Shamir's Secret Sharing Algorithm in Pixel-Based Image Encryption

I. Introduction

Secret sharing is a fundamental cryptographic term that deals with the secure sharing of a secret among several participants. It relies on the fact that only the stakeholders to whom the secret is distributed can recreate it, and that no unauthorized person can obtain any information about it [2]. The term secret sharing, first introduced by Shamir [5] and Blakley [4] in 1979, has since evolved into a critical paradigm for security with widespread use in applications in secure multi-party computing, threshold cryptography, and access control mechanisms [3]. A secret sharing scheme is essentially a dealer that distributes parts of a secret among a group according to an access structure; a rule that controls which people can recreate the secret. A threshold secret sharing system divides a secret into a large number of shares, such that any subset of at least t people can recreate the secret, but shares less than t cannot reveal any information about the original secret [3]. This property makes secret sharing extremely useful for cryptographic key management, distributed storage, and secure voting systems [3]. In addition, secret sharing offers both confidentiality and fault tolerance, since the loss of some shares does not prevent the secret from being reconstructed as long as the threshold is met [3]. However, the amount of shares is a significant obstacle to creating effective secret sharing schemes. These structures may not be suitable for large-scale applications, as they sometimes require shares that increase exponentially with the number of people [2]. Although linear secret sharing systems offer more efficient solutions, they may have limitations, especially in terms of computational efficiency and space complexity [2].

Security of secret sharing against fraud is another important issue. Malicious actors can manipulate the reconstruction process to deceive others, or an untrustworthy vendor can misdistribute shares in real-world situations [3]. Verifiable secret sharing (VSS) techniques have been introduced to mitigate such risks by saying that the integrity of the share can be checked before the secret is reconstructed [3]. This feature is important for distributed cryptographic protocols such as secure cloud storage, e-voting, and blockchain applications [3].

Today, the secret sharing paradigm has begun to evolve from a mere term for cryptographic key management to more complex uses. For example, secure multi-party computation (MPC) is an important application in which multiple stakeholders collaborate to compute a function based on their private inputs without disclosing them to each other [2]. The use of secret sharing is also common in threshold cryptography, which allows cryptographic operations to be distributed across multiple entities and ensures that security cannot be compromised by a single person or institution [2]. More recently, secret sharing has also been incorporated into multimedia cryptography, including encrypted image and audio sharing, secure IoT connectivity, and federated learning [3]. As a result, secret sharing has evolved from early conceptual frameworks to robust, adaptable systems capable of addressing today's security challenges. As this field of study evolves, initiatives to increase security and efficiency will strengthen the position of secret sharing in protecting digital data for various purposes.

II. Related Works

A secret s is represented in Shamir's secret sharing scheme as n different values called shares, so that when at least k shares of n shares are kept, the secret can be reconstructed. Fewer shares do not disclose any details about s. They refer to this as (n,k)-secret sharing. Polynomial interpolation based on any k shares can be used to recreate the secret s [5].

In this study conducted by Jaya Nirmala et al., it is mentioned that data outsourcing is a more cost-effective paradigm for the user, who does not need to purchase expensive hardware and software for data storage provided by third-party storage services, but before data outsourcing becomes applicable, the data provider must ensure that the data is secure, can execute queries on the data, and the results of the queries are also secure and invisible to the data provider, and for this, the performance of two secret sharing algorithms is compared. Shamir's secret sharing algorithm and Rabin's Information Distribution Algorithm (IDA) are implemented in a private cloud setup using the OpenStack Cloud framework. In the experimental setup, a data set containing 4 million records in a cloud of 6 virtual machines is divided into 3 parts with Shamir's method and each part is stored in these virtual machines. In this experiment, high security is provided since the original data cannot be recovered without providing the threshold. However, since it maintains the exact size of each data piece here, it increases the storage cost [6].

In this article, A.M. Ekşicioğlu and his colleagues present a secret-sharing based key transport protocol to overcome the limitations of using fixed symmetric or public-key cryptographic algorithms for message authentication. This protocol effectively combines the advantages of symmetric and public-key ciphers while eliminating the need for a password. Thanks to the fixed secret shares placed in advance to the receivers and the activator share sent during the broadcast, a different symmetric key is securely created for each transmission [7].

Doganay et al. (2008), within the scope of their studies on privacy-preserving data mining methods, developed an additive secret sharing based protocol to perform secure k-means clustering on vertically split data.
In the scenario where the data is vertically split, a local distance calculation is performed for each piece based on its data attributes and these values are shared with other pieces via the secret sharing mechanism. After the shared data is collected, the distances between the clusters are compared using secure multi-party computation techniques and each data point is assigned to the closest cluster.
The experimental results show that the proposed protocol provides more efficient results with lower communication costs when run on both synthetic and real data sets [8].

Tompa et al. suggested in this study that secret sharing methods can have a cheating issue. For instance, the cheaters can get the genuine secret if they submit fake shares during the reconstruction phase, causing the legitimate users to rebuild a fake secret. One effective tactic to offer security against such cheating is cheating identification.
The seller transmits each share si to a user Pi after splitting the secret s into n shares, s1, s2,..., sn, during the sharing phase. A group of m users that are more than the threshold k send their shares to recreate the secret during the reconstruction phase. To find cheaters, these m shares are subjected to a cheating identification algorithm. Let C be the group of users who this algorithm has determined to be cheating. Reconstructing the secret s from the shares of users not in set L will provide (s, C) if $(m - |C|) \geqslant k$, but C if $(m-|C|) < k$ [9].

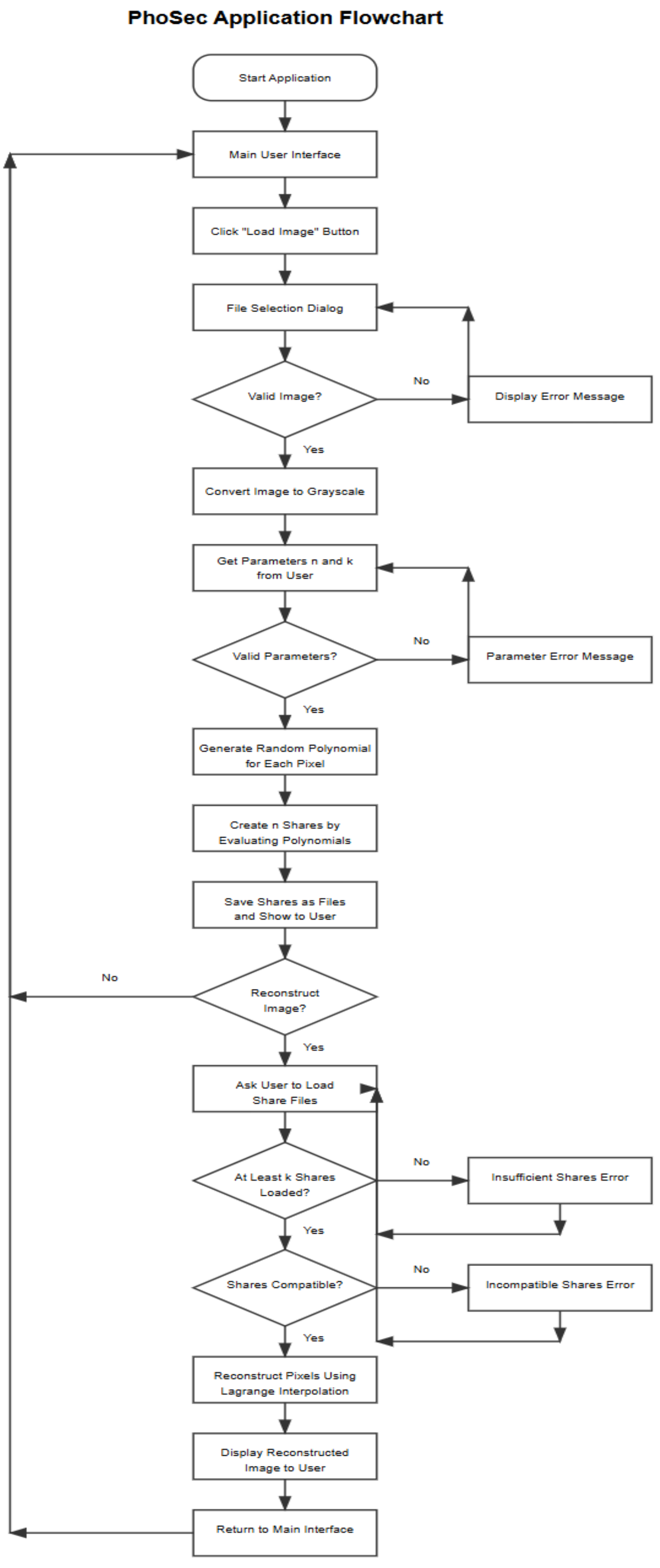
III. My Proposed Study

PhoSec App is an application based on sharing images using Shamir's Secret Sharing scheme, which is a cryptographic method. In this application, pixels from an image are encrypted using the (t,n) sharing scheme, resulting in n new images. By combining t of these n images, the original image can be restored.

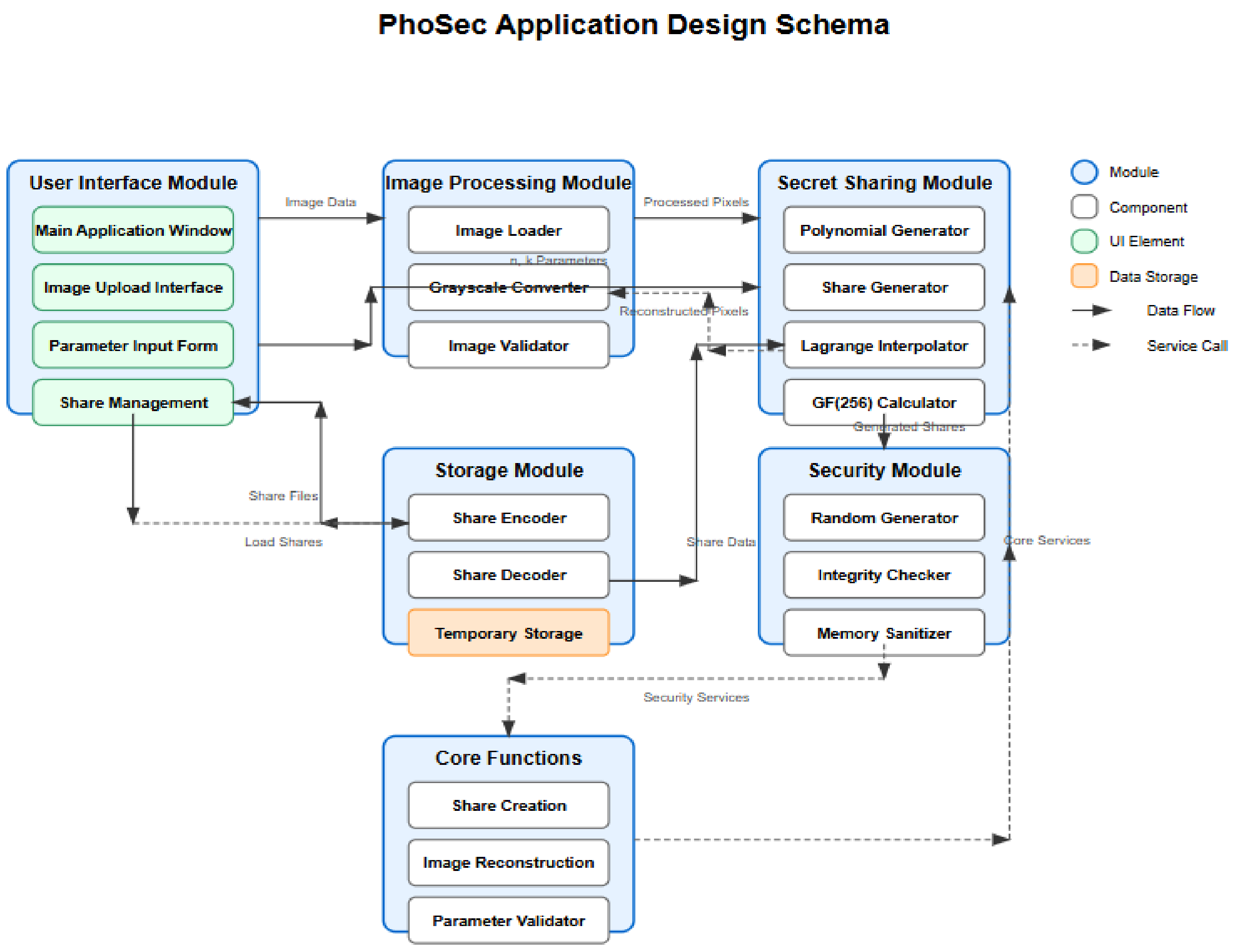
The application will basically work as follows: after checking whether an uploaded image is suitable, it is turned into gray and the relevant secret sharing parameters are taken from the user (n and k). After checking these parameters, a random polynomial is generated for each pixel in the image. Then, these polynomials are processed to create n shares. And finally, these shares will create a new encrypted image.

If the image is desired to be recreated after this part, the user is asked to upload the share files. It is checked whether at least k shares are uploaded. If the shares are compatible, the image is recreated using Lagrange interpolation.

In the test scenarios section, correct file loading and processing will be important. If the user loads a valid image file and enters the correct threshold (k) and total share (n) values, the system should be able to perform the expected operations successfully. In addition, incorrect file entry should be checked and the relevant error messages should be


PhoSec Application Flowchart
Start Application
Main User Interface
Click "Load Image" Button
File Selection Dialog
Valid Image?
No
Display Error Message
Yes
Convert Image to Grayscale
Get Parameters n and k from User
Valid Parameters?
No
Parameter Error Message
Yes
Generate Random Polynomial for Each Pixel
Create n Shares by Evaluating Polynomials
Save Shares as Files and Show to User
Reconstruct Image?
No
Yes
Ask User to Load Share Files
At Least k Shares Loaded?
No
Insufficient Shares Error
Yes
Shares Compatible?
No
Incompatible Shares Error
Yes
Reconstruct Pixels Using Lagrange Interpolation
Display Reconstructed Image to User
Return to Main Interface

displayed. The cases where the user enters the required parameters (k and n values) incompletely or incorrectly



should also be checked.

User Interface Module: This module contains the interface that allows the user to select an image, specify the n and k parameters, download the generated shares, and then upload these shares to recreate the original image.

Image Processing Module: This module loads images in different formats (JPG, PNG, BMP), converts color images to grayscale. It extracts pixel data and passes it to the Secret Sharing module, which converts pixel values to image format in the process of reconstructing the image.

Secret Sharing Module: Cryptographic operations are applied in this module. Shamir's Secret Sharing algorithm is implemented. It creates polynomials with random coefficients for each pixel, creates shares by dividing these polynomials and recreates the original pixel values using Lagrange interpolation in the image reconstruction part.

Storage Module: This module allows shares to be stored in the file system. It encodes shares into files with appropriate metadata (image size, n, k values, creation date) and then decodes these files back into usable data structures.

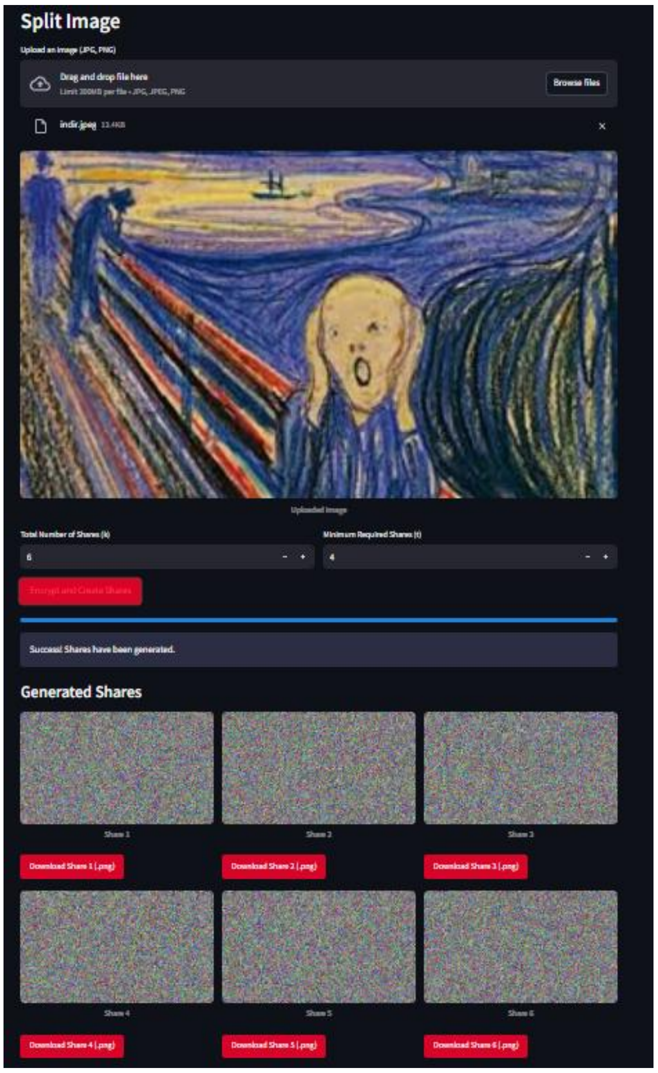
Security Module: This module can be used for secure random number generation, securely clearing data from memory after use, and can add an additional layer of encryption to shares.

Core Functions: This module is there for common functions that all other modules use. It includes basic helper functions for features like share creation, image recovery process, parameter validation, error handling, etc.

IV. IMPLEMENTATION

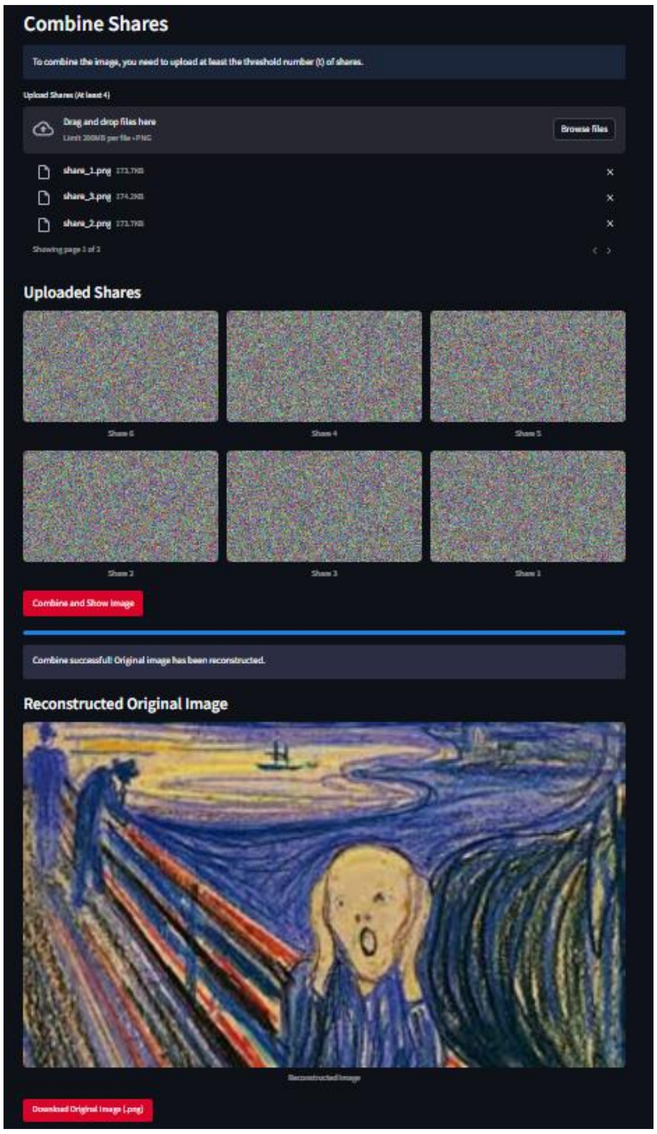
The interface was created using Streamlit. The application allows the user to upload an image and split it into a certain number of shares (k) using Shamir's Secret Sharing algorithm and determine the minimum number of shares needed to reconstruct the original image. It contains two main tabs: Split image (Encrypt) and Combine Shares (Decrypt).

The user uploads an image and enters the k and t values. The color channel values (RGB) of each pixel of the image are processed separately. Each color channel value (0-255) is encrypted using Shamir's Secret Sharing algorithm and divided into k shares. This process begins with generating a random polynomial of degree t-1 and setting the constant term of this polynomial (where it intersects the y-axis) equal to the current color value. Then, the corresponding y values (shares) are calculated for different x values from 1 to k using this polynomial. After repeating this process for all pixels and all color channels of the image, the y values of each share number are brought together to form k shares, which are usually meaningless and noisy. The original image cannot be reproduced without at least t of these share images.



Success! Shares have been generated.

In the Decryption section, the user uploads at least t (threshold value) of the previously created shares to the system. Each share contains an encrypted part of the color channels of each pixel of the original image and a unique index (x) value of that share. The algorithm processes the corresponding pixel locations in these loaded share images simultaneously. For each pixel, in order to recover the original values of the RGB color channels separately, the (index, encrypted color part) pairs in all loaded shares belonging to that pixel are added. Then, these (index, encrypted color part) pairs are applied for each color channel, which is the basis of Shamir's Secret Sharing algorithm. Given a sufficient number of share points, this method mathematically calculates the original polynomial passing through these points and the original color channel value, which is the constant term of the polynomial. After repeating this process for all color channels of each pixel of the image, the recovered R, G and B values are combined to form the original pixel, and the original image is successfully reconstructed by combining all pixels.



To combine the image, you need to upload at least the threshold number (t) of shares.

Combine successful! Original image has been reconstructed.

V. Test Results

Performance Metrics (Encryption):

Input Parameters	k (Total Shares)	t (Threshold)	Total Encryption Time (s)	SSS Core Encryption Time (s)	Pixels Processed per Second (pixels/s)	Total Time per Pixel (ms)	SSS Time per Pixel (ms)
Input Image Size: 0.01 MB Image Dimensions: 301 x 167 pixels Total Pixels: 50267	3	2	0.607	0.357	82.801	0.0121	0.0071
	6	6	1.526	1.115	32.945	0.0304	0.0222
	6	4	1.212	0.812	41.489	0.0241	0.0162
	8	6	1.832	1.325	27.431	0.0365	0.0264
	8	4	1.480	0.971	33.964	0.0294	0.0193
Input Image Size: 0.12 MB Image Dimensions: 1800 x 1017 pixels Total Pixels: 1,830,600	3	2	22.607	13.563	80.975	0.0123	0.0074
	6	6	58.217	42.826	31.445	0.0318	0.0234
	6	4	46.721	31.189	39.182	0.0255	0.0170
	8	6	71.000	49.606	25.783	0.0388	0.0271
	8	4	57.230	36.726	31.987	0.0313	0.0201

Observations and Conclusions for Encryption:

As can be seen in the table, as the number of pixels in the image increases, the total encryption time also increases significantly.
For example, for k=3, t=2, the number of pixels increases by about 36.4 times (from 50,267 to 1,830,600), while the total encryption time increases by about 37.2 times (from 0.607 seconds to 22.607 seconds). This shows that the total processing time of the algorithm is linearly proportional to the number of pixels for fixed values of k and t. SSS Core Encryption Time also showed a similar scaling.

As the t value increases in both images, both the total encryption time and the SSS core encryption time increase significantly. The degree of the polynomial used in Shamir's Secret Sharing (SSS) algorithm is t-1. So, higher t value means a higher degree polynomial. This means that more random coefficients are generated for each color channel of each pixel and more complex polynomial calculations are performed.

Since increasing k while keeping t constant means that the SSS algorithm needs to evaluate the generated polynomial at more x points, processing times increase as the k value increases.

The "SSS Core Encryption Time" accounts for a significant portion of the "Total Encryption Time" (between 60%-75% in the results). This indicates that cryptographic operations such as polynomial generation and evaluation take up a large portion of the overall processing load. That is, the most CPU-intensive part is the SSS algorithm; operations such as reading/writing the image or generating previews (especially for the image sizes in these tests) are less effective.

Both the "Total Time per Pixel" and "SSS Time per Pixel" values are similar for small and large images. This indicates that the processing cost per pixel is approximately constant overall. As k and t values increase, the processing time per pixel increases because more computations are performed for each pixel.

Performance Metrics (Decryption):

Image Dimensions	**Total Pixels**	**Number of Share Used**	**Input Shares Total Size (MB)**	**Total Decryption Time**	**SSS Core Decryption Time**	**Pixels Processed Per Second (pixels/s)**	**Total Processing Time per Pixel (ms)**	**SSS Processing Time Per Pixel (ms)**
301 x 167	50,267	4	0,68	3.282	2.495	15,316	0.0653	0.0496
301 x 167	50,267	6	1,02	5.325	4.204	9,439	0.1059	0.0836
1800 x 1017	1,830,600	4	24,51	129.529	100.554	14,133	0.0708	0.0549
1800 x 1017	1,830,600	6	36,76	214.478	171.765	8,535	0.1172	0.0938

Observations and Conclusions for Decryption:

As the number of shares used in the decryption process increases, both the "Total Decryption Time" and the "SSS Core Decryption Time" increase. This is valid for both image sizes. because in the SSS algorithm, decryption (Lagrange Interpolation) requires an addition and multiplication operation that includes as many terms as the number of shares used. More shares means more calculations.

As the number of pixels in the reconstructed image increases, the total decoding time increases significantly. Using 4 shares: the small image with 50,267 pixels was decoded in 3.282 seconds, while the large image with 1,830,600 pixels (approximately 36.4 times more pixels) was decoded in 129.529 seconds. The difference is approximately 39.5 times. This shows that the decoding time also scales roughly linearly with the number of pixels.

The "SSS Core Decryption Time" accounts for a very large portion of the "Total Decryption Time". That is, the main computational load in the decryption process is caused by the SSS core algorithms such as Lagrange Interpolation. Other operations such as reading, merging and converting the fragments to images take less time compared to the core cryptographic computations.

As the number of shares used increases, both the total processing time and the SSS processing time per pixel increase. When the same number of shares is used, the "SSS Processing Time Per Pixel" values for large and small images are quite close to each other (e.g. 0.0496 ms vs 0.0549 ms for 4 shares; 0.0836 ms vs 0.0938 ms for 6 shares). Here, it can be seen that the SSS computation cost per pixel does not depend on the number of pieces used, and the total time actually varies as a proportion of the number of pixels and the number of shares used.

References

[1] Krawczyk, H. (1993, August). Secret sharing made short. In Annual international cryptology conference (pp. 136-146). Berlin, Heidelberg Springer Berlin Heidelberg.

[2] Beimel, A. (2011). Secret-Sharing Schemes: A Survey. In: Chee, Y.M., et al. Coding and Cryptology. IWCC 2011. Lecture Notes in Computer Science, vol 6639. Springer, Berlin, Heidelberg.

[3] Chattopadhyay, A. K., Saha, S., Nag, A., & Nandi, S. (2024). Secret sharing: A comprehensive survey, taxonomy and applications. Computer Science Review, 51, 100608.

[4] Blakley, G.R.: Safeguarding cryptographic keys. In: Merwin, R.E., Zanca, J.T., Smith, M. (eds.) Proc. of the 1979 AFIPS National Computer Conference. AFIPS Conference Proceedings, vol. 48, pp. 313–317. AFIPS Press (1979)

[5] Shamir, A. (1979). How to share a secret. *Communications of the ACM*, *22*(11), 612-613.

[6] Nirmala, S. J., Bhanu, S. M. S., & Patel, A. A. (2012). A comparative study of the secret sharing algorithms for secure data in the cloud. *International Journal on Cloud Computing: Services and Architecture (IJCCSA)*, *2*(4), 63-71.

[7] Eskicioglu, A. M., & Delp, E. J. (2002). A key transport protocol based on secret sharing applications to information security. *IEEE Transactions on Consumer Electronics*, *48*(4), 816-824.

[8] Doganay, M. C., Pedersen, T. B., Saygin, Y., Savaş, E., & Levi, A. (2008, March). Distributed privacy preserving k-means clustering with additive secret sharing. In *Proceedings of the 2008 international workshop on Privacy and anonymity in information society* (pp. 3-11).

[9] Tompa, M., & Woll, H. (1989). How to share a secret with cheaters. *journal of Cryptology*, *1*(3), 133-138.