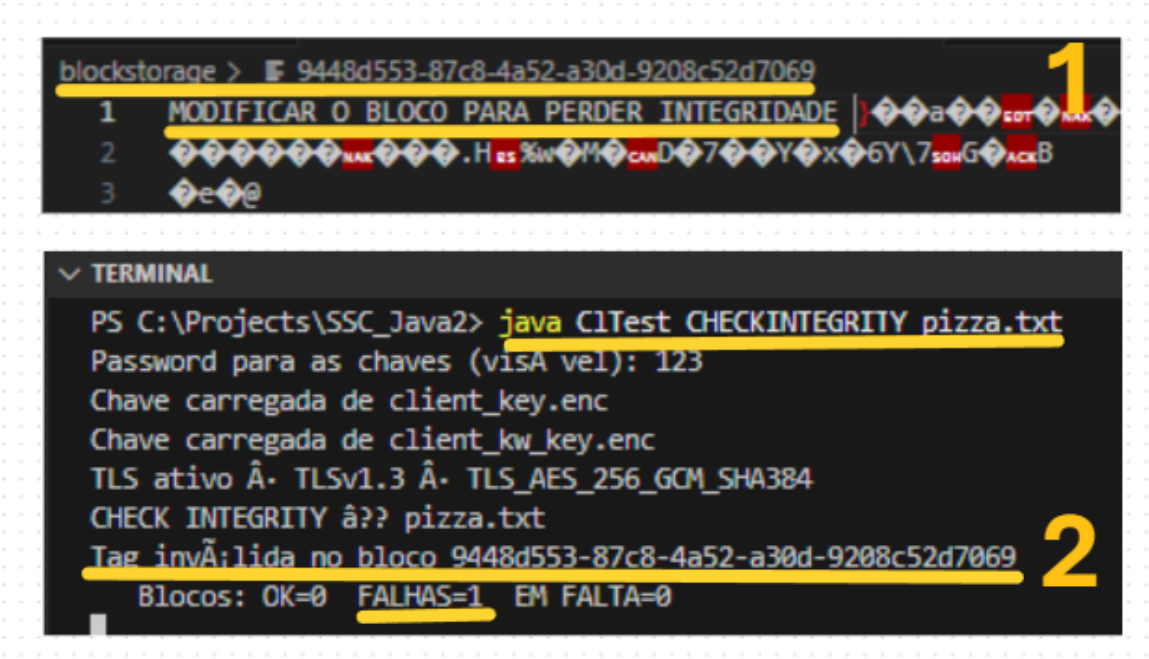
4.5. Comando CHECK INTEGRITY

O comando **CHECK INTEGRITY** permite confirmar que os blocos armazenados no servidor permanecem inalterados, sem necessidade de reconstruir o ficheiro.

1. O cliente obtém do índice local a lista de blocos do ficheiro a validar.
2. Para cada bloco envia *GET_BLOCK* e recebe o bloco cifrado e o HMAC.
3. Recalcula o HMAC e compara com o recebido.
4. Tenta decifrar com AES-GCM apenas para confirmar a validade da *tag*.
5. O comando devolve um relatório no terminal com o número de blocos válidos, falhados ou em falta.
6. Se o bloco não foi alterado então não existe falha,



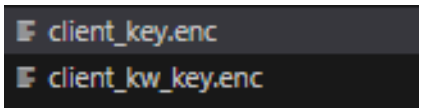
7. Se o bloco for alterado então há falha e a *tag* é invalida

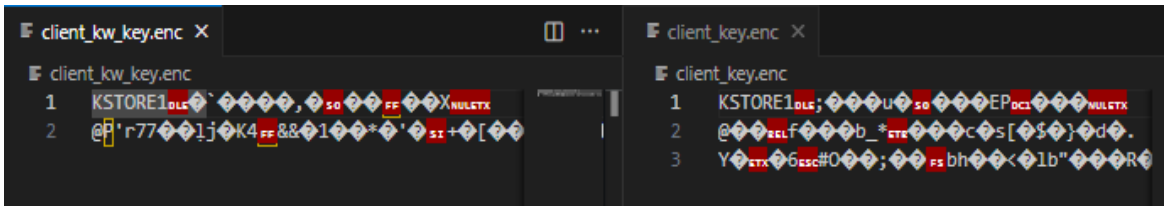


5. Estrutura de Dados e Análise de Segurança

O sistema foi desenvolvido para garantir persistência tanto no lado do cliente como no servidor, armazenando todos os blocos cifrados no diretório *blockstorage/*, e mantém os metadados cifrados no ficheiro *metadata.enc*, protegido por uma chave AES armazenada em *server_meta_key.bin*.

No cliente, o mapeamento entre nomes de ficheiro, identificadores de documento e blocos é guardado no ficheiro *client_index.ser*, guardando as chaves criptográficas do utilizador de forma cifrada nos ficheiros *client_key.enc* e *client_kw_key.enc*.



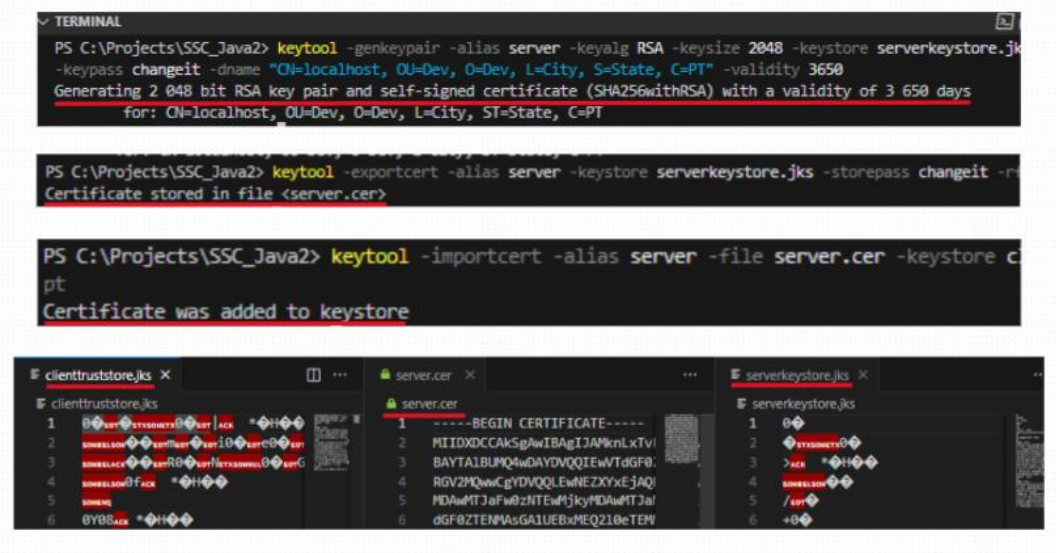


Estes mecanismos asseguram que, após um reinício, o servidor e o cliente conseguem restaurar todo o estado anterior sem perda de informação, garantindo consistência e continuidade no serviço.

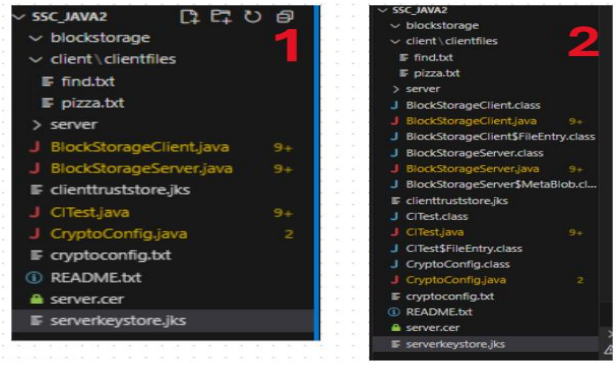
6. Simulação Prática

Neste tópico iremos fazer uma demonstração prática do funcionamento do sistema Searchable Secure Block Storage, demonstrando cada etapa de configuração e execução do projeto.

1. **Criação dos Certificados TLS -** Foram criados certificados digitais para garantir a comunicação segura (TLS 1.3) entre cliente e servidor, usando os seguintes comandos:
    a. *keytool -genkeypair -alias server -keyalg RSA -keysize 2048 -keystore serverkeystore.jks -storepass changeit -keypass changeit -dname "CN=localhost, OU=Dev, O=Dev, L=City, S=State, C=PT" -validity 3650*
    b. *keytool -exportcert -alias server -keystore serverkeystore.jks -storepass changeit -rfc -file server.cer*
    c. *keytool -importcert -alias server -file server.cer -keystore clienttruststore.jks -storepass changeit -noprompt*

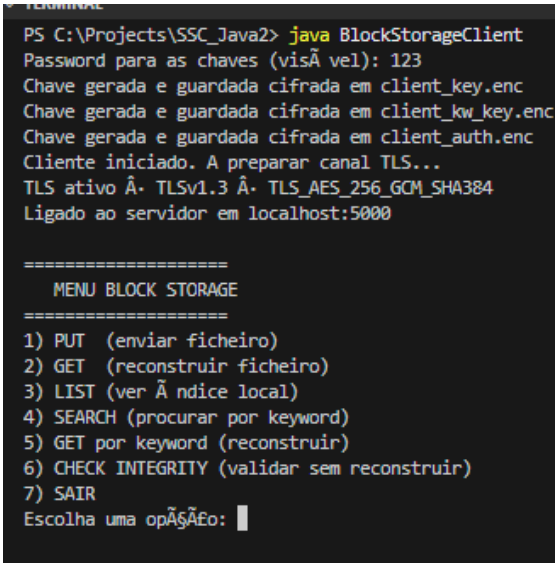


2. Compilação do Projeto - Todos os ficheiros Java foram compilados através do comando:
    a. *javac *.java*

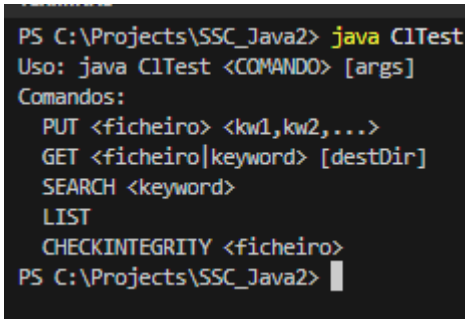


3. **Execução do Servidor -** O servidor é iniciado com:
    a. *java BlockStorageServer*

4. **Execução do Cliente Interativo –** O cliente teste apresenta um menu interativo.
    a. *java BlockStorageClient*



5. **Execução do Cliente de Teste** - O cliente de teste permite executar comandos diretos sem o menu interativo, sendo útil para validação rápida, através do comando:
    a. *java ClTest*



7. Conclusão

Em suma, foram implementadas com sucesso as funcionalidades ***PUT***, ***GET***, ***LIST***, ***SEARCH*** e ***CHECK INTEGRITY***, assegurando confidencialidade, integridade e autenticidade através de AES-GCM, HMAC-SHA256, PBKDF2 e TLS 1.3 O projeto abordou com sucesso todos os tópicos obrigatórios do enunciado, incluindo armazenamento seguro, criptografia aplicada, searchable encryption, persistência de dados, verificação de integridade, canal TLS seguro e cliente de teste funcional, cumprindo integralmente os requisitos técnicos e de segurança definidos.

Como perspetivas de melhoria, seria relevante introduzir autenticação multiutilizador, uma interface gráfica de apoio ao utilizador e mecanismos de replicação e tolerância a falhas, de forma a aumentar a escalabilidade e robustez do sistema.