Digital Forensics and Incident Response

HASH COLLISION DETECTOR

PES1UG22CS832 - R Divya Srujana
PES1UG21CS550 - Tabasum
PES1UG21CS527 - Sampreethi Y

Abstract

This project develops a tool for digital forensic investigations to generate, manage, and verify file hashes. The application uses a graphical interface built with Tkinter and an SQLite database for storing file information and hash values.

The Hash Collision Detector Project aims to develop a robust tool capable of detecting hash collisions in various cryptographic hash functions. Hash functions are fundamental cryptographic tools used to map input data of arbitrary size to fixed-size values, known as hashes. While these functions are designed to produce unique hashes for different inputs, collisions—where different inputs produce the same hash—can occur due to algorithmic limitations or malicious intent.

Problem or Application

The tool we developed addresses the need for secure file management through cryptographic hashing. It facilitates the detection of file tampering or duplication by comparing hash values generated from file content. The ability to verify integrity and detect collisions is crucial in environments where file authenticity and uniqueness are critical.

Research Highlight

This tool makes use of well-known cryptographic hashing algorithms (e.g., MD5, SHA-256) to ensure the integrity and uniqueness of files. By integrating these algorithms into a user-friendly GUI, it makes complex cryptographic operations accessible to users without technical expertise in cryptography.

Implementation Highlight

The core functionality is built using Python's Tkinter library for the GUI, enabling cross-platform compatibility. The application uses SQLite for persistent storage of file hashes, allowing for integrity checks over time. Hash values are generated using Python's hashlib, and file management is enhanced by features like adding and removing files dynamically from the user interface.

Analysis Highlight

The tool performs real-time hash generation and collision detection, providing instant feedback to the user. It also includes error handling to manage exceptions during file operations or hash calculations, ensuring the application remains stable under various conditions. The implementation of collision detection algorithms is optimized to minimize the computational overhead involved in comparing multiple files. This tool integrates metadata management for image files. The check_metadata function exemplifies this by allowing users to inspect EXIF data for supported image formats (PNG, JPG, JPEG, GIF). This feature is crucial for users who need to review or audit image metadata for authenticity or origin, further enhancing the application's utility in digital forensics. When a non-image file is selected, the application intelligently notifies the user that metadata viewing is unsupported, ensuring a user-friendly experience that prevents confusion and potential errors.

Conclusion

This tool significantly simplifies the process of file verification and integrity checking using cryptographic hashes. It provides a robust tool for users needing to manage and secure files effectively, ensuring data integrity and helping prevent malicious tampering or accidental duplication.

Problem Description

Why is this problem important?

Ensuring data integrity is crucial in digital forensics to provide reliable evidence in legal contexts. This tool automates and simplifies the process of verifying file integrity through hash comparisons.

What is your hypothesis and why are you doing this particular work?

Our hypothesis is that a user-friendly tool can significantly reduce the time and potential errors in manual hash verification and collision detection, thereby enhancing forensic investigations.

Implementation

What was implemented and how?

The tool is implemented in Python using Tkinter for the GUI, SQLite for database management, and hashlib for cryptographic hash generation. It allows users to add files, delete them, generate and view hash values, and check for hash collisions.

What are the major design points?

Modular design separating the GUI, hashing operations, exif tool module to display image's meta data ,and database management.

Use of SQLite to avoid dependency on external database servers.

Extensive hash algorithm support including MD5, SHA-1, and SHA-256.

What is the architecture of your tool?

The architecture includes:

Front-end: Tkinter-based GUI for interaction.

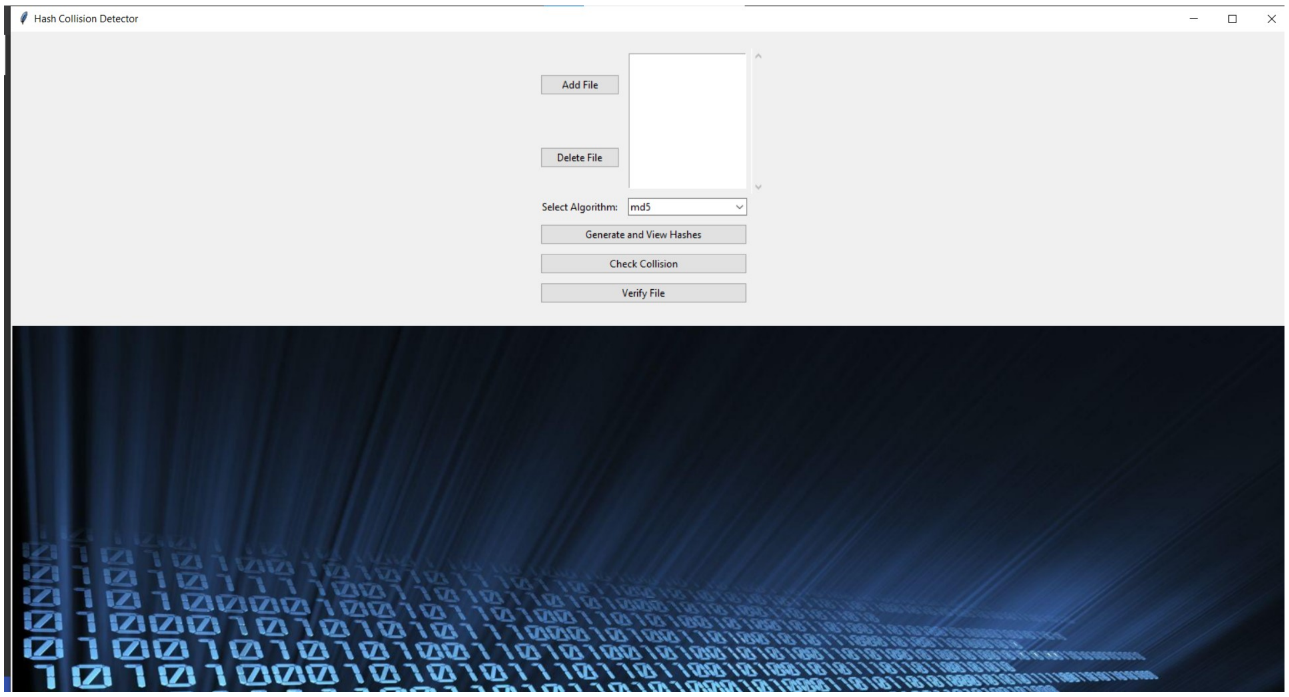
Back-end: SQLite database for storage and hashlib for hash computations.

Utilities: External integration with ExifTool for metadata extraction.

Results

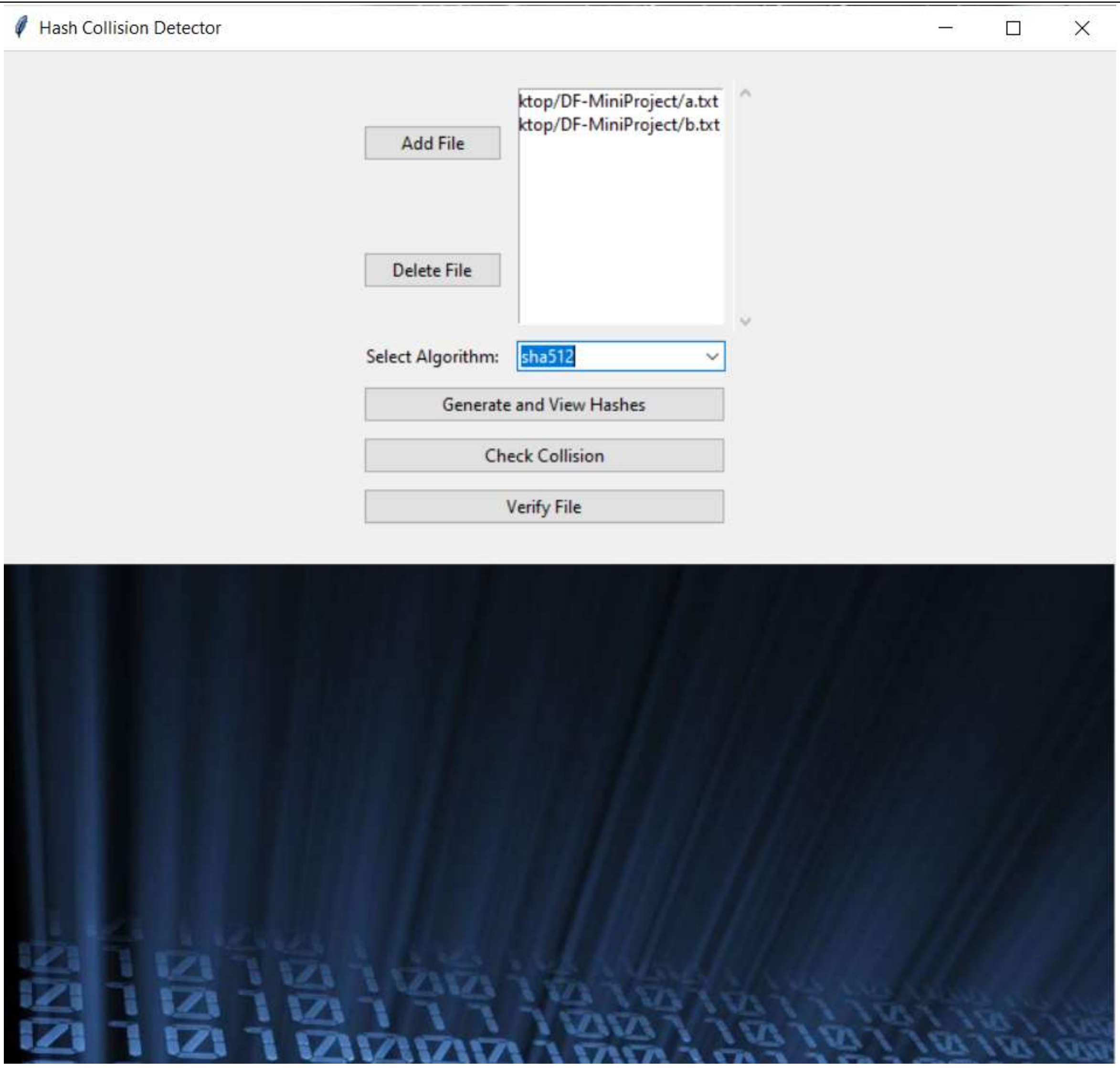
Displays data / results in a clear manner

The GUI clearly displays file paths, hash results, and integrity status. It provides visual feedback for actions like collision detection and hash verification.



this is the UI of our project it has 6 buttons , which performs different operations accordingly

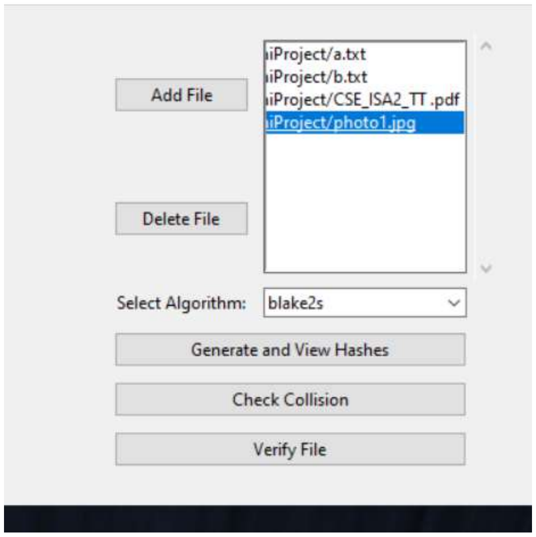
Add File : This button is used to add the files from the file system to our tool inorder to carry out further tasks.



ktop/DF-MiniProject/a.txt
ktop/DF-MiniProject/b.txt

Select Algorithm: sha512

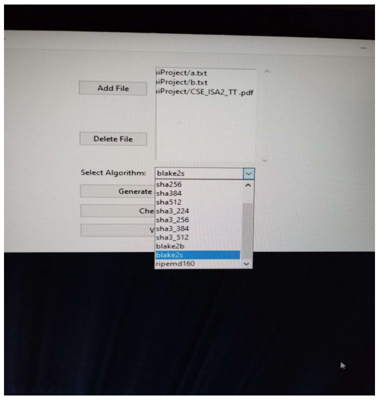
Delete File : This button is used to delete the files from the tool



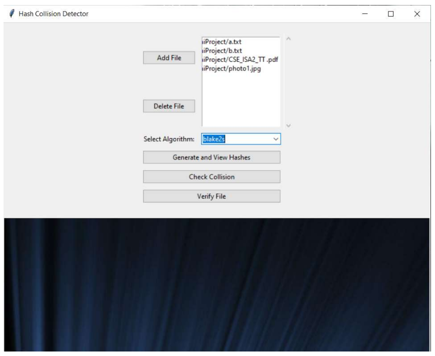
iProject/a.txt
iProject/b.txt
iProject/CSE_ISA2_TT.pdf
iProject/photo1.jpg

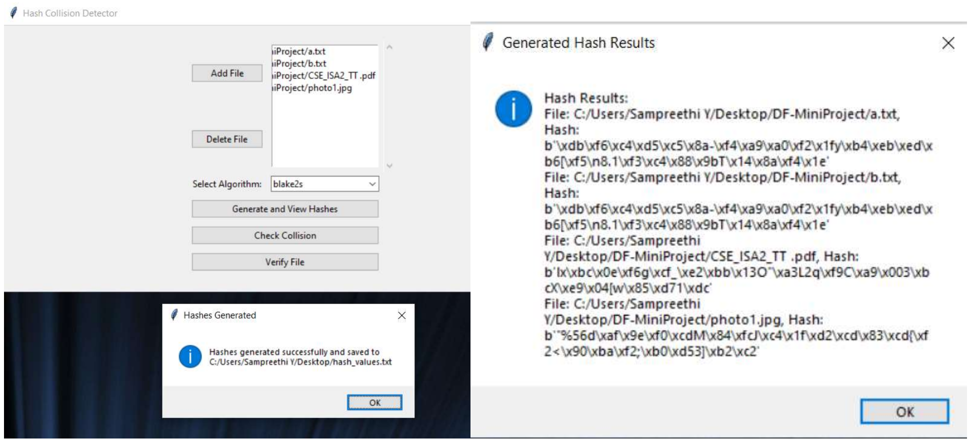
Select Algorithm: blake2s

Select algorithm : This drop down consists of several algorithms that we can choose to generate hash values for our files/file



Generate and View Hashes : after adding files to the tool we can generate and view the hash values of the multiple or single files.



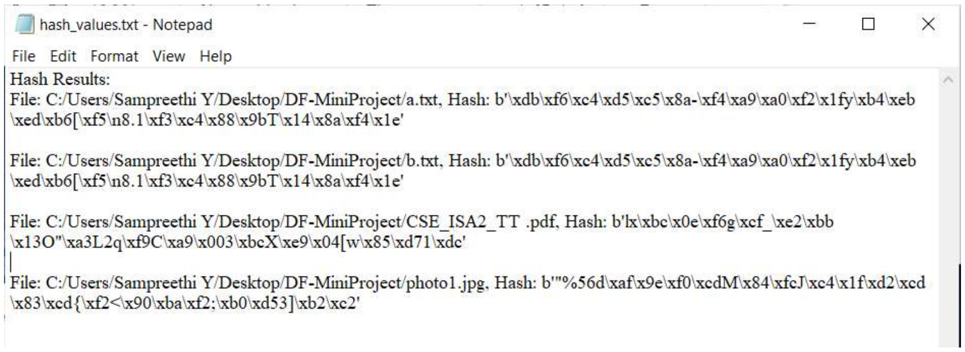


Hashes Generated

Hashes generated successfully and saved to C:/Users/Sampreethi Y/Desktop/hash_values.txt

Generated Hash Results

```
Hash Results:
File: C:/Users/Sampreethi Y/Desktop/DF-MiniProject/a.txt,
Hash:
b'\xdb\xf6\xc4\xd5\xc5\x8a-\xf4\xa9\xa0\xf2\x1fy\xb4\xeb\xed\x
b6[\xf5\n8.1\xf3\xc4\x88\x9bT\x14\x8a\xf4\x1e'
File: C:/Users/Sampreethi Y/Desktop/DF-MiniProject/b.txt,
Hash:
b'\xdb\xf6\xc4\xd5\xc5\x8a-\xf4\xa9\xa0\xf2\x1fy\xb4\xeb\xed\x
b6[\xf5\n8.1\xf3\xc4\x88\x9bT\x14\x8a\xf4\x1e'
File: C:/Users/Sampreethi
Y/Desktop/DF-MiniProject/CSE_ISA2_TT .pdf, Hash:
b'lx\xbc\x0e\xf6g\xcf_\xe2\xbb\x13O"\xa3L2q\xf9C\xa9\x003\xb
cX\xe9\x04[w\x85\xd71\xdc'
File: C:/Users/Sampreethi
Y/Desktop/DF-MiniProject/photo1.jpg, Hash:
b'"%56d\xaf\x9e\xf0\xcdM\x84\xfcJ\xc4\x1f\xd2\xcd\x83\xcd{\xf
2<\x90\xba\xf2;\xb0\xd53]\xb2\xc2'
```



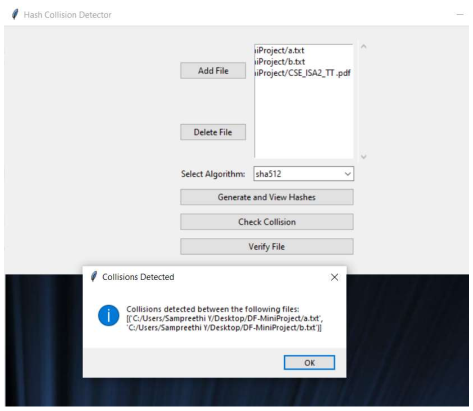
```
Hash Results:
File: C:/Users/Sampreethi Y/Desktop/DF-MiniProject/a.txt, Hash: b'\xdb\xf6\xc4\xd5\xc5\x8a-\xf4\xa9\xa0\xf2\x1fy\xb4\xeb
\xed\xb6[\xf5\n8.1\xf3\xc4\x88\x9bT\x14\x8a\xf4\x1e'

File: C:/Users/Sampreethi Y/Desktop/DF-MiniProject/b.txt, Hash: b'\xdb\xf6\xc4\xd5\xc5\x8a-\xf4\xa9\xa0\xf2\x1fy\xb4\xeb
\xed\xb6[\xf5\n8.1\xf3\xc4\x88\x9bT\x14\x8a\xf4\x1e'

File: C:/Users/Sampreethi Y/Desktop/DF-MiniProject/CSE_ISA2_TT .pdf, Hash: b'lx\xbc\x0e\xf6g\xcf_\xe2\xbb
\x13O"\xa3L2q\xf9C\xa9\x003\xbcX\xe9\x04[w\x85\xd71\xdc'

File: C:/Users/Sampreethi Y/Desktop/DF-MiniProject/photo1.jpg, Hash: b'"%56d\xaf\x9e\xf0\xcdM\x84\xfcJ\xc4\x1f\xd2\xcd
\x83\xcd{\xf2<\x90\xba\xf2;\xb0\xd53]\xb2\xc2'
```

Check collision: This feature in our tool helps us to identify collisions between the files based on hash values , it will take multiple files input and it will generate hash values with respect to particular algorithm selected and it will display whether the files are victim to collision.

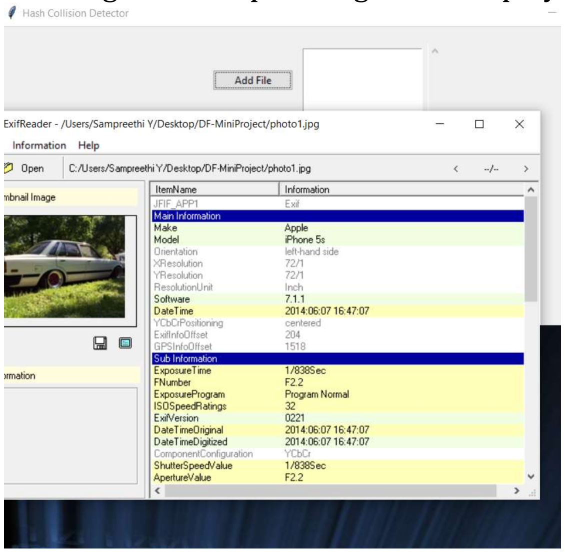


iProject/a.txt
iProject/b.txt
iProject/CSE_ISA2_TT .pdf

Select Algorithm: sha512

Collisions detected between the following files:
[('C:/Users/Sampreethi Y/Desktop/DF-MiniProject/a.txt', 'C:/Users/Sampreethi Y/Desktop/DF-MiniProject/b.txt')]

for image file on uploading it will display metadata as exif tool is integrated

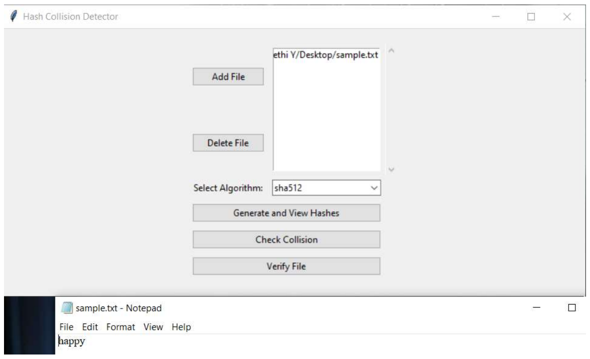


ExifReader - /Users/Sampreethi Y/Desktop/DF-MiniProject/photo1.jpg

C:/Users/Sampreethi Y/Desktop/DF-MiniProject/photo1.jpg

ItemName	Information
JFIF_APP1	Exif
Main Information	
Make	Apple
Model	iPhone 5s
Orientation	left-hand side
XResolution	72/1
YResolution	72/1
ResolutionUnit	Inch
Software	7.1.1
DateTime	2014:06:07 16:47:07
YCbCrPositioning	centered
ExifInfoOffset	204
GPSInfoOffset	1518
Sub Information	
ExposureTime	1/838Sec
FNumber	F2.2
ExposureProgram	Program Normal
ISOSpeedRatings	32
ExifVersion	0221
DateTimeOriginal	2014:06:07 16:47:07
DateTimeDigitized	2014:06:07 16:47:07
ComponentConfiguration	YCbCr
ShutterSpeedValue	1/838Sec
ApertureValue	F2.2

Verify file : This function takes only one file as an input at a time, when the user clicks on this button, he will be able to verify the integrity of the file selected, the original files hash value will be stored in the database, the integrity of the file is checked based on the hash value

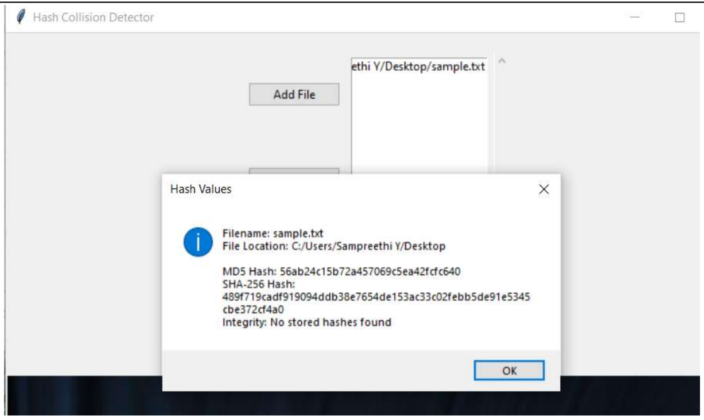


ethi Y/Desktop/sample.txt

Select Algorithm: sha512

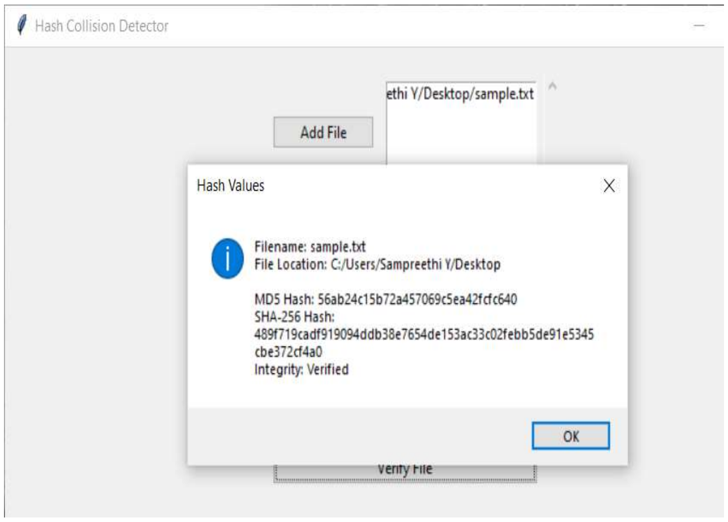
happy

as you can see in the above screenshot, we have uploaded a file named sample.txt to our tool, the content present in the file is happy



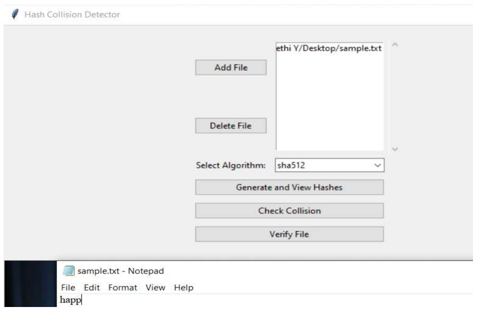
```
Filename: sample.txt
File Location: C:/Users/Sampreethi Y/Desktop

MD5 Hash: 56ab24c15b72a457069c5ea42fcfc640
SHA-256 Hash:
489f719cadf919094ddb38e7654de153ac33c02febb5de91e5345
cbe372cf4a0
Integrity: No stored hashes found
```



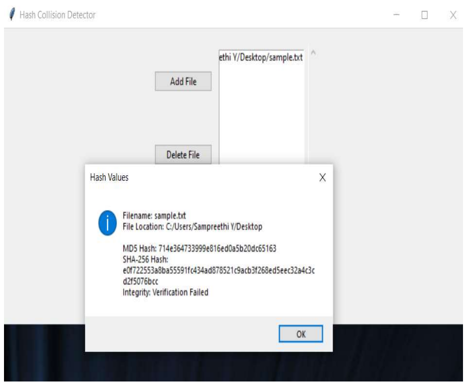
```
Filename: sample.txt
File Location: C:/Users/Sampreethi Y/Desktop

MD5 Hash: 56ab24c15b72a457069c5ea42fcfc640
SHA-256 Hash:
489f719cadf919094ddb38e7654de153ac33c02febb5de91e5345
cbe372cf4a0
Integrity: Verified
```



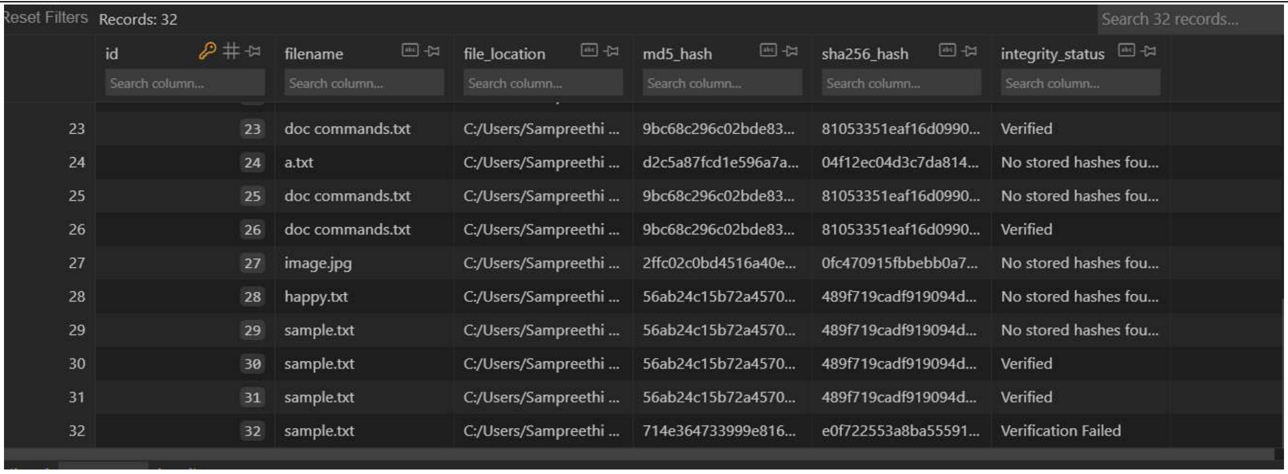
Select Algorithm: sha512

```
happ
```



```
Filename: sample.txt
File Location: C:/Users/Sampreethi Y/Desktop

MD5 Hash: 714e364733999e816ed0a5b20dc65163
SHA-256 Hash:
e0f722553a8ba55591fc434ad878521c9acb3f268ed5eec32a4c3c
d2f5076bcc
Integrity: Verification Failed
```



Reset Filters Records: 32

	id	filename	file_location	md5_hash	sha256_hash	integrity_status
23	23	doc commands.txt	C:/Users/Sampreethi ...	9bc68c296c02bde83...	81053351eaf16d0990...	Verified
24	24	a.txt	C:/Users/Sampreethi ...	d2c5a87fcd1e596a7a...	04f12ec04d3c7da814...	No stored hashes fou...
25	25	doc commands.txt	C:/Users/Sampreethi ...	9bc68c296c02bde83...	81053351eaf16d0990...	No stored hashes fou...
26	26	doc commands.txt	C:/Users/Sampreethi ...	9bc68c296c02bde83...	81053351eaf16d0990...	Verified
27	27	image.jpg	C:/Users/Sampreethi ...	2ffc02c0bd4516a40e...	0fc470915fbbebb0a7...	No stored hashes fou...
28	28	happy.txt	C:/Users/Sampreethi ...	56ab24c15b72a4570...	489f719cadf919094d...	No stored hashes fou...
29	29	sample.txt	C:/Users/Sampreethi ...	56ab24c15b72a4570...	489f719cadf919094d...	No stored hashes fou...
30	30	sample.txt	C:/Users/Sampreethi ...	56ab24c15b72a4570...	489f719cadf919094d...	Verified
31	31	sample.txt	C:/Users/Sampreethi ...	56ab24c15b72a4570...	489f719cadf919094d...	Verified
32	32	sample.txt	C:/Users/Sampreethi ...	714e364733999e816...	e0f722553a8ba55591...	Verification Failed

How the results address the original hypothesis

Results confirm that automating hash generation and verification speeds up the process and reduces errors, supporting our hypothesis.

Conclusion

What the experiment showed

The tool successfully automates the hash generation and verification process, proving to be effective in detecting hash collisions and verifying data integrity.

Gives a recommendation for future use/development/behavior based on the experimental results

Future development could include support for more complex forensic tasks, integration with cloud storage, and enhanced security features for enterprise-level applications.

Appendix

collaborated in a team of 3 to develop this tool, since this tool is done with respect to modular design each of us contributed to 3 different modules one to generate hash value and integrating with Exif tool for image meta data analysis , one to check collision and one person worked on file integrity verification and data base integration.
integrating the modules was challenging specially exif tool with the other modules,