UNIVERSITATEA PETROL - GAZE DIN PLOIEŞTI
FACULTATEA DE INGINERIE MECANICĂ ȘI ELECTRICĂ
SPECIALIZAREA: AUTOMATICĂ ȘI INFORMATICĂ APLICATĂ
GRUPA 10116, ANUL IV

APLICAȚIE PENTRU VERIFICAREA INTEGRITĂȚII FIȘIERELOR FOLOSIND ALGORITMUL MD5

Coordonator:
Prof. univ. dr. ing. Otilia Cangea

Student:
Schifîrneț Petre-Iustin

INTRODUCERE

În era digitalizării accelerate, protejarea integrității datelor este o necesitate esențială. Fie că este vorba despre transmiterea unui fișier, descărcarea unui document sau partajarea de informații între sisteme, asigurarea că fișierul nu a fost alterat devine o cerință de bază. Una dintre cele mai utilizate metode în acest sens este verificarea hash-ului, iar algoritmul MD5, deși depășit din punct de vedere criptografic, rămâne util în scopuri non-critice, precum validarea rapidă a conținutului unui fișier.

Lucrarea de față prezintă o aplicație software simplă, realizată în limbajul Python, care permite verificarea integrității fișierelor .txt prin calculul hash-ului MD5 și compararea acestuia cu un hash de referință. Scopul este acela de a demonstra, într-un mod practic și accesibil, modul de funcționare al unui algoritm hash și aplicabilitatea sa în verificarea fișierelor.

În primul capitol este realizat un scurt studiu teoretic privind algoritmii de hash, cu accent pe structura și mecanismul intern de funcționare al MD5. Următorul capitol este dedicat proiectării aplicației, prezentând scopul acesteia, tehnologiile folosite și schema logică a funcționării. Capitolul al treilea descrie implementarea efectivă a aplicației, atât din punct de vedere al interfeței grafice, cât și al codului sursă. Capitolul patru reunește rezultatele obținute în urma testării aplicației, oferind interpretări și concluzii asupra eficienței soluției propuse. La final, lucrarea se încheie cu o serie de observații privind posibile îmbunătățiri viitoare și scenarii de utilizare practică.

CAPITOLUL I
Fundamente teoretice privind verificarea integrității fișierelor

1.1 Integritatea datelor și necesitatea verificării

În contextul informaticii moderne, protejarea și verificarea datelor a devenit o activitate de bază pentru orice sistem informatic care procesează sau transferă informații. Integritatea datelor presupune ca acestea să rămână exacte, complete și nealterate, indiferent de operațiile la care sunt supuse sau de mediul prin care sunt transmise. Fie că discutăm despre documente oficiale, fișiere executabile sau arhive de sistem, modificarea neintenționată sau rău intenționată a acestora poate avea consecințe grave asupra funcționalității și securității unui sistem.

De exemplu, într-o situație comună precum descărcarea unui program de pe internet, utilizatorul nu are garanția că fișierul obținut este exact cel pus la dispoziție de dezvoltator. Acest lucru este valabil mai ales în condițiile în care fișierele pot fi interceptate, corupte sau modificate de actori malițioși. Aici intervine noțiunea de verificare a integrității: un proces prin care se compară fișierul descărcat cu o valoare de referință (denumită „hash”) oferită de sursa originală. Dacă cele două valori coincid, fișierul este considerat valid; dacă nu, integritatea sa este compromisă.

Verificarea integrității este folosită pe scară largă în industria IT: în actualizările automate ale aplicațiilor, în verificarea backup-urilor, în semnăturile digitale, în securitatea cibernetică, dar și în simple aplicații educaționale sau casnice. Indiferent de domeniu, obiectivul este același: garantarea faptului că datele nu au suferit modificări nedorite.

1.2 Algoritmi hash – definiție și aplicabilitate

Un algoritm hash este o funcție matematică deterministă care transformă o cantitate de date de orice dimensiune (text, fișier, imagine etc.) într-un șir de caractere de lungime fixă. Această valoare rezultată, numită **hash** sau **amprentă digitală**, este unică (teoretic) pentru fiecare set de date diferit. Un algoritm hash bun are proprietatea că o modificare oricât de mică a inputului va genera un hash complet diferit. Astfel, hash-ul poate fi folosit ca o semnătură digitală a unui fișier.

Există mai multe tipuri de algoritmi hash, fiecare cu niveluri diferite de complexitate și securitate. Cei mai cunoscuți sunt:

a. **MD5 (Message Digest 5)** – rapid și eficient pentru verificări simple, dar vulnerabil la coliziuni;
b. **SHA-1 (Secure Hash Algorithm 1)** – mai sigur decât MD5, dar și el a fost depășit;
c. **SHA-256** – parte din familia SHA-2, mult mai sigur și utilizat în aplicații moderne, inclusiv în blockchain.
d. Aplicațiile algoritmilor hash sunt vaste:
e. **Verificarea integrității fișierelor** – așa cum se va face în aplicația propusă în acest proiect;
f. **Stocarea parolelor** – parolele nu se salvează direct, ci sub formă de hash;
g. **Semnături digitale** – unde hash-ul mesajului este criptat cu cheia privată a semnatarului;
h. **Detecția modificărilor** – în sisteme de versionare sau în securitate.

Este important de menționat că, în ciuda eficienței lor, algoritmii hash nu sunt metode de criptare. Ei nu pot fi „inversați” pentru a obține conținutul original, iar securitatea lor depinde de dificultatea găsirii unui input diferit care să genereze același hash (coliziune).

1.3 Algoritmul MD5 – structură, etape de funcționare, exemplu de cod

Algoritmul **MD5 (Message Digest 5)** a fost dezvoltat de profesorul Ronald Rivest în 1991 și a fost, pentru o perioadă îndelungată, unul dintre cei mai utilizați algoritmi de hashing din lume. El produce un hash de 128 de biți (32 de caractere hexazecimale) indiferent de dimensiunea fișierului analizat.

Deși MD5 nu mai este considerat sigur pentru aplicații criptografice, el continuă să fie utilizat pentru verificarea integrității fișierelor și pentru detecția modificărilor în scenarii non-critice.

Etapele de funcționare ale MD5:

1. **Preprocesare (padding)** – Fișierul este completat cu un bit '1' urmat de un număr de zerouri astfel încât lungimea să fie congruentă cu 448 mod 512. La final se adaugă lungimea originală a mesajului, exprimată pe 64 de biți.
2. **Inițializarea registrelor** – Algoritmul utilizează patru registre (A, B, C, D) de 32 de biți fiecare, inițializate cu valori hexazecimale fixe:
    - A = 0x67452301
    - B = 0xefcdab89
    - C = 0x98badcfe
    - D = 0x10325476
3. **Procesarea în blocuri de 512 biți** – Mesajul este împărțit în blocuri egale, fiecare procesat prin 64 de pași care implică operații logice (AND, OR, XOR), rotații circulare și adunări mod 2^{32}. Se folosesc funcții non-lineare: F, G, H și I.
4. **Generarea hash-ului final** – După procesarea tuturor blocurilor, valorile finale ale registrelor A, B, C și D sunt concatenate și formează hash-ul MD5.

Exemplu de implementare în limbajul C:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <math.h>

typedef uint32_t WORD;  // Definește un tip pentru cuvinte de 32 biți

// Macro pentru rotirea circulară la stânga a unui număr pe 32 biți
#define LEFTROTATE(x, c) (((x) << (c)) | ((x) >> (32 - (c))))

// Constantele de rotație pentru fiecare rundă
const uint32_t r[] = {
   7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22,
   5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20,
   4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23,
   6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21
};

// Valori inițiale ale registrelor A, B, C, D
const uint32_t h0_init = 0x67452301;
const uint32_t h1_init = 0xefcdab89;
const uint32_t h2_init = 0x98badcfe;
const uint32_t h3_init = 0x10325476;

// Constantele K calculate din valoarea absolută a sinusului
uint32_t k[64];

// Inițializarea constantei K
void init_k() {
   for (int i = 0; i < 64; ++i) {
      k[i] = (uint32_t)(fabs(sin(i + 1)) * pow(2, 32));
   }
}

// Funcția principală MD5
void md5(const uint8_t *initial_msg, size_t initial_len, uint8_t *digest) {
   init_k();  // inițializează vectorul K

   uint8_t *msg = NULL;

   // Calcularea dimensiunii noului mesaj cu padding
   size_t new_len = ((((initial_len + 8) / 64) + 1) * 64);
   msg = calloc(new_len, 1);  // Alocă memorie pentru mesajul extins
   memcpy(msg, initial_msg, initial_len);  // Copiază mesajul original
   msg[initial_len] = 0x80;  // Adaugă bitul '1' (padding)

   // Adaugă lungimea originală în biți la sfârșitul mesajului
   uint64_t bits_len = 8 * initial_len;
```

```
memcpy(msg + new_len - 8, &bits_len, 8);

// Inițializează registrele A, B, C, D
uint32_t a0 = h0_init;
uint32_t b0 = h1_init;
uint32_t c0 = h2_init;
uint32_t d0 = h3_init;

// Procesează fiecare bloc de 512 biți
for (size_t offset = 0; offset < new_len; offset += 64) {
    uint32_t *w = (uint32_t *)(msg + offset);  // Pointer la blocul curent

    uint32_t A = a0, B = b0, C = c0, D = d0;

    for (int i = 0; i < 64; i++) {
        uint32_t F, g;

        if (i < 16) {
            F = (B & C) | ((~B) & D);
            g = i;
        } else if (i < 32) {
            F = (D & B) | ((~D) & C);
            g = (5 * i + 1) % 16;
        } else if (i < 48) {
            F = B ^ C ^ D;
            g = (3 * i + 5) % 16;
        } else {
            F = C ^ (B | (~D));
            g = (7 * i) % 16;
        }

        uint32_t temp = D;
        D = C;
        C = B;
        B = B + LEFTROTATE((A + F + k[i] + w[g]), r[i]);
        A = temp;
    }

    // Adaugă rezultatul înapoi în registre
    a0 += A;
    b0 += B;
    c0 += C;
    d0 += D;
}

// Copiază rezultatul final în digest
memcpy(digest, &a0, 4);
memcpy(digest + 4, &b0, 4);
memcpy(digest + 8, &c0, 4);
memcpy(digest + 12, &d0, 4);
```

```
    free(msg);  // Eliberare memorie alocată
}

// Funcție principală pentru testare
int main() {
    const char *msg = "Salut lume!";
    uint8_t result[16];

    md5((const uint8_t *)msg, strlen(msg), result);

    printf("Hash-ul MD5 este: ");
    for (int i = 0; i < 16; i++)
        printf("%02x", result[i]);
    printf("\n");

    return 0;
}
```

```
204
205
206
207 // Funcție principală pentru testare
208
209 int main() {
210
211     const char *msg = "Iustin!";
212
213     uint8_t result[16];
214
215
216
217     md5((const uint8_t *)msg, strlen(msg), result);
218
219
220
221     printf("Hash-ul MD5 este: ");
222
223     for (int i = 0; i < 16; i++)
224
225         printf("%02x", result[i]);
226
227     printf("\n");
228
229
230
```

input

```
Hash-ul MD5 este: 6c1f1b12be24f6f51f95eefc42c6afdd


...Program finished with exit code 0
Press ENTER to exit console.
```

CAPITOLUL II
Proiectarea aplicației

2.1 Scopul aplicației

Scopul aplicației dezvoltate în cadrul acestui proiect este de a permite utilizatorilor să verifice rapid și eficient dacă un fișier text are integritatea păstrată, prin compararea amprentei sale digitale MD5 cu o valoare de referință. Această verificare este utilă, de exemplu, atunci când un fișier este descărcat de pe internet și utilizatorul dorește să se asigure că fișierul nu a fost modificat în timpul transferului sau alterat în mod intenționat.

Aplicația urmărește să ofere o interfață grafică intuitivă, prietenoasă chiar și pentru utilizatorii fără cunoștințe de programare, în care aceștia pot selecta un fișier, pot calcula automat hash-ul MD5 corespunzător și îl pot compara cu un hash introdus manual. Rezultatul

comparației este afișat printr-un mesaj clar care confirmă sau infirmă autenticitatea fișierului analizat.

2.2 Tehnologii utilizate

Pentru implementarea aplicației s-a utilizat limbajul de programare Python 3, datorită simplității sale, a sintaxei clare și a bibliotecilor standard foarte bogate. În mod special, au fost folosite două biblioteci esențiale pentru funcționalitatea proiectului:

- hashlib – modul standard din Python pentru calcularea hash-urilor criptografice. Este stabil, rapid și nu necesită instalare separată. Funcția hashlib.md5() permite calculul amprentei MD5 pentru un șir de octeți sau un fișier citit în mod binar.
- tkinter – biblioteca standard de GUI în Python, ideală pentru crearea de aplicații grafice simple. Aceasta oferă widget-uri precum butoane, câmpuri de text, etichete și ferestre de dialog, necesare pentru o interacțiune facilă cu utilizatorul.

Combinarea acestor două biblioteci face posibilă realizarea unei aplicații funcționale, fără a necesita librării externe sau cunoștințe avansate de programare. Codul rămâne portabil și poate fi rulat pe orice sistem cu Python instalat, fără configurații suplimentare.

2.3 Structura funcțională a aplicației

Arhitectura aplicației este simplă, fiind formată din trei componente logice principale:

- Modulul de interfață grafică (GUI) – se ocupă de interacțiunea cu utilizatorul (selectarea fișierului, afișarea hash-ului și introducerea hash-ului original);
- Modulul de procesare – citește fișierul în format binar și calculează hash-ul MD5 folosind hashlib;
- Modulul de verificare – compară hash-ul calculat cu cel introdus de utilizator și afișează rezultatul în funcție de potrivirea acestora.

Fluxul logic al aplicației:

1. Utilizatorul apasă pe butonul „Alege fișier”, moment în care se deschide o fereastră de tip file dialog pentru selectarea unui fișier .txt.
2. După selectarea fișierului, aplicația citește conținutul acestuia în mod binar și calculează hash-ul MD5, folosind funcția hashlib.md5().hexdigest(). Valoarea este afișată imediat într-un câmp dedicat.
3. Utilizatorul introduce, în câmpul corespunzător, hash-ul MD5 oferit de sursa originală (de exemplu, de către producătorul fișierului).
4. La apăsarea butonului „Verifică hash-ul”, aplicația compară cele două valori. Dacă acestea coincid, aplicația confirmă faptul că fișierul este autentic. Dacă valorile sunt diferite, se semnalează o posibilă modificare.
5. Mesajele de rezultat sunt afișate într-o fereastră pop-up (messagebox), sub forma unui mesaj pozitiv sau negativ .

Această logică simplă asigură un proces de verificare clar, cu pași intuitivi, ușor de urmărit și reprodus de către orice utilizator. Aplicația este orientată spre simplitate și funcționalitate, fără elemente tehnice complicate la nivel de interfață, tocmai pentru a fi ușor de demonstrat și înțeles într-un context educațional.

CAPITOLUL III

Implementarea aplicației și prezentarea funcționării

3.1 Implementarea aplicației

Aplicația a fost dezvoltată în limbajul de programare Python, alegere justificată prin simplitatea sintaxei, suportul extins pentru operații cu fișiere și hash-uri, precum și existența unei biblioteci GUI (tkinter) incluse în distribuția standard. Din punct de vedere tehnic, implementarea este împărțită în mai multe funcții cu roluri bine definite:

a. calculeaza_md5(cale_fisier) – citește un fișier în mod binar și returnează hash-ul MD5 corespunzător;
b. alege_fisier() – deschide o fereastră de selectare a fișierului și afișează calea acestuia;

c. verifica_hash() – compară hash-ul calculat cu cel introdus de utilizator și afișează un mesaj de confirmare sau avertizare.

Interfața grafică este realizată cu tkinter și include:

- un buton pentru alegerea fișierului,
- câmpuri pentru afișarea căii fișierului și introducerea hash-ului original,
- un câmp de afișare a hash-ului calculat,
- un buton de verificare,
- mesaje de rezultat afișate într-o fereastră pop-up (messagebox).

Structura codului este clară și modulară, permițând o întreținere și extindere facilă a aplicației. Toate acțiunile utilizatorului sunt gestionate în timp real, fără a necesita repornirea aplicației sau utilizarea liniei de comandă.

3.2 Prezentarea funcționării aplicației

Pentru a evidenția funcționarea aplicației, în această secțiune sunt prezentate capturi de ecran realizate în timpul utilizării.

Inițial, utilizatorul deschide aplicația și apasă pe butonul **„Alege fișier”**, ceea ce deschide o fereastră de tip File Dialog. După selectarea unui fișier .txt, aplicația citește automat conținutul și calculează hash-ul MD5, afișându-l într-o etichetă dedicată.

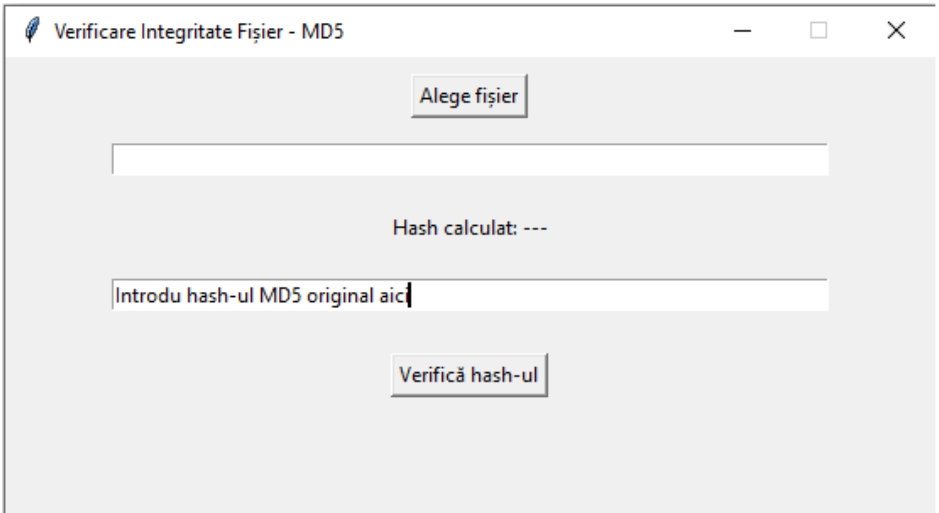

Fig.1, Captură de ecran cu aplicația

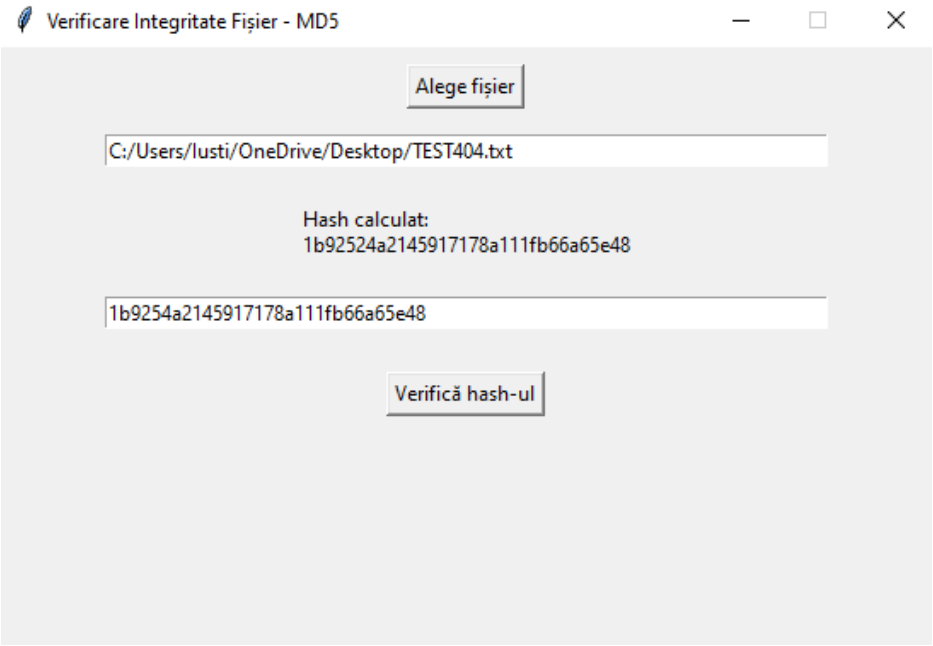

Fig.2, Captură de ecran cu aplicația în starea de după selectarea fișierului

În continuare, utilizatorul poate introduce manual hash-ul original al fișierului, oferit de producător sau sursa originală. După introducerea valorii, se apasă butonul **„Verifică hash-ul”**, moment în care aplicația compară cele două hash-uri.

Dacă valorile sunt identice, aplicația afișează un mesaj de confirmare: *„Fișierul este autentic (hash-ul coincide).”*

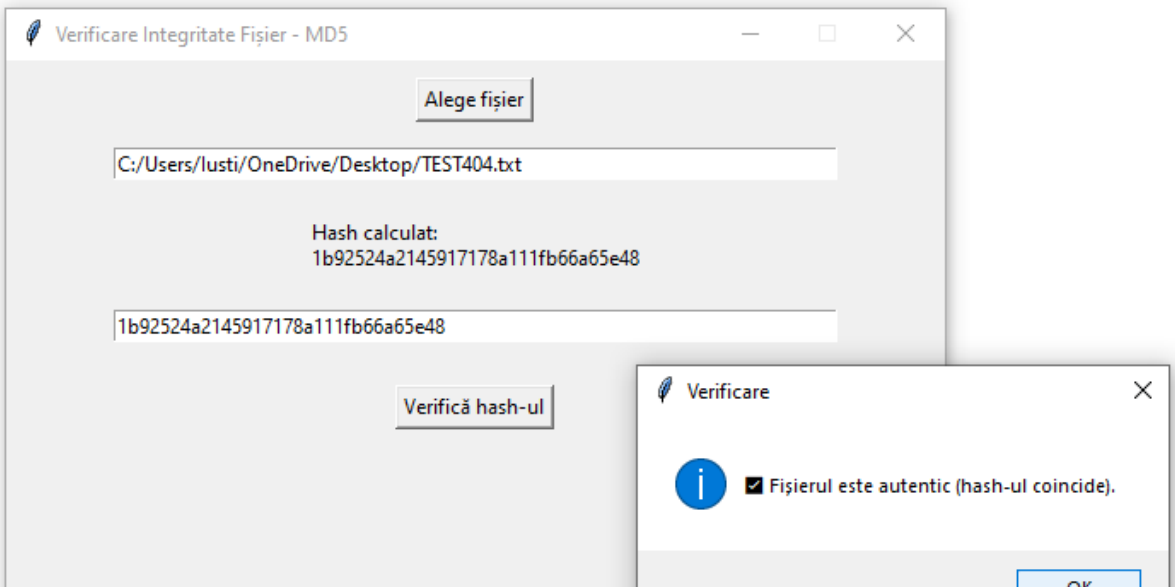

Fig.3, Captură de ecran cu mesajul de succes

Dacă valorile nu corespund, aplicația notifică utilizatorul cu un mesaj de eroare: *„Fișierul NU este autentic (hash-ul diferă).”*

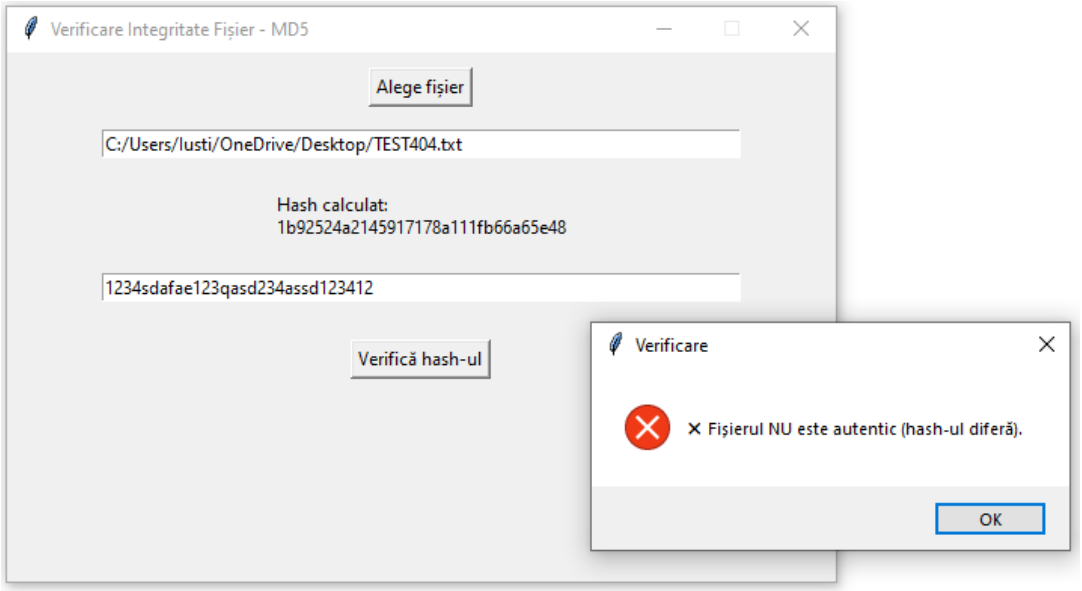

Fig.4, Captură cu mesajul de eșec

CONCLUZII

Proiectul prezentat demonstrează aplicarea practică a unui algoritm hash pentru verificarea integrității fișierelor, folosind un limbaj de programare accesibil. Aplicația realizată în Python reușește să integreze două concepte esențiale — hashingul MD5 și interacțiunea grafică — într-un mod simplu și eficient.

Lucrarea a oferit o oportunitate de înțelegere a funcționării algoritmului MD5, a avantajelor și limitărilor acestuia, precum și a modului în care poate fi folosit într-un scenariu real. Deși MD5 nu este potrivit pentru aplicații criptografice avansate, el rămâne un instrument valoros pentru verificări simple, rapide și clare.

Pe viitor, aplicația poate fi extinsă pentru a include suport pentru algoritmi mai avansați (precum SHA-256), verificarea multiplă a fișierelor sau salvarea hash-urilor într-un fișier extern.