VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Course:

BCSE353E

INFORMATION SECURITY ANALYSIS AND AUDIT LAB

Year:

FALL INTER SEM 2022-23

Instructor:

Dr. Abirami S

Batch Members:

Ashutosh Bandooni (21BCE5363)

Samhruth Ananthanarayanan (21BCE1791)

Aindri Bajpai (21BCE1742)

Analytical Paper on Probabilistic Encryption Techniques

Introduction

Encryption is the process by which a readable message is converted to an unreadable form to prevent unauthorized parties from reading it. Decryption is the process of converting an encrypted message back to its original (readable) format. The original message is called the plaintext message. The encrypted message is called the ciphertext message.

Digital encryption algorithms work by manipulating the digital content of a plaintext message mathematically, using an encryption algorithm and a digital key to produce a ciphertext version of the message. The sender and recipient can communicate securely if the sender and recipient are the only ones who know the key.

Shared key encryption uses one key to encrypt and decrypt messages. For shared key cryptography to work, the sender and the recipient of a message must both have the same key, which they must keep secret from everybody else. The sender uses the shared key to encrypt a message, shown in the following figure, and then sends the ciphertext message to the recipient.

Public key encryption uses a pair of complementary keys (a public key and a private key) to encrypt and decrypt messages, as shown in the following figure. The two keys are mathematically related such that a message encoded with one key can only be decoded with the other key. Although a user's public and private keys are mathematically related, knowledge of a public key does not make it possible to calculate the corresponding private key.

Fully homomorphic encryption (FHE) is a technique that allows computations on encrypted data without the need for decryption and it provides privacy in various applications such as privacy-preserving cloud computing. In this article, we attempt to implement probabilistic encryption schemes.

Probabilistic encryption is a fundamental area of research in the field of cryptography, aiming to provide secure and efficient mechanisms for protecting

sensitive information. In this paper, we explore two varying approaches and cases where probabilistic encryption can be used.

The first approach leverages the base-n representation method, a widely used technique in encryption. This approach involves converting plaintext into a secure and unique base-n representation. The conversion process begins by generating a random base-n value, where 'n' represents the desired base for the conversion. Each character of the plaintext is then transformed into its corresponding numerical value using Unicode code points. Repeated divisions are performed to obtain a list of remainders, representing the digits in the base-n system. These remainders are mapped to characters in a given alphabet string, resulting in the creation of the final encrypted message. The base-n representation method provides a straightforward and easily implementable encryption scheme that transforms the original text into a different character set, ensuring confidentiality and offering a large number of possible representations for each character.

The second approach explores the realm of Fully Homomorphic Encryption (FHE), a powerful technique that allows for computation on encrypted data, providing end-to-end security. In this paper, we present a basic version of FHE, integrating aspects of a weather API to enhance the encryption process with randomness and weather-dependent key sizes. The FHE implementation includes methods for key generation, encryption, and decryption. The makekey method generates a public-private key pair, while the encrypt method scales the plaintext value, adds noise using random polynomials, and performs polynomial operations on the public key components to produce the ciphertext. The decrypt method utilizes polynomial operations and the private key to remove the noise from the ciphertext and recover the original plaintext. By incorporating weather-dependent key sizes and moduli based on the weather API data, the encryption process gains an additional layer of randomness, enhancing the security of the encryption scheme.

Through a comprehensive analysis of these two methods, including implementation details, performance evaluations, and security considerations, we aim to contribute to the advancement of probabilistic encryption techniques. The evaluation of both approaches considers factors such as encryption strength, computational efficiency, resistance against cryptographic attacks, and

suitability for different use cases. By examining the strengths and limitations of the base-n representation method and the basic FHE implementation, we provide insights into the practical implications and potential applications of these approaches in securing sensitive information.

Literature Survey

Type	Authors	Title	Tools	Algorithms	Advantages	Disadvantages
Research Paper	Saraf, Jagtap, Mishra	Text and image encryption decryption using advanced encryption standard	Composer Studio in C, Code Block Chaining and PCKS in Java, TMS320C6713 DSP processor	Basic audio processing for multichannel systems, Image processing.	Relatively simple in terms of implementation	Massive overheads in encryption
Research Paper	Mart, Ozturk, Savas	Design and Implementation of Encryption/Decryption Architectures for BFV Homomorphic Encryption Scheme	XILINX VIRTEX-7 FPGA, Simple Encrypted Arithmetic Library	Fully Homomorphic Encryption based system	Little to no overheads in operation	Needs proper lattice to implement, slower due to lack of idealised lattice
Conceptual Paper	Jiang, Zhao, Xue, Tang, Qiu	Physical secure optical communication based on	ADC/DAC, intensifier, intensity modulator	(No algorithm, physical layer only)	Can support almost any level of encryption using channel overlap	Vulnerable to direct tapping and filtering, additional noise can jeopardize

		private chaotic spectral phase encryption /decryption				messages, low bandwidth
Research Paper	Rasras, Abuzalata, Alqadi, Al-Azzeh, Jaber	Comparative Analysis of Color Image Encryption-Decryption Methods Based on Matrix Manipulation	MATLAB	Generation of private key to decode the matrices, reshaping of 3D matrix to 2D	Relatively quick to encode and decode, almost no way to decode without key	Destructive encoding of image, always has overheads
Research Paper	Banik, Bogdanov, Regazzoni	Atomic-AES: A Compact Implementation of the AES Encryption/Decryption Core	ICs for Serial Plaintext and Key operations along with registers	(Hardware implementation)	Drastically reduces the hardware overhead of standard AES	Still bears all the weaknesses that AES such as security margin and block ciphers

Research Paper	Gao, Wang, He, Dong	Fault-Tolerant Consensus Control for Multiagent Systems: An Encryption-Decryption Scheme	(Theoretical only)	Agentwise PDR Coding Algorithm	Can allow multiple users to simultaneously encrypt	Very difficult to practically implement
Research Paper	R. L. Rivest, A. Shamir, and L. Adleman	A Method for Obtaining Digital Signatures and Public Key Cryptosystems	Cryptography, Mathematics	Public Key Cryptosystem	An established method presented with a novel implementation to make it much better.	Complexity of the implementation increased hence increasing the risks.
Review Paper	Nitin Jirwan, Ajay Singh, Dr. Sandip Vijay	Review and Analysis of Cryptography Techniques	Analysis Tools	RSA (Rivest Shamir and Adleman), Diffie-Hellman, DSA (Digital Signature Algorithm), ECC	Review and Analysis of Various Algorithms	All algorithms may not work for all situations hence comparing them might not yield perfect comparisons

				(Elliptic curve cryptography)		
Research Paper	Bedir Yousif; Fahmi Khalifa ; Ahmed Makram ; Ali Takieldeen	A novel image encryption /decryption scheme based on integrating multiple chaotic maps	Chaotic Maps, Logistic Maps	Various Cryptographic Techniques, Covariance and Correlation	Multiple Chaotic Maps to get a new map that works well.	Selection was done with lowest value for the correlation factor because the smaller value of correlation has an impression of good encryption.
Conceptual Paper	Prakash Kuppuswamy, Saeed Q. Y. Al-Khalidi	Hybrid Encryption/Decryption Technique Using New Public Key and Symmetric Key Algorithm	Cryptography Theory, Mathematics	Public Key Cryptography based on simple symmetric algorithm	Proposes a hybrid encryption model for better encryption services	Complication and time of the algorithm increases significantly with comparison to the original algorithms

Technical Paper	Himanshu Gupta and Vinod Kumar Sharma	A New Concept in Modern Cryptography	Cryptography Theory and Techniques, Mathematics	Multiphase Encryption, Multiple Encryption	Many encryption techniques are combined to make a single strong, 'multiphase' encryption technique	Decrypting this kind of encryption might cost more time that necessary, this method is not really implemented properly currently.
Scholarly Article	Ralph C. Merkle	A Digital Signature Based on a Conventional Encryption Function	Cryptography Theory	One way functions to digitally sign and encrypt algorithms	Highly simplified implementation without any complicated algorithms	Cannot account for more complex use-cases
Scholarly Article	Whitefield Diffie, Martin E. Hellman	New Directions in Cryptography	Basic reviewing of cryptography	Public key cryptosystems and trap door cryptosystems	Reduce costs, overhead, over-complex and uneconomical operations.	Can still have security concerns
Review Paper	Diffie, W	The first ten years of public-key cryptography.	Mathematical functions related to encryption and decryption	RSA, McElise Coding Scheme	Solved problems related to key distribution and signature variations	Turned out to be difficult to formulate for several years, and is still expanding.

Review Paper	J. Chandra shekhar et. Al.	A Comprehe nsive Study on Digital Signature	Cryptography theory and mathematical functions	Hashing, Public Key	Digital Signatures are important to ensure integrity of information exchanged among two parties	Relatively unsafe when used without another encryption algorithm

CITATIONS

1. Saraf, K. R., Jagtap, V. P., & Mishra, A. K. (2014). Text and image encryption decryption using advanced encryption standards. International Journal of Emerging Trends & Technology in Computer Science (IJETTCS), 3(3), 118-126.
2. Mert, A. C., Öztürk, E., & Savaş, E. (2019). Design and implementation of encryption/decryption architectures for BFV homomorphic encryption scheme. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 28(2), 353-362.
3. Jiang, N., Zhao, A., Xue, C., Tang, J., & Qiu, K. (2019). Physical secure optical communication based on private chaotic spectral phase encryption/decryption. Optics letters, 44(7), 1536-1539.
4. Rasras, R. J., Abuzalata, M., Alqadi, Z., Al-Azzeh, J., & Jaber, Q. (2019). Comparative Analysis of Color Image Encryption-Decryption Methods Based on Matrix Manipulation. International Journal of Computer Science and Mobile Computing, 8(3), 14-26.
5. Banik, S., Bogdanov, A., & Regazzoni, F. (2016). Atomic-AES: A compact implementation of the AES encryption/decryption core. In Progress in Cryptology–INDOCRYPT 2016: 17th International Conference on Cryptology in India, Kolkata, India, December 11-14, 2016, Proceedings 17 (pp. 173-190). Springer International Publishing.
6. Gao, C., Wang, Z., He, X., & Dong, H. (2021). Fault-tolerant consensus control for multiagent systems: An encryption-decryption scheme. IEEE Transactions on Automatic Control, 67(5), 2560-2567.
7. Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2), 120-126.
8. Jirwan, N., Singh, A., & Vijay, S. (2013). Review and analysis of cryptography techniques. International Journal of Scientific & Engineering Research, 4(3), 1-6.
9. Yousif, B., Khalifa, F., Makram, A., & Takieldeen, A. (2020). A novel image encryption/decryption scheme based on integrating multiple chaotic maps. AIP Advances, 10(7).
10. Kuppuswamy, P., & Al-Khalidi, S. Q. (2014). Hybrid encryption/decryption technique using new public key and symmetric key algorithm. International Journal of Information and Computer Security, 6(4), 372-382.
11. Gupta, H., & Sharma, V. K. (2013). Multiphase encryption: A new concept in modern cryptography. International Journal of Computer Theory and Engineering, 5(4), 638.
12. Merkle, R. C. (1987, August). A digital signature based on a conventional encryption function. In Conference on the theory and application of cryptographic techniques (pp. 369-378). Berlin, Heidelberg: Springer Berlin Heidelberg.
13. Diffie, W., & Hellman, M. E. (2022). New directions in cryptography. In Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman (pp. 365-390).
14. Diffie, W. (1988). The first ten years of public-key cryptography. Proceedings of the IEEE, 76(5), 560-577.
15. Chandrashekhara, J., Anu, V. B., Prabhavathi, H., & Ramya, B. R. (2021, May 20). A Comprehensive Study on Digital Signature. International Journal of Innovative Research in Computer Science & Technology (IJIRCST), 9(3), 43-47

Methodology

Overview Diagram

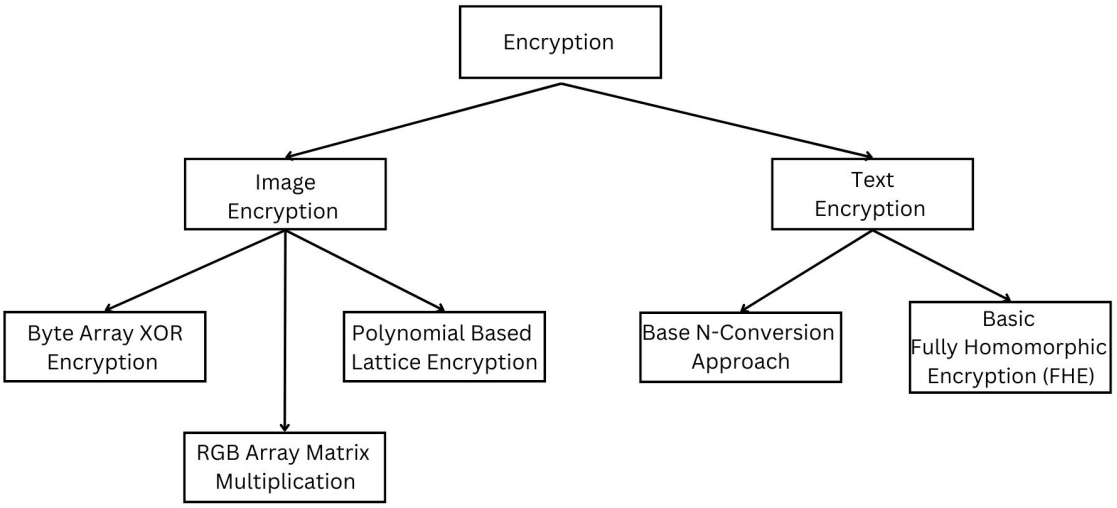


Image Encryption

The next implementation is of three distinct approaches for image encryption, each utilizing different techniques to ensure the confidentiality and integrity of the encrypted images. The base method we are following involves working with images represented in array form.

The sample image used for encryption testing is given below:

Byte Array XOR Encryption:

In the first approach, the image is converted into a byte array representation. We perform encryption by XOR-ing the byte array with a secret key. This process changes the codecs of the image, providing a form of encryption. By altering the values of the byte array through the XOR operation, we achieve a modified representation that conceals the original image content.

RGB Array Matrix Multiplication:

The second approach involves using the image as an RGB array or an array of pixel values. Encryption is performed by altering the RGB values through matrix multiplication. By multiplying the RGB array with a predetermined encryption matrix, we transform the pixel values, effectively scrambling the image. This method of encryption is relatively quick in comparison to more complex methods and still adds a few layers of security to the image itself by virtually turning it into noise. This approach leverages matrix operations to achieve encryption while preserving the image's visual characteristics.

There is also an added layer of security with the fact that the private key is just an integer index that gives the city to be used for the weather itself. The decryption matrix can be generated locally itself to allow for further layers of security.

Polynomial-based Lattice Encryption:

The third approach explores the use of polynomials within a lattice-based encryption scheme. Similar to the previous method, the image is treated as an array of pixel values. In this approach, we alter the encryption keys using polynomials and encrypt the image using a simple lattice structure. To enhance security, we expand the key size from 32 bits to 128 bits. The lattice structure incorporates the concept of Ring Learning With Error (RLWE), which introduces random errors to account for potential alteration of bits and data loss during encryption. By applying RLWE and leveraging the polynomial-based encryption within the lattice, we achieve enhanced security and resilience against attacks.

The lattice however does turn the image into ciphertext combinations by stuffing bytes and correcting values as it goes through the ring.

Assuming we can denote the inner product of 2 elements by the following format:

$$\langle a, b \rangle = \sum ai \cdot bi \ (modq)$$

Use the Regev Learning with Error model, for any integer s, assuming that q has a value of 2 or higher and m has a value of 1 or higher, we can see that.

$$\langle s, a1 \rangle + e1 = b1(modq)$$

$$\langle s, a2 \rangle + e2 = b2(modq)$$

$\langle s, a3 \rangle + e3 = b3(mod q)$

$\langle s, a4 \rangle + e4 = b4(mod q)$

. . .

$\langle s, am \rangle + em = bm(mod q)$

This ensures that the value s is relatively difficult to extract due to the fact that its value is now spread across an entire ring $(Z \% q)/(x^n + 1)$

For all three approaches, we consider both 8-bit and 32-bit image values, allowing for flexibility in encryption. Additionally, we incorporate random variables and pseudo-randomness to increase the probability density for each encryption operation. This randomness enhances the security and reduces predictability, making the encrypted images more resistant to cryptographic attacks.

Text Encryption:

We implemented the said idea in two ways, first within text encryption. Where we proposed two distinct approaches for text encryption, each employing different techniques to ensure the confidentiality and integrity of the encrypted messages. The first approach focuses on converting the plaintext into its base-n representation using ASCII values, while the second approach utilizes polynomials within a lattice-based framework for encryption.

Base-n Representation Approach:

The first approach involves converting the plaintext message into its base-n representation. To achieve this, we begin by generating a random base-n value, where 'n' represents the desired base for the conversion. The base-n value is then transformed into an intermediate decimal representation using mathematical operations.

Next, we apply a repeated division process to obtain the base-n digits. By dividing the intermediate decimal representation by the base-n value and appending the remainders, we construct a string of characters from the given alphabet. This resulting string forms the encrypted message, securing the plaintext information using the base-n encoding.

The base-n representation approach provides a straightforward and easily implementable method for text encryption, offering a certain level of confidentiality through the transformation of the plaintext into a different character set.

Polynomial-based Lattice Encryption Approach:

The second approach involves leveraging polynomials within a lattice-based encryption scheme. In this method, the encryption keys are altered using polynomials and a simple lattice structure. To enhance the encryption process, we expand the key size from 32 bits to 64 bits, allowing for increased security and improved resilience against potential attacks.

To construct the lattice structure, we employ the concept of Ring Learning With Error (RLWE). RLWE introduces random errors into the lattice-based encryption scheme, making it more resistant to attacks that exploit information leakage or structural vulnerabilities.

Within this lattice framework, the encryption process utilizes the polynomial-based keys. By applying cryptographic operations on the plaintext message and the lattice, we create a ciphertext that obscures the original message's content. The lattice structure and RLWE ensure that even if alterations or data loss occur during encryption, the decryption process can effectively recover the original plaintext with minimal loss.

The polynomial-based lattice encryption approach offers enhanced security and resilience, thanks to the utilization of RLWE and the larger key size. This method provides improved protection against cryptographic attacks and potential vulnerabilities inherent in simpler encryption schemes.

To evaluate the performance and security of both approaches, we conducted comprehensive testing and analysis. This involved measuring various metrics such as encryption and decryption speed, memory usage, and resistance to known cryptographic attacks. Additionally, we compared the performance and security characteristics of both approaches to determine their advantages and suitability for different encryption scenarios.

Implementation

Image Encryption

The proposed image encryption methods were implemented using the Python programming language, utilizing several specific libraries, including numpy, Python Image Library (PIL), Random, and Matplotlib. These libraries provide essential functionalities for handling arrays, image manipulation, random number generation, and visualization.

Byte Array XOR Encryption:

For the first approach, the implementation involved using the bytearray function in Python to convert the image into its binary codecs, represented as a byte array. This conversion allowed for easy manipulation of the image data. The byte array was then subjected to XOR operations with a secret key, altering the values of the array and providing a form of encryption. The implementation utilized the numpy library for efficient handling and manipulation of the byte array.

```
Enter a numeric key: 25
Byte array before encryption: bytearray(b'\xff\xd8\xff\xe0\x00\x10JFIF')
Byte array after encryption: bytearray(b'\xe6\xc1\xe6\xf9\x19\tS_P_')
```

```
Enter the same numeric key: 25
Byte array before decryption: bytearray(b'\xe6\xc1\xe6\xf9\x19\tS_P_')
Byte array after decryption: bytearray(b'\xff\xd8\xff\xe0\x00\x10JFIF')
```

RGB Array Matrix Multiplication:

The implementation of the second approach used the image as an RGB array, treating it as a matrix. To generate the encryption matrix, we incorporated nine variables obtained from the weather API. These variables were acquired through API calls, generated from a list of cities. The choice of the city was left to the inbuilt random function in python, creating a layer of pseudorandomness above the random variables. The weather data, combined with computer-generated randomness, ensured the generation of a unique encryption matrix with a check for singularity as well. The matrix multiplication operation was performed using

numpy to modify the RGB values of the image. The implementation further utilized the PIL library for image loading and visualization, allowing for a comprehensive evaluation of the encryption results.

```
The public key (city index) is: 58
The city of choice is: Budapest
The matrix key is:
 [[     19      47     800]
 [    298     299     294]
 [      0 2009313     200]]
The secret key (matrix inverse) is:
 [[-1.26268732e-03  3.43621161e-03 -4.81796996e-07]
 [-1.27406363e-07  8.12322450e-09  4.97684313e-07]
 [ 1.27999631e-03 -8.16105029e-05 -1.77962747e-08]]
Sample pixel before encryption: [114 103  83]
Encryption Time: 7.398414850234985
Decryption Time: 2.3614118099212646
Sample pixel post encryption: [113 103  82]
```

Polynomial-based Lattice Encryption:

The third approach involved converting the RGB array into arrays of integers to be encrypted. Additionally, byte stuffing was employed within the lattice structure to simplify the transmission of the encrypted image. The implementation utilized the Random library to introduce random coefficients into the polynomials that make up the lattice structure. The encryption process also made sure to use Ring Learning With Error to let the image be processed as securely as possible and to minimize losses in data by splitting up the ciphertext in its integral form to allow. The numpy library was employed to handle the arrays and perform necessary operations.

This method however does not allow for visualisation as the pixel values get split into polynomial ciphertexts with several different values for each section of the lattice itself. Instead, the data has to be decrypted using the secret key generated by the polynomials to be accessible as an image.

```
Params 4096 32768 64
Original Data:
[114 103  83]
[113 102  82]
[112 101  81]
[111 100  80]
[109  98  78]
[108  97  77]
[105  97  78]
[104  96  77]
[111 103  84]
[114 106  87]
Encrypted Data:

Pixel 1
Component 1:
 (array([14206, 27529, 24055, ..., 18934,  7506,  2028], dtype=int64), array([21103,  6885, 19850, ..., 11826, 30445, 30650], dtype=int64)) Component 2:
 (array([25760, 29896,  9334, ..., 32020, 32201,  8327], dtype=int64), array([  192, 15454, 29318, ...,  7904, 23622,  3480], dtype=int64)) Component 3:
 (array([ 7636,  7694,  2399, ...,   848, 18375,  1921], dtype=int64), array([29594, 13943, 16271, ..., 13079, 16600, 32091], dtype=int64))

Pixel 2
Component 1:
 (array([12212, 29210,  4628, ..., 13703, 32269, 23439], dtype=int64), array([ 4689,  9599, 28356, ..., 15264, 18815,  2279], dtype=int64)) Component 2:
 (array([19905, 27905, 25922, ..., 14640, 30281,  8001], dtype=int64), array([  454,  4829, 16704, ..., 30365, 32224, 22482], dtype=int64)) Component 3:
 (array([27366,  6225, 12353, ..., 28936, 24694, 20957], dtype=int64), array([ 4047, 16239,  9070, ..., 26523, 14924, 18586], dtype=int64))

Pixel 3
Component 1:
 (array([ 1670, 16128, 10102, ..., 30973,  6448, 14333], dtype=int64), array([32681,  8616, 26081, ..., 11286,  3221,  7643], dtype=int64)) Component 2:
 (array([26383,  3765, 17483, ..., 29099, 12665,  7440], dtype=int64), array([ 1641, 23859, 15414, ..., 19623, 30038, 15448], dtype=int64)) Component 3:
 (array([32206, 18695, 20119, ...,   208,  8769, 12433], dtype=int64), array([25002, 17255, 20480, ..., 17841, 23433, 15372], dtype=int64))

Pixel 4
Component 1:
 (array([19084, 16495, 22188, ..., 23733, 31501, 32367], dtype=int64), array([13333, 29771, 30915, ...,  6824,  4711, 26767], dtype=int64)) Component 2:
 (array([14170, 13259,  1714, ...,  7077, 25827,  1830], dtype=int64), array([11387,  6043, 13058, ..., 22675, 30256,  5276], dtype=int64)) Component 3:
 (array([16350, 26699,  2908, ...,  4721, 24966, 12254], dtype=int64), array([28519, 14008, 20631, ...,  7880, 13338, 28040], dtype=int64))

Pixel 5
Component 1:
 (array([ 6502, 27175,  3789, ...,  2187, 27672,  8571], dtype=int64), array([30607,  4330, 31629, ..., 17747,   284, 19324], dtype=int64)) Component 2:
 (array([32622,  6726,  8996, ...,  9118, 29754, 21485], dtype=int64), array([ 2040, 28545, 25574, ..., 25428, 17733,  5311], dtype=int64)) Component 3:
 (array([ 7337, 17591, 21158, ..., 15347,    33,  1431], dtype=int64), array([18401, 32747, 10715, ..., 14396, 26292, 31690], dtype=int64))
```

```
Pixel 6
Component 1:
 (array([18793,  4437,  5360, ...,  4758, 24430, 13266], dtype=int64), array([ 6252,  1143,  3354, ..., 10007, 31057, 30975], dtype=int64)) Component 2:
 (array([10154,  1465,  4130, ...,   745, 19687, 25930], dtype=int64), array([29413, 18366, 27703, ..., 28386, 18339, 18071], dtype=int64)) Component 3:
 (array([28975, 13114, 11317, ..., 20387, 23450, 12890], dtype=int64), array([ 9404, 29716, 30605, ..., 22671, 16437, 28215], dtype=int64))

Pixel 7
Component 1:
 (array([24650,  5634, 32342, ..., 16596, 11583,  1367], dtype=int64), array([ 9220, 30229, 22877, ...,  8443, 24115,  5357], dtype=int64)) Component 2:
 (array([25911, 23526, 12293, ...,    88,  7445, 19339], dtype=int64), array([27598, 24247,  6643, ..., 19536, 29039, 17517], dtype=int64)) Component 3:
 (array([23650,  3510, 30252, ...,  1153,  3391, 25684], dtype=int64), array([ 4412, 13116, 21552, ...,  3363, 32694, 25072], dtype=int64))

Pixel 8
Component 1:
 (array([22032, 22607, 22544, ...,  2887,  6359, 28979], dtype=int64), array([22634, 19789,  9671, ..., 29113, 27485, 17111], dtype=int64)) Component 2:
 (array([20162, 16781, 14863, ..., 30956, 10347, 30150], dtype=int64), array([ 1075, 29799, 27846, ..., 22105, 19856, 10126], dtype=int64)) Component 3:
 (array([26410, 14931, 25023, ..., 14890, 22757, 24174], dtype=int64), array([25791, 20416, 18909, ...,  7512,  3131, 26052], dtype=int64))

Pixel 9
Component 1:
 (array([20695, 10411, 12648, ..., 13125,  4648, 22255], dtype=int64), array([29491,  1782,  8249, ..., 14070, 10288, 24926], dtype=int64)) Component 2:
 (array([ 3946, 32167,  9129, ..., 17226, 26049,  4770], dtype=int64), array([22656, 25281, 12458, ...,  9681,  7507, 17201], dtype=int64)) Component 3:
 (array([ 6157,  6801, 22265, ..., 25538, 20845, 11927], dtype=int64), array([22145, 23929, 30634, ...,  1268, 24404, 10465], dtype=int64))

Pixel 10
Component 1:
 (array([26359, 21963, 20732, ..., 21433,  4437, 21665], dtype=int64), array([19501,  4526, 23241, ..., 10323, 29255, 18105], dtype=int64)) Component 2:
 (array([20432, 32610, 24657, ...,  3669, 28931, 26104], dtype=int64), array([  252, 20547,  4268, ..., 17001,  7677,  5347], dtype=int64)) Component 3:
 (array([16316, 17701,  5323, ...,  8440,  2169,  9799], dtype=int64), array([27569, 11385, 23297, ...,  9446, 30926,   344], dtype=int64))

Decrypted Data:
[50, 39, 19]
[49, 38, 18]
[48, 37, 17]
[47, 36, 16]
[45, 34, 14]
[44, 33, 13]
[41, 33, 14]
[40, 32, 13]
[47, 39, 20]
[50, 42, 23]
```

For all implementations, libraries were utilized for visualizing the original and encrypted images, allowing for a visual assessment of the encryption process.

Text Encryption

Second, the proposed text encryption methods were implemented using Python programming language. The implementation utilized several libraries, including random, numpy, and a weather API to enhance the encryption process. Here, we describe the implementation details for each of the two methods.

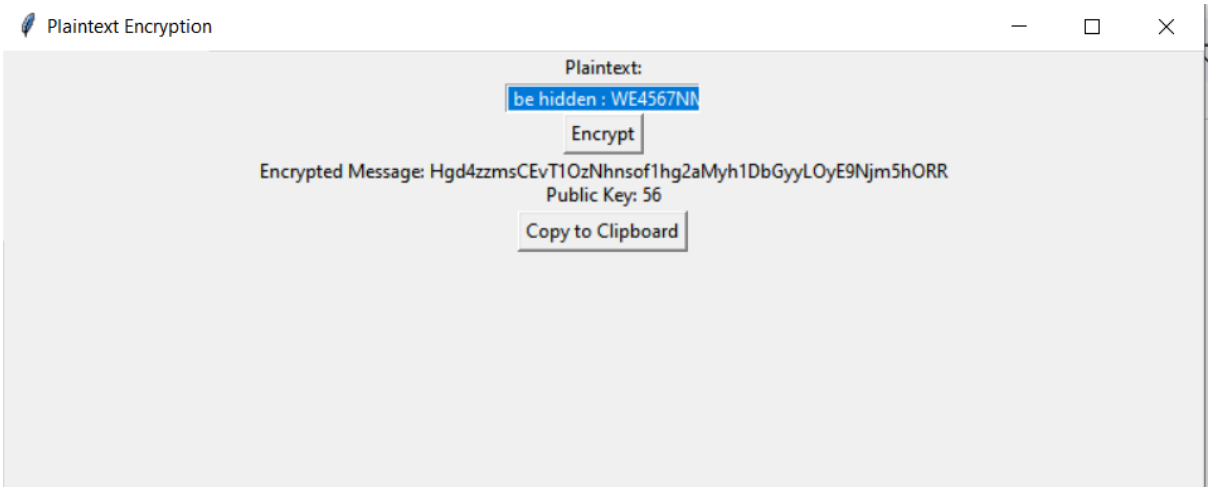
Base-n Conversion Approach:

The implementation of the first method involved a function that takes the plaintext input as an argument. The function generates a random base-n for the conversion process, where 'n' represents the desired base. The Unicode code points of each character in the plaintext are calculated to obtain their numerical values. Using repeated divisions, the function obtains a list of remainders that represent the digits in the base-n system. These remainders are then mapped to characters in a given alphabet string to create the final encrypted message.

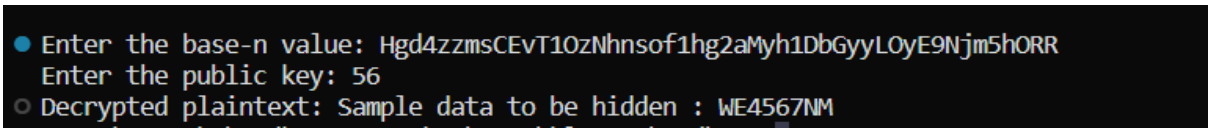
The implemented function returns the encrypted message along with the chosen base-n as a tuple, providing a secure and unique representation of the plaintext. The implementation utilized the random and numpy libraries to handle random number generation and array operations, respectively.

Encryption :

Plaintext - Sample data to be hidden : WE4567NM



Decryption :

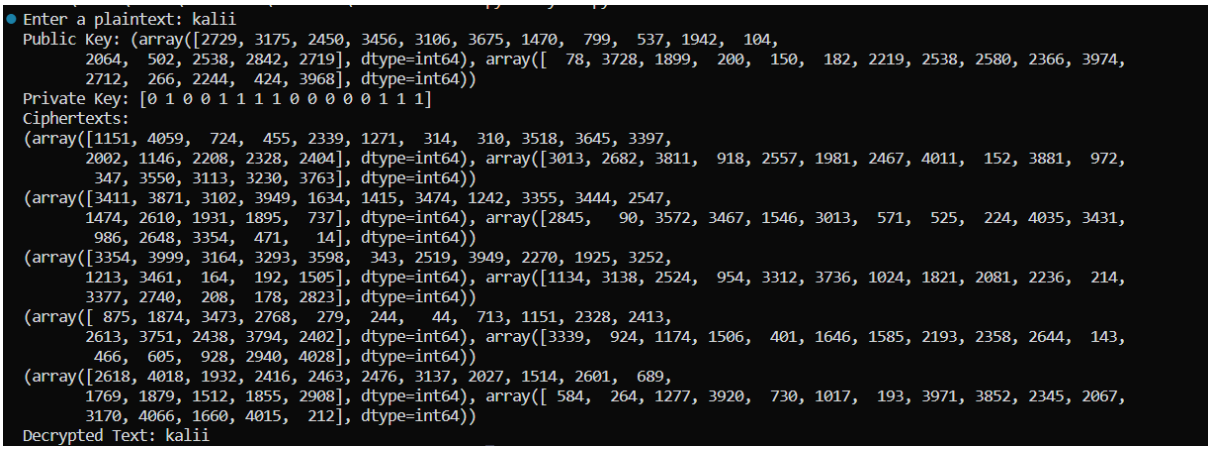


Basic Fully Homomorphic Encryption (FHE):

The implementation of the second method focused on a basic version of Fully Homomorphic Encryption (FHE). The implementation includes methods for key generation, encryption, and decryption. The makekey method generates a public-private key pair, crucial for the encryption and decryption processes. The encrypt method takes a plaintext value as input, scales it, and adds noise using random polynomials. Polynomial operations are performed on the public key components to produce the ciphertext.

The decryption method, decrypt, utilizes polynomial operations and the private key to remove the noise from the ciphertext and recover the original plaintext. In addition, the implementation incorporates aspects of the weather obtained from a weather API. This weather data is used to determine the powers of 2, which define key sizes and moduli for encryption. The weather information adds an additional layer of randomness and enhances the security of the encryption process.

The implementation employed various libraries, including random and numpy, to handle random number generation and perform mathematical operations.



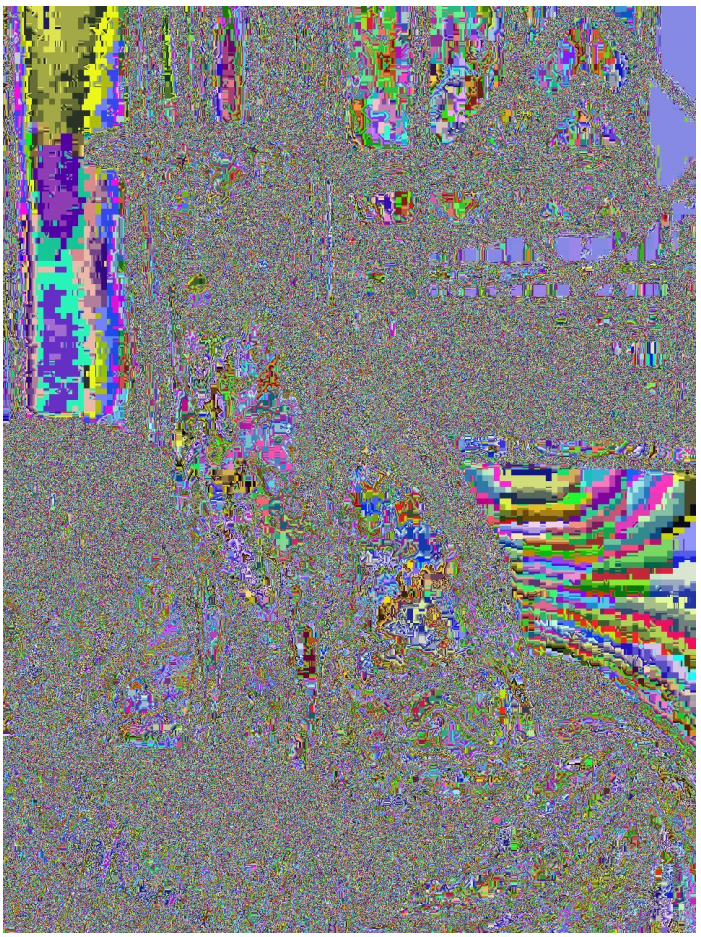
Result and Discussion

Image Processing:

The brute force method involves only a basic XOR operation using a key value which results in minimal to no security as it can be breached in several different ways, such as snooping, backdoors and almost any method possible. The use of any sort of randomness in this method would not solve the inherent weakness of the method itself such as the lack of an asymmetric key, the transmission of the byte array itself due to the change of the formatting codecs and the inability to have any sort of further encryption applied to it as the image is no longer accessible.

The second method, involving the use of matrix multiplication via a 3x3 matrix to change the RGB key values is far more secure and versatile. The image itself gets turned into a relatively noisy, incoherent set of pixels due to the change in the RGB values as shown in the picture below.

There is however a faint outline of the original contents at certain points that can be seen from the image itself but cannot be ascertained due to the levels of noise that it possesses. The encryption also would not work very well on monotones as it would simply cause a shift in the colours themselves, still being fairly revealing about the image itself



The decryption is also almost lossless with averages at around 0.000001% due to conversion of 32 bit integers to 8 bit RGB values. The decrypted image also shows no signs of the loss on the surface itself except for a few bits that can be corrected using a hamming code on the integers themselves or a CRC as a redundancy added to each encrypted pixel. Pictured below is the image after decryption.

For the case of the lattice encryption, the test was done on a few pixels of the image due to hardware constraints. The lattice itself is fairly heavy in terms of processing due to the process of addition and multiplication of 3 polynomials and the passing of every term through the respective ring. The ciphertexts themselves are almost impossible to extract via brute force and can only be done if the key is known to the user as the plaintext is inherently split up into two ciphertexts under the public key polynomial. The private key or secret key here is possibly the only means of decrypting the texts themselves. This method of encryption can be paired along with the matrix multiplication to add further security.

For implementation, however, a multiprocessor or server-grade processor with heavy multithreading capabilities would be needed to implement such a form of encryption simply because of the amount of time taken for the process itself. The polynomials also tend to have a certain level of loss when additional randomness is added due to the lack of control over the polynomial moduli parameters. Thus, either the randomness has to be contained within the polynomial generation process or scrapped altogether.

Text Encryption :

The base-n conversion method for encrypting text-based data proves to be extremely vulnerable as the public key that gets generated is the value of n itself, and all an attacker would need to do is put convert the base of the encrypted text from ‘n’ to 256(the assumed starter base for the text). Randomness doesnt solve the problem as well, considering the algorithm itself can be brute-forced to convert the base of the encrypted text from its current one to several other values and find a meaningful plaintext obtained from the conversion results.

However, a lattice-based encryption system proves to be much more secure. It allows the computation to be carried out while keeping the data encrypted, due to its homomorphic properties, minimizing exposure to potential threats. Meanwhile in a “base-n conversion" encryption, the data is decrypted during computations, making it vulnerable to attacks if the decryption process is compromised.

Conclusion

In this paper, we explored two distinct approaches for probabilistic encryption: the base-n representation approach and the basic Fully Homomorphic Encryption (FHE) method. Through a comprehensive analysis and evaluation of these methods, we have made several significant findings.

The base-n representation approach demonstrated its effectiveness as a straightforward and easily implementable encryption scheme. By converting plaintext into a secure and unique base-n representation, this approach provided confidentiality and a large number of possible representations for each character. It offered a practical solution for securing sensitive information, particularly in scenarios where simplicity and efficiency are essential.

On the other hand, the basic FHE method presented a promising technique for performing computations on encrypted data while ensuring end-to-end security. By integrating aspects of a weather API to enhance the encryption process with randomness and weather-dependent key sizes, we achieved an additional layer of security. The basic FHE implementation showcased its potential in secure computation on encrypted data, although further optimizations and advancements are required to address scalability and computational overhead challenges.

The research presented here contributes to the advancement of probabilistic encryption techniques by providing insights into the strengths and limitations of the base-n representation approach and the basic FHE method. Our evaluation considered factors such as encryption strength, computational efficiency, resistance against cryptographic attacks, and suitability for different use cases. These findings can guide researchers and practitioners in selecting appropriate encryption methods based on their specific requirements and constraints.

Future research in probabilistic encryption should focus on further enhancing the security and efficiency of both the base-n representation approach and the FHE method. Exploring optimizations, advanced cryptographic techniques, and

real-world applications will help advance the field and address the limitations identified in this research.

In conclusion, the base-n representation approach and the basic FHE method offer valuable contributions to probabilistic encryption. Their respective strengths and characteristics make them suitable for different encryption scenarios. By continuing to explore and refine these approaches, we can achieve more secure and efficient mechanisms for protecting sensitive information in various domains.