National College of Ireland

HexaCha: A Lightweight Hybrid Encryption Model for Password and Message Protection

MSc Research Project
MSc Cybersecurity

Aryan Ingale
Student ID: x22178511

School of Computing
National College of Ireland

Supervisor: Rohit Verma

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Aryan Sahebrao Ingale
Student ID:	x22178511
Programme:	MSc in Cybersecurity **Year:** 2023-24
Module:	MSc Research Project
Supervisor:	Rohit Verma
Submission Due Date:	15th December 2023
Project Title:	HexaCha: A Lightweight Hybrid Encryption Model for Password and Message Protection
Word Count:	6907 **Page Count:** 22

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Aryan Sahebrao Ingale

Date: 14th December 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

HexaCha: A Lightweight Hybrid Encryption Model for Password and Message Protection

Aryan Ingale
x22178511

Abstract

The purpose of this study is to investigate the integration and performance assessment of the Hybrid combination of Honey Encryption and ChaCha20 naming "*HexaCha*" encryption within a Flask-based web application framework. The study's goal is to determine how effective these algorithms are in strengthening data security and discouraging illegal access. It studies the throughput and execution times of HexaCha Encryption across various data sizes via systematic testing, considering encryption and decryption speeds in proportion to data size and then comparing it to prior studies of AES Encryption which is today's Industry Standard for data encryption. The paper dives into the avalanche effect in cryptography and the system's potential to deceive unwanted access attempts, delivering convincing outputs for wrong password inputs, and highlighting the importance of Honey Encryption in enhancing security by creating both true and decoy passwords. This finding highlights HexaCha's ability to achieve quicker execution rates and effectively process a wide range of data volumes. This study shows that the HexaCha which is a Hybrid encryption algorithm, is superior in terms of security comparatively with industry leading algorithm.

Keywords: Hybrid Encryption, ChaCha20, HexaCha

1 Introduction

1.1 Background

With the growth of encryption methods targeted at protecting sensitive information, the landscape of data security within information related services has seen major advances. The honey encryption innovation (Win et al., 2018) is the design of the distribution-transforming encoder (DTE). Encryption systems such as the Advanced Encryption Standard (AES) and the ChaCha algorithm have emerged as major challengers in data transport and storage security in prior studies. AES, a symmetric encryption standard, has gained popularity due to its strong security features, but ChaCha, a stream cypher, has gained popularity due to its lightweight design and efficient performance. The dynamic nature of cyber threats, on the other hand, needs ongoing study and evaluation of encryption solutions to maintain adequate security against unauthorised access and data breaches. Furthermore, the use of Honey Encryption, which creates dummy passwords in addition to genuine ones, is a viable option for enhancing security measures (Jaeger et al., 2016). The essence of this research is combining the lightweight nature of ChaCha algorithm and securing properties of Honey encryption to create an efficient Hybrid encryption for devices with low computational power such as IOT devices, smart homes etc. to secure any stored or transmitted data from un-

authorized access while understanding the complexities, performance, and comparative capabilities of leading encryption method with developed Hybrid algorithm. This research will investigate the performance, efficiency, and applicability of HexaCha algorithm, to provide significant insights into improving data security in resource limited environments.

1.2 Aim of the study

The goal of this research is to develop and thoroughly evaluate by comparing the performance of the developed hybrid encryption of ChaCha20 and Honey which is "*HexaCha*" Encryption with the industry leading encryption algorithm "*AES*", with a focus on assessing throughput, execution times, and efficiency across a wide range of data sizes. The study's objectives are to investigate the impact of varying data sizes on the encryption and decryption processes of both algorithms, and to examine the security of the hybrid model by calculating the avalanche effect on decryption outputs, finally we will use comparative analysis on both the algorithms to understand their respective pros and cons and to conclude which algorithm would be best suited for low resource environments and will follow the following question for the complete research.

How does HexaCha compare with AES in low resource environments?

Furthermore, the study aims to evaluate the system's capacity to deceive unauthorized access attempts by producing false outputs for wrong password inputs and generate alerts for such attempts. This research intends to give useful insights into the efficiency and dependability of HexaCha in securing sensitive data in current security contexts by understanding their comparative strengths, shortcomings, and practical applications within data security.

1.3 Research Objectives

The research objectives of this report are:

1. **Develop Algorithm**: To develop HexaCha and AES-Honey Encryption models using Python and deploy them on a flask-based web-application.
2. **Analyse Throughput and Execution Times**: To compare and assess the throughput and execution times of HexaCha encryption algorithms across different data sizes.
3. **Investigate Avalanche Effect**: Examine the effect of minor changes in input data on the HexaCha encryption decryption output.
4. **Comparative Analysis**: To contrast the advantages and disadvantages of HexaCha over AES, with a focus on their provided data security and execution efficiency.

1.4 Research Gaps

The restricted breadth of comparative studies comparing the performance details of Honey and ChaCha encryption algorithms working in a hybrid environment within practical application frameworks is one significant gap. There may be a collection of substantial comparison assessments that completely evaluate these algorithms in terms of throughput, execution durations, and efficiency over a wide range of data sizes and real-world scenarios in the existing literature. This gap prevents a comprehensive knowledge of their applicability

and limits in various settings such as online applications, IOT devices, smart vehicles and many more. Furthermore, while the use of Honey Encryption improves security by generating decoy passwords, the literature may under investigate its effectiveness in deterring unauthorised access attempts or its adaptability to emerging cyber threats while also analysing its performance usage while generating random seeds and messages from a huge dictionary of words. In-depth examinations of Honey Encryption's optimization or refinement procedures may be lacking in the research, limiting its full potential for improving security measures in low computational devices. Furthermore, while the current study focuses largely on encryption technique in a web application and sets a foot in a direction of additional study of these hybrid models, the broader usefulness of these algorithms in diverse digital domains such as IoT ecosystems, cloud computing, or network security may go unexplored.

2 Literature Review

2.1 Honey Encryption

Honey encryption is a cryptographic method for securing data in which false also known as "*honey*" information is generated in the event of an invalid decryption attempt. The word honey is often used in computer security to refer to decoys. This approach attempts to make it difficult for attackers to distinguish between correct and incorrect decryption results while performing a brute force attack (Juels and Ristenpart, 2014). Honeywords have been proposed as a method of protecting passwords (Burgess, 2017). In computer security, the term honey refers to a fake resource designed to deceive or lure an attacker (Noorunnisa and Afreen, 2016). Using a wrong key in classical encryption usually results in complete gibberish or random output. Honey encryption, on the other hand, uses a different technique. When an incorrect key is used to decrypt data, plausible looking but inaccurate information is output that appears to be genuine rather than pure gibberish. This deceptive output, sometimes referred to as "*honey*" data, can fool attackers into thinking they have successfully decrypted the information when they have received fake data or decoys.

2.2 Hybrid Models

In a rapidly evolving communication technology landscape, (Raj et al., 2020) pioneered a Hybrid cryptographic method to improve data exchange security. Their method combined Rivest Cipher 5 (RC5) for encryption and decryption with Honey Encryption (HE), addressing the problem of secret key exchange in symmetric cryptography. (Bangera et al., 2020) offered a unique solution to fundamental data security problems in digital communication by combining Homomorphic Encryption (HE) and Honey Encryption (HE) to protect data confidentiality and user authentication from brute force attacks on decryption keys. (Prabhu et al., 2021) focused on the vital requirement for comprehensive security inside Medical Imaging Systems, emphasising the protection of sensitive patient information. On the cloud computing front, (Dutta et al., 2023) recognised the vulnerabilities of data transmitted over the internet and consolidated various encryption techniques to fortify data privacy and authentication, leveraging Advanced Encryption Standard (AES), proxy re-encryption, Honey encryption, and N-th degree Truncated Polynomial Ring Unit (NTRU). (Jain et al., 2022) aimed to enhance password protection systems against rising cyber threats,

proposing a combination of Honey encryption (HE) and Twofish encryption to decrease eavesdropping risks provided by compromised cryptographic keys. The use of Honey Encryption, which injects bogus but believable data into encrypted messages to obscure illegal access attempts, is a common thread across this research.

Study (Year)	Focus Area	Encryption Techniques Used	Key Objectives	Main Challenges Addressed
Raj et al. (2020)	Data Exchange Security	RC5, Honey Encryption (HE)	Bolster data exchange security	Secret key exchange in symmetric cryptography
Bangera et al. (2020)	Digital Communication Security	Homomorphic Encryption (HE), HE	Fortify data confidentiality, user authentication	Brute force attacks on decryption keys
Prabhu et al. (2021)	Medical Imaging System Security	-	Robust security for medical data	Protection of sensitive patient details
Dutta et al. (2023)	Cloud Computing Data Security	AES, proxy re-encryption, HE, NTRU	Enhance data privacy, authentication	Vulnerabilities of data transmitted over the internet
Jain et al. (2022)	Password Protection System Security	Twofish, Honey Encryption (HE)	Strengthen password security, mitigate interception risks	Risks associated with compromised cryptographic keys

Table 2.1: Literature Review on Hybrid Models

2.3 ChaCha algorithm

In recent years, the explosion in data quantities across computer networks has prompted creative techniques to protect and manage this increasing amount of data, driven by internet use, smart gadgets, and broad online applications. Several researchers have proposed novel cryptographic algorithms to solve the issues given by growing data quantities and complexities. (Mahdi et al., 2021) proposed the Super ChaCha stream cypher to improve IoT data security by increasing resistance to cryptanalysis without sacrificing performance. (Rajaprakash et al., 2020) employed the RBJ25 method to handle big data volumes in web applications, leveraging matrix operations to address efficiency difficulties inherent in large datasets. Similarly, (Raichandran et al., 2020) attempted to improve the security of the ChaCha stream cypher family by utilising prime number analysis and quadratic equations to increase security without losing encryption performance. (Jain et al., 2022) took a different approach to securing Cyber-Physical Systems (CPS), proposing an access control mechanism combining authentication, encryption, and access control for robust data protection within CPS environments, and (Jaishanker et al., (2021) introduced the RBJ20 algorithm, employing layered matrix operations to address the difficulties and challenges of data transferred over networks, focusing on encryption and protection in large-scale data storage.

Author (Year)	Purpose	Key Features	Strengths	Weaknesses	Proposed Algo/Approach
Mahdi et al. (2021)	Enhance IoT data security	Improved ChaCha variant	Increased resistance to cryptanalysis	Potential trade-off between security and speed	ChaCha
Rajaprakash	Address	RBJ25	Managing	Efficiency	ChaCha

et al. (2020)	data volume in web apps	algorithm with matrix operations	massive data volumes, security	challenges in large datasets	
Raichandran et al. (2020)	Enhance ChaCha stream cipher family	Prime number analysis, quadratic equations	Improved security without compromising speed	Trade-offs between encryption speed and security	ChaCha
Jain et al. (2022)	Secure CPS data and access control	Authentication, encryption, access control	Robust security for CPS data	Balancing efficiency and security	ChaCha
Jaishanker et al. (2021)	Address data complexity in networks	RBJ20 algorithm with layered matrix operations	Innovative approach for complex data protection	Lack of specific result assessment	ChaCha

Table 2.2: Comparative analysis of ChaCha

Along with these, there are several other authors who have worked on ChaCha, such as (Shobana et al., 2020) who investigated the ChaCha family design, stressing the encryption speed of ChaCha20 variations and noting ChaCha20's problem in choosing speed above data security. To counteract this, they developed the RB22 algorithm, which focuses on column operations, secret key multiplication, and perfect number swapping. (Aggarwal et al., 2023) focused on improving encryption efficiency for Twitter data, offering the S-RSB-23 encryption scheme, which required converting ASCII to hexadecimal numbers via cell pairing and swapping. (Jaichandaran et al., 2023) attempted to improve security in healthcare data analysis by proposing the ES-BR22-001 approach, which prioritised data security and performance via key determination, sparse matrix use, and paired values manipulation. (Zahoor and Kaur, 2021) addressed IoT security problems by employing the current network security approaches of ChaCha, with the goal of fulfilling important security aspects while navigating obstacles linked to the limits of IoT. (Basha et al., 2022) concentrated on securing health-related data for predictive analysis, proposing the BR22-01 method, which included secret key determination, matrix manipulations, and enhanced security measures to support predictive analysis while dealing with adoption complexities and potential computational demands.

Author & Year	Purpose	Key Features	Strengths	Weaknesses	Proposed Algorithm/Approach
Shobana et al., 2020	Evaluate ChaCha family design and propose improvement	ChaCha20 variants comparison, Encryption speed focus	High encryption speed, Variants comparison	Limited focus on data security, Algorithm bias towards speed	RB22 Algorithm: Column operations, Secret key multiplication, Swap perfect numbers
Aggarwal et al., 2023	Enhance encryption efficiency for Twitter data	S-RSB-23 encryption technique, ASCII to hexadecimal conversion	Improved encryption throughput	Lack of dependable cryptographic techniques, Limited context applicability	S-RSB-23 Encryption Technique: Cell pairing, Hexadecimal transformation

Jaichandaran et al., 2023	Strengthen security in healthcare data analysis	ES-BR22-001 method, Data security and performance focus	Improved security, Comparative performance	Complexity of method implementation, Integration challenges	ES-BR22-001 Method: Key determination, Sparse matrix utilization, Paired values manipulation
Zahoor and Kaur, 2021	Address security concerns in IoT technology	Leveraging existing network security techniques for IoT	Essential security facets addressed	Adaptation challenges, Unique IoT constraints	Utilizing Network Security Techniques for IoT
Basha et al., 2022	Secure health-related data for predictive analysis	BR22-01 method, Security enhancement in healthcare data	Augmented security measures, Predictive analysis support	Complexity in adoption, Potential computational demands	BR22-01 Method: Secret key determination, Matrix manipulations, Enhanced security measures

Table 2.3: Comparative analysis of ChaCha and other Encryption Approaches

The studies highlighted in the literature review mostly focus on the performance of ChaCha20 in lightweight environments and the use of honey encryption can be helpful in mitigation of any MITM attacks or brute force attacks, since smaller devices tend to have a limited storage capacity and computing power, combining Honey Encryption along with already efficient ChaCha20 could be extremely beneficial in terms of efficiency as well as securing the stored and transmitted data from unauthorized access.

3 Research Methodology

The study builds upon existing literature, using it as a basis for evaluating the HexaCha encryption model as its primary objective. The methodology involves employing the Plan, Design, Code, Test, and approach procedures at each stage to construct a secure model. This research delves into the workings of Honey encryption with ChaCha20 and finally a comparative analysis is conducted between HexaCha and a hybrid encryption of AES and Honey algorithm to assess computational time, security, and effectiveness between the algorithms.

3.1 Technique for Generating Random Honeywords

Honey Encryption is an excellent protection method against password assaults, particularly brute force efforts, with the goal of protecting systems from unwanted access. Its major goal is to generate honeywords, which are fake credentials used to detect and thwart brute-force attacks. These honeywords help to strengthen the security architecture by misleading potential attackers who are seeking to understand legitimate passwords. Honeywords work by inserting bogus credentials alongside legitimate passwords into the system. These misleading passwords are generated using specific honeyword creation algorithms. The goal is to obscure the attacker’s assurance about the validity of passwords, reducing the value of stolen credentials and confusing hackers trying illegal access.

The honeyword production procedure begins with the input of a password and a message, followed by the selection of a random seed. This seed acts as a link between two dictionaries: passwords and seeds, and seeds and messages. Honeywords are created via mathematical manipulations and adjustments, sometimes known as password tweaking or tailing, which results in the formation of honeywords alongside genuine passwords. These honeywords, when integrated into the system, function as bogus passwords, strategically boosting the system's resistance against unwanted access attempts and therefore augmenting the overall security posture.

3.2 ChaCha20 Encryption

ChaCha20 encryption is a well-known cryptographic technique known for its durability and effectiveness in data security. This symmetric encryption technique uses stream cypher principles to secure the confidentiality and integrity of information during transmission or storage. One of ChaCha20's primary features is its ability to produce a keystream of pseudo-random integers using an encryption key and a unique nonce (number used once) (De Santis, Schauer and Sigl, 2017). This keystream is then merged with plaintext via a simple XOR operation to encrypt the data. Notably, the ChaCha20 design provides a high level of security against a variety of attacks, including differential and linear cryptanalysis. Because of the algorithm's versatility, it can handle a variety of key lengths and nonces, ensuring compatibility across a wide range of security protocols and applications. Furthermore, its computational efficiency across a broad range of platforms, including both hardware and software implementations, leads to its extensive use in protecting communications, data transfers, and storage systems.

ChaCha20 is frequently combined with the Poly1305 authenticator to produce the ChaCha20-Poly1305 architecture, which offers encryption as well as message authentication by generating a 16-bit hash for message block verification prior to the decryption function. This combination provides a strong deterrent to illegal access and data alteration, protecting not just secrecy but also integrity and authenticity. In practice, ChaCha20 encryption improves data security across a wide range of sectors, from securing communication channels in networking protocols like TLS 1.2 (Transport Layer Security) to safeguarding sensitive data in storage devices and assuring privacy in various software applications.

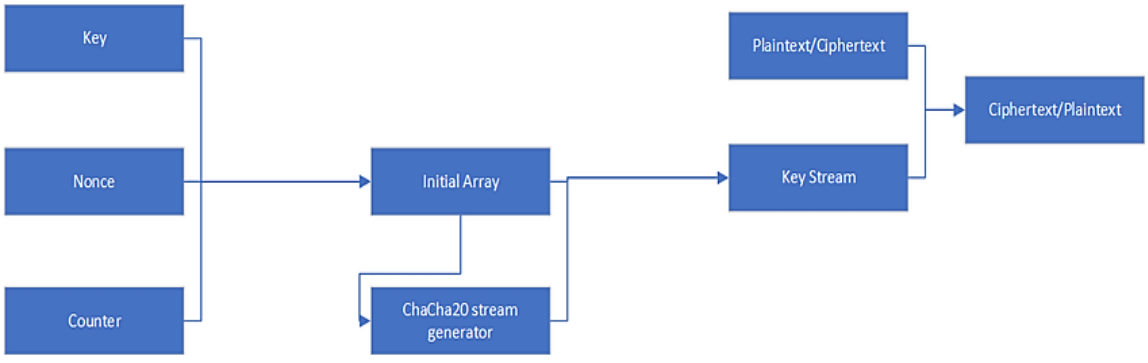

Figure 3.1. ChaCha Encryption/Decryption Flow by (McLaren et al., 2019)

4 Design

This section will discuss the design specifications of HexaCha, and the algorithms developed around those designs. This development phase employs a design technique centered on developing a user-friendly strategy for producing honey words, with an emphasis on simplicity and usability. The addition of Deterministic Threshold Encoding (DTE) is crucial to this approach (Juels and Ristenpart, 2014). The suggested method accomplishes two goals: it secures passwords with ChaCha20 encryption, and it provides security against illegal access by the generation of honey words. Furthermore, honey encryption is used to harden communications, especially protecting them against brute force assaults. This study also aims for a comparison analysis, examining the effectiveness of honey encryption when combined with AES, which is an industry leader in encryption standards, and comparing it with HexaCha. The research aims to discern the strengths and weaknesses of each approach in safeguarding sensitive information against potential threats by contrasting the use of honey encryption and by calculation of avalanche effect alongside performance matrices such as execution time and throughput, with these distinct cryptographic algorithms, thereby contributing to an understanding of their respective security and performance capabilities.

4.1 Algorithms

- **Proposed Algorithm: AES and Honey Encryption**

1. Start
2. The user inputs an encryption password.
3. The user inputs a secret message.
4. User input password and messages are mapped together.
5. A dictionary containing manipulated passwords (Honey Words) is generated which are similar to user input password.
6. The honey words are mapped with messages from a hardcoded dictionary of secret messages (in this program, these messages are U.S. states).
7. AES Encryption with padding is done on the user input password-message pair as well as on the honeywords-message pairs.
8. Encrypted Data is stored together with the "*honey data*".
9. User enters a decryption password to view the message.
10. A try/catch block is used to search for passwords in honey words, if the decrypted password exists in the dictionary of honey words it will raise an alarm and show a fake message and if the decrypted password is not found in the dictionary of honey words and does not match the correct user password an "Password not found" error is showed.
11. If the condition matches with the input password, then it will show the real message.
12. Evaluation of Algorithm using Avalanche Effect via entered Avalanche Password.
13. Stop

- **Proposed Algorithm: HexaCha Encryption**

1. Start
2. The user inputs an encryption password.
3. The user inputs a secret message.
4. User input password and messages are mapped together.
5. A dictionary containing manipulated passwords (Honey Words) is generated which are similar to user input password.
6. The honey words are mapped with messages from a hardcoded dictionary of secret messages (in this program, these messages are U.S. states).
7. Poly1305-ChaCha10 Encryption is done on the user input password-message pair as well as on the honeywords-message pairs.
8. Encrypted Data is stored together with the "*honey data*".
9. User enters a decryption password to view the message.
10. A try/catch block is used to search for passwords in honey words, if the decrypted password exists in the dictionary of honey words it will raise an alarm and show a fake message and if the decrypted password is not found in the dictionary of honey words and does not match the correct user password an "Password not found" error is showed.
11. If the condition matches with the input password, then it will show the real message.
12. Evaluation of Algorithm using Avalanche Effect via Avalanche Password.
13. Stop

5 Implementation

The implementation phase is discussed in this section, following were the system configuration on which HexaCha was developed and deployed on:

Specification	**Description**
Processor	Intel ® Core ™ i5-CPU@ 1.60GHz
Operating System	Windows 11
RAM	8.00 GB
System Type	64-bit operating system, x64-based processor

The programming language utilized in this project is Python 3.0, while Visual Studio Code served as the primary code editor. The primary python script has multiple classes defined for each hybrid encryption and respective Flask routes to move user input data to and from the backend script and acting as the entry point, coordinating the execution by invoking functionalities from these segregated classes.

5.1 HexaCha Encryption

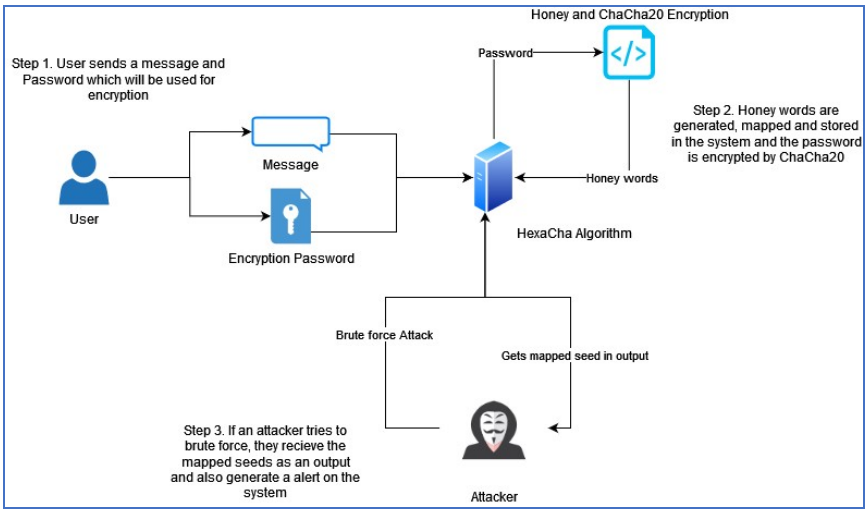

Figure 5.1.1: HexaCha Algorithm flow

The diagram (Figure 5.1.1) shows the general flow of HexaCha's implementation where the user passed message and password is used to generate honeywords and encryption. The password the user gives in the first step is mapped with the original message and then honey words are generated which are like the original password entered by the user and these honey words are mapped with fake messages and are linked to an alert system. So, if an adversary tries brute forcing the password to gain the original message and enters one of the honeywords while doing so the algorithm sends back the mapped fake message which in our case is the seed also while sending back an attack alert to the user or the system admin. We have used a similar mechanism for AES and Honey Hybrid for comparative analysis.

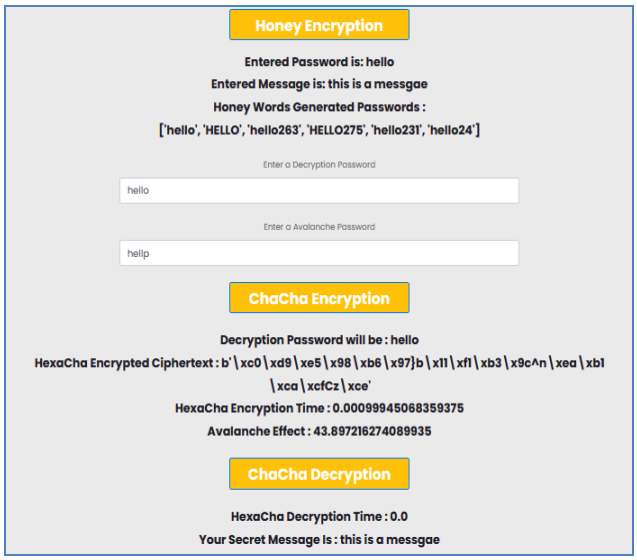

Figure 5.1.2: Program Output (HexaCha)

Initially the user will enter an encryption password and message to safeguard it (Figure 5.1.2), and then honey words are generated using honey encryption, e.g. HELLO, hello263, HELLO0275 are some of the honeywords of the original passwords "*hello*". Once the honey words are generated, they are mapped to their respective seed i.e. message and are stored in

the backend after performing ChaCha20 encryption, as the user enters a correct password for decryption the original message is displayed by the algorithm. Similar implementation was done for AES and Honey encryption.

As the secret message (Figure 5.1.2) displays the wrong message as the output, when an erroneous password is supplied for decryption and it matches with any of the honey words, this notice appears, indicating a discrepancy between the input and the system's stored passwords. If this is a brute force attack by any attacker, then they will receive a wrong message as the output in this case "*California*" which is displayed instead of the original message which was "*this is a message in AES*" also, an incident will be reported to the admin or the user regarding the password use. On the other hand, if the decryption password doesn't match any of the honey words or the original password then an error message "*Password not found*" is shown as output.

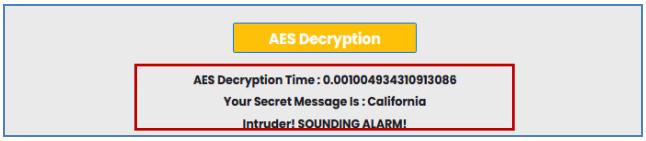

Figure 5.1.2: Fake Message as output

5.2 Honey Encryption

The primary factor of honey encryption is the generation of honey words and mapping them to seeds which are fake messages.

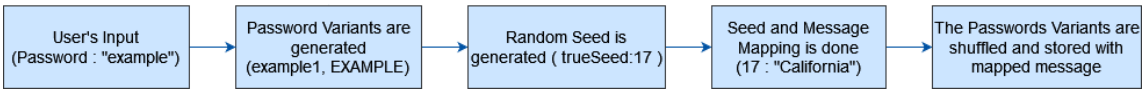

Figure 5.2.1 Working of Honey Encryption

The "*honeyencryption*" function defined in the python script, implements this honey encryption scheme, creating a secure mapping of passwords to ciphertexts (Figure 5.2.1), It initializes two dictionaries for mapping passwords to seeds and seeds to messages, respectively. The true seed is a randomly selected integer, and a list of U.S. state abbreviations is used as potential messages. The function then generates multiple honey words by manipulating the input password and mapping them to incrementally derived seeds, each linked to a different state name which would be the fake messages. The function culminates by shuffling the honey words, obfuscating the true password among decoys. This approach enhances security by complicating the differentiation between the actual password and honey words for an attacker.

6 Evaluation

6.1 Analysis of HexaCha

In this section we will investigate the performance and security analysis of the developed HexaCha algorithm.

6.1.1 Performance Analysis

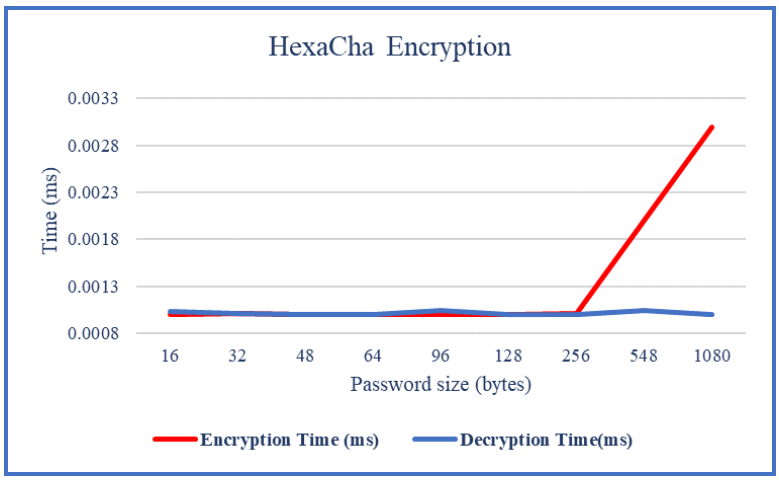

Figure 6.1: Graph of the decryption and encryption time of Chacha

This data in Figure 6.1 demonstrates the link between password size (measured in bytes) and the required time for HexaCha algorithm encryption and decryption operations. The graph shows that while decryption time remains relatively stable and low across all password sizes, the encryption time remains stable until the password size reaches 256 bytes, after which then increases sharply. This suggests that the HexaCha Encryption algorithm may become significantly slower to encrypt as the password size increases beyond a certain point, while decryption time is less affected by password size, furthermore the following formulae will be used for additional calculations:

Total execution time = Encryption Time + Decryption Time

Throughput = Password Size in MB / Total time for execution in seconds

The following results were obtained from the calculations, which will be essential for comparative analysis.

Size (bytes)	Size (MB)	Encryption Time (ms)	Decryption Time(ms)	Total time for Execution (ms)	Total Time for Execution (s)	Throughput
16	0.000016	0.000998974	0.001028538	0.002027512	2.027511597	0.078914468
32	0.000032	0.001010656	0.001013994	0.002024651	2.024650574	0.158051964
48	0.000048	0.001000643	0.000998974	0.001999617	1.999616623	0.240046014
64	0.000064	0.00100255	0.000999451	0.002002001	2.002000809	0.319680191
96	0.000096	0.001001358	0.00104475	0.002046108	2.046108246	0.469183388
128	0.000128	0.000998259	0.000999689	0.001997948	1.997947693	0.640657413
256	0.000256	0.001006603	0.000999212	0.002005816	2.005815506	1.276288867
548	0.000548	0.002002954	0.001038074	0.003041029	3.041028976	1.802021632
1080	0.00108	0.002997637	0.001000881	0.003998518	3.99851799	2.701000727

Table 6.1: Shows the throughput values of HexaCha.

6.1.2 Security Analysis

Plain Text (Password)	Size (bytes)	Changed Plaintext	Avalanche Effect
1234Abcd!@#$5678	16	1234Abcd!@#$567**9**	51.50%
1234Abcd!@#$Efgh5678!@#$	25	1234Abcd!@#$Efgh5678!@#**!**	45.60%
1234Abcd!@#$Efgh5678Ijkl!@#$%&*()12342	38	1234Abcd!@#$Efgh5678Ijkl!@#$%&*()1234**1**	52.57%
1234Abcd!@#$Efgh5678Ijkl!@#$%&*()1234Abcd!@#22	46	1234Abcd!@#$Efgh5678Ijkl!@#$%&*()1234Abcd!@#2**3**	50.74%
1234Abcd!@#$Efgh5678Ijkl!@#$%&*()1234Abcd!@#$Efgh5678213	56	1234Abcd!@#$Efgh5678Ijkl!@#$%&*()1234Abcd!@#$Efgh567821**4**	45.08%
1234Abcd!@#$Efgh5678Ijkl!@#$%&*()1234Abcd!@#$Efgh5678Ijkl!@#$%&*()123432eawd	76	1234Abcd!@#$Efgh5678Ijkl!@#$%&*()1234Abcd!@#$Efgh5678Ijkl!@#$%&*()123432eaw**e**	45.77%

Table 6.2: Avalanche Effect of HexaCha

The Avalanche Effect is illustrated in Table 6.2 in the context of encryption, showing how little changes in the plaintext input result in huge changes in the cipher text output. The '*Plain Text (Password)*' column displays the original passwords, whereas the '*Size (bytes)*' column displays the data sizes in bytes. The '*Plaintext Changed*' column displays revisions made to the original password, highlighting minor changes (alphabets in bold font) where we have made sure to change only one byte of the original password. The '*Avalanche Effect*' column calculates the proportion of change caused by the alterations based on the differences in the resultant cipher texts. HexaCha has a strong Avalanche Effect, hovering around 50%, which indicates even a single byte change in the password results in a major change in ciphertexts.

6.2 Analysis of AES and Honey Encryption

6.2.1 Performance Analysis

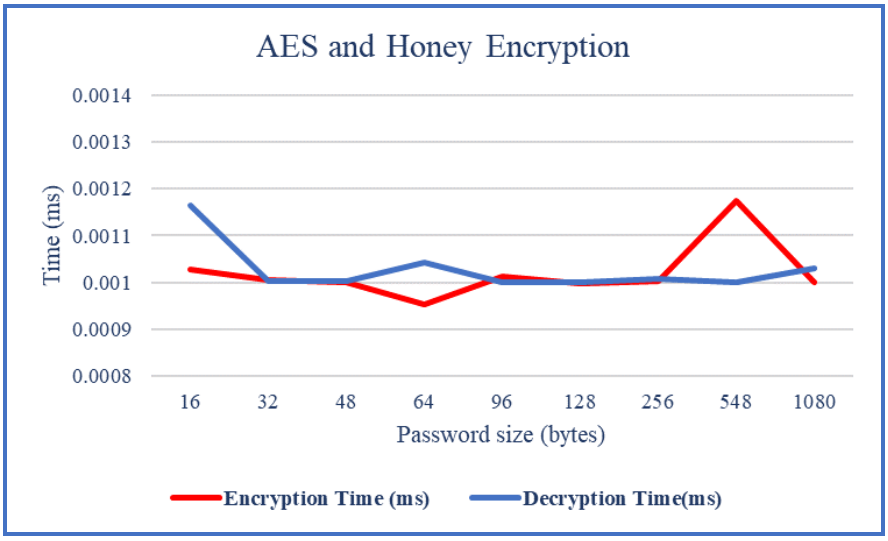

Figure 6.2: Graph of the decryption and encryption time of AES

This graph in Figure 6.2 depicts the link between different data byte sizes and the required time for encryption and decryption using the AES method. Overall, both lines exhibit fluctuations across the password sizes, with no clear trend of increase or decrease as password size changes. The encryption and decryption times for both algorithms appear relatively close, with occasional crosses, suggesting that for AES and Honey Encryption, the time it takes to encrypt, or decrypt does not significantly differ as the password size changes within the observed range. The graph indicates that both encryption and decryption processes for these algorithms are relatively stable and efficient across the given password sizes, and similarly did for HexaCha we have calculated Total Execution times and Throughput for AES.

Size (bytes)	Size (MB)	Encryption Time (ms)	Decryption Time(ms)	Total time for Execution (ms)	Total Time for Execution (s)	Throughput
16	0.000016	0.001027346	0.001163721	0.002191067	2.191066742	0.073023791
32	0.000032	0.00100565	0.001002312	0.002007961	2.007961273	0.159365623
48	0.000048	0.000999212	0.001001835	0.002001047	2.001047134	0.23987441
64	0.000064	0.000953436	0.001041651	0.001995087	1.99508667	0.320788069
96	0.000096	0.001012802	0.001000643	0.002013445	2.013444901	0.476794771
128	0.000128	0.000998497	0.000999451	0.001997948	1.997947693	0.640657413
256	0.000256	0.00100255	0.001008272	0.002010822	1.008272171	2.538996983
548	0.000548	0.001174688	0.001000881	0.00217557	2.175569534	2.518880649
1080	0.00108	0.000999928	0.00103116	0.002031088	2.031087875	5.317347482

Table 6.3: Shows the throughput values of AES.

6.2.2 Security Analysis

Plain Text (Password)	Size (bytes)	Changed Plaintext	Avalanche Effect
1234Abcd!@#$5678	16	1234Abcd!@#$567**9**	23.89%
1234Abcd!@#$Efgh5678!@#$	25	1234Abcd!@#$Efgh5678!@#**!**	44.37%
1234Abcd!@#$Efgh5678Ijkl!@#$%&*()12342	38	1234Abcd!@#$Efgh5678Ijkl!@#$%&*()1234**1**	48.45%
1234Abcd!@#$Efgh5678Ijkl!@#$%&*()1234Abcd!@#22	46	1234Abcd!@#$Efgh5678Ijkl!@#$%&*()1234Abcd!@#2**3**	52.29%
1234Abcd!@#$Efgh5678Ijkl!@#$%&*()1234Abcd!@#$Efgh5678213	56	1234Abcd!@#$Efgh5678Ijkl!@#$%&*()1234Abcd!@#$Efgh567821**4**	11.78%
1234Abcd!@#$Efgh5678Ijkl!@#$%&*()1234Abcd!@#$Efgh5678Ijkl!@#$%&*()123432eawd	76	1234Abcd!@#$Efgh5678Ijkl!@#$%&*()1234Abcd!@#$Efgh5678Ijkl!@#$%&*()123432eaw**e**	55.36%

Table 6.4: Avalanche Effect of AES.

Similarly, as illustrated in Table 6.2, The above table (Table 6.4) depicts the Avalanche Effect in AES encryption, demonstrating how little changes in plaintext inputs result in significant changes in the encrypted output. Similarly, changes to different passwords resulted in varied degrees of variation in the encrypted output. However, the observations

depict a comparatively lower avalanche effect for smaller sized passwords and a gradual increase as the byte size increases, this shows AES gives similar looking cipher text, almost 80% similarity between them, when the sizes are smaller, but the effect becomes noticeable once the byte size starts increasing.

6.3 Comparative Analysis of HexaCha and AES

The graph in figure 6.3 depicts a comparison between HexaCha and AES encryption times.

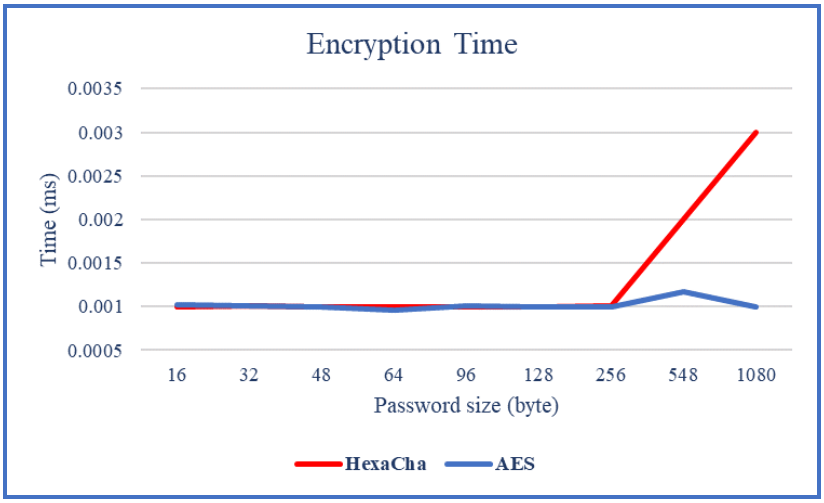

Figure 6.3: Graph of the encryption time of HexaCha and AES

The graph in Figure 6.4 depicts a comparison between HexaCha and AES decryption. Variation in execution time when password length changed. By combining Honey Encryption with ChaCha20, this study developed HexaCha, a hybrid encryption technique. We observed that the combination of Honey Encryption with ChaCha20 exceeds the rest in terms of security and performance after comparing it to other hybrid encryption schemes such as "*AES plus Honey*".

The graph (Figure 6.5) illustrates the throughput performance of AES and HexaCha encryption algorithms across various password sizes, which are depicted on the x-axis ranging from 16 to 1080 bytes. For both algorithms, the throughput appears to increase as the password size increases. The AES algorithm shows a more gradual and consistent increase throughout the range of password sizes. In contrast, the HexaCha throughput remains relatively stable up to 256 bytes, after which it exhibits a substantial increase, particularly from 548 to 1080 bytes.

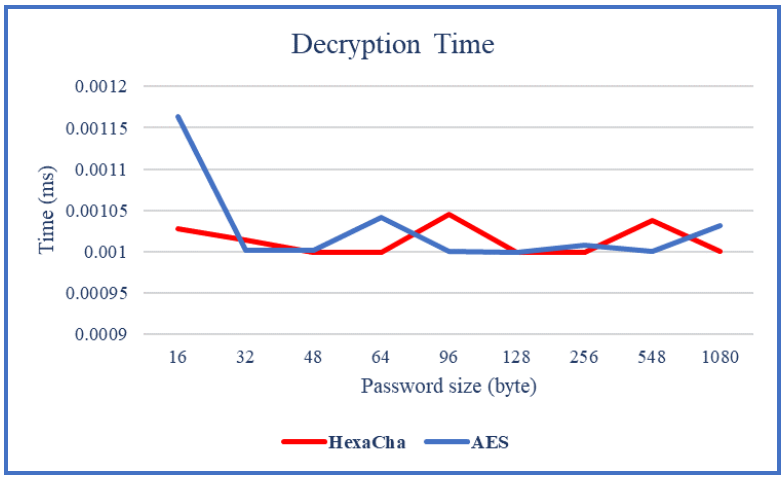

Figure 6.4: Graph of the Decryption time of HexaCha and AES

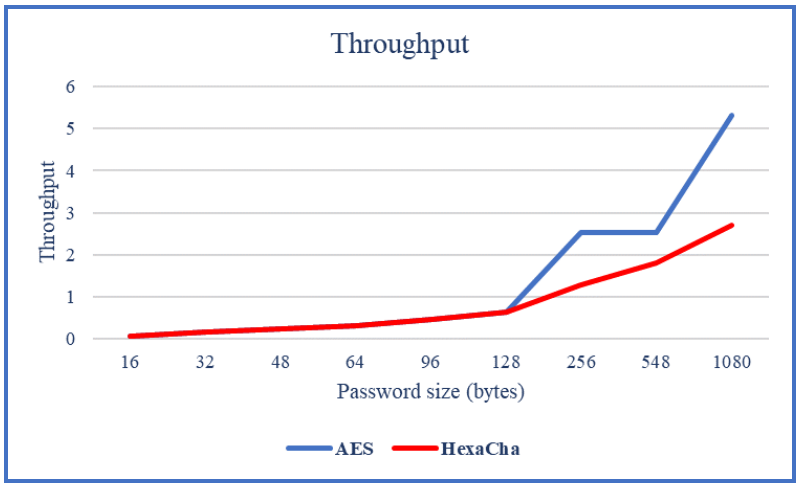

Figure 6.5: Graph of the Throughput of HexaCha and AES

In general, a higher throughput means the algorithm is consuming lower power. The overall trend suggests that larger password sizes may lead to higher throughput for these encryption algorithms, with AES showing a more pronounced growth at higher password sizes compared to HexaCha, meaning AES consumes lesser power as compared to HexaCha when processing larger sized bytes meaning both the algorithms can be preferable when processing lower sizes data but AES seems beneficial when working on larger data sizes.

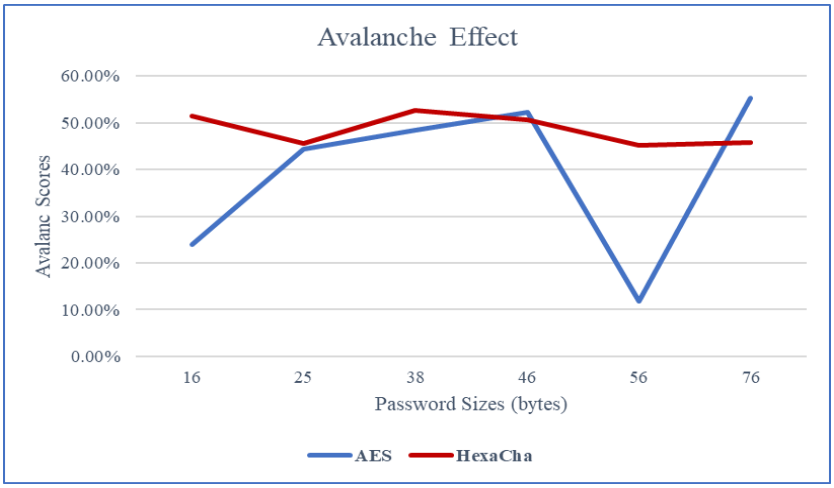

Figure 6.6: Graph of the Avalanche Effect of HexaCha and AES

The above graph (Figure 6.6), illustrates and compares the Avalanche Effect of HexaCha and AES, can be approached by considering the underlying principles of the Avalanche Effect and how they apply to both encryption algorithms, using the data from Tables 6.2 and 6.4 as additional aids.

Avalanche Effect in Cryptography: The Avalanche Effect is a desirable property in cryptography, where a small change in the input (e.g., a single bit) should cause a significant and unpredictable change in the output. This makes the encryption algorithm more secure against differential cryptanalysis, as it obscures the relationship between the plaintext and the ciphertext (Burgess, 2017).

In this analysis, It is observed that HexaCha has a steady observation at around 45-50% meaning that a single bit change between passwords results in approximately 50% difference in ciphertexts which is a cascading difference and the same cannot be said for AES and Honey encryption as the curve behaves erratically and showing extremely low readings of 11%, HexaCha outperforms AES and Honey hybrid for some degree in terms of constant avalanche effects and is secure for practical uses while being efficient.

6.4 Discussion

In this chapter, we compared HexaCha and AES encryption algorithms across various performance metrics. While HexaCha maintains stable encryption and decryption times up to a 256-byte password size, it experiences a significant slowdown in encryption beyond this point. AES, on the other hand, shows minor fluctuations without a clear trend relating to password size. Throughput analysis indicates AES's gradual performance increase across password sizes, whereas HexaCha’s throughput surges dramatically after the 548-byte mark. Additionally, HexaCha consistently demonstrates a stronger Avalanche Effect, suggesting enhanced security through better sensitivity to password changes. Overall, HexaCha shows potential advantages in security features, while AES displays steady performance, presenting a trade-off between security and efficiency. HexaCha would be a valid choice if the application is on a lightweight system or device, as the throughput data shows AES and HexaCha performs similarly at smaller data ranges while HexaCha provided better security overall. The use of HexaCha in IOT or similar devices will be a preferable choice as it provides better security and is lightweight and efficient at processing smaller data sizes and securing it from brute-force attacks.

7 Conclusion and Future Work

7.1 Conclusions

This study helps to understand the performance and security differences between the two algorithms, HexaCha and AES as they performed similarly in terms of performance while processing smaller data sizes but an increase in password size resulted in a substantial rise in execution time and resource utilization of both the algorithms where HexaCha ended up being less efficient. On the other hand, HexaCha outperformed AES in terms of stable and predictable security with higher and constant Avalanche Scores while processing smaller password sizes and AES barely catching up with the avalanche score at the higher password sizes. This gives the understanding that HexaCha performs like AES in terms of performance while processing smaller size data but offers way better security by a better avalanche score, making it a suitable option for devices or systems which process sensitive data and have a smaller computational power such as POS machines, IOT authenticator devices etc. which usually process smaller but sensitive data sizes.

7.2 Limitations

The study recognises many limitations that may have an impact on the breadth and generalizability of the findings. One significant limitation is the range of data sizes investigated. The study concentrated on a specific range of data sizes for analysing the performance of the Honey and ChaCha20 encryption algorithms within the web application framework. This narrow focus may limit the study's results' application to a larger range of data changes found in real-world circumstances. As a result, while the findings provide insights into algorithmic effectiveness across certain data sizes, extrapolation to various datasets may need more analysis and confirmation also in the complete evaluation process the complete memory used by the algorithm was not calculated which is one of the vital resources for any application and any low resource environment. Furthermore, development and evaluation of encryption algorithms in a web-based platform cannot completely prove its efficiency in IOT or other hardware-based devices so additional research and evaluation on different hardware can be beneficial.

7.3 Future Works

To confirm and extend the current findings, future research efforts should include a more extensive examination of data volumes and real-world scenarios. Investigating the practical implementations of these encryption algorithms in various applications, as well as their adaptability technologies such as IoT and cloud computing, will improve our knowledge of their applicability. Furthermore, future studies should focus on improving Honey Encryption approaches and determining their effectiveness against emerging cyber-attacks. Exploring hybrid encryption methods that incorporate Honey, ChaCha, and other developing algorithms may provide a more complete approach to data protection in changing digital landscapes.

References

McLaren, P., Buchanan, W.J., Russell, G. and Tan, Z., 2019. Deriving ChaCha20 key streams from targeted memory analysis. *Journal of Information Security and Applications*, *48*, p.102-372.

Jaeger, J., Ristenpart, T. and Tang, Q., 2016. Honey encryption beyond message recovery security. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I 35* (pp. 758-788). Springer Berlin Heidelberg.

Win, T., and Moe, K.S.M., 2018. *Protecting private data using improved honey encryption and honeywords generation algorithm* (Doctoral dissertation, MERAL Portal).

Burgess, J., 2017. Honey encryption review. *Queen's University Belfast*.

Noorunnisa, N.S. and Afreen, D.K.R., 2016. Review of honey encryption technique. *International Journal of Science and Research (IJSR)*, *5*(2), pp.1683-1686.

Raj, K.V., Ankitha, H., Ankitha, N.G. and Hegde, L.K., 2020, June. Honey encryption based hybrid cryptographic algorithm: a fusion ensuring enhanced security. In *2020 5th international conference on communication and electronics systems (ICCES)* (pp. 490-494). IEEE.

Bangera, S., Billava, P. and Naik, S., 2020, March. A hybrid encryption approach for secured authentication and enhancement in confidentiality of data. In *2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC)* (pp. 781-784). IEEE.

Jayahari Prabhu, G., 2021. Multimodal medical imaging security using hybridization of honey encryption algorithm with binary particle swarm optimization. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, *12*(10), pp.3905-3912.

Dutta, A., Bose, R., Roy, S. and Sutradhar, S., 2023. Hybrid Encryption Technique to Enhance Security of Health Data in Cloud Environment. *Archives of Pharmacy Practice*, *14*(3).

Jain, S., 2022. *Honey2Fish-An enhanced hybrid encryption method for password and messages* (Doctoral dissertation, Dublin, National College of Ireland).

Mahdi, M.S., Hassan, N.F. and Abdul-Majeed, G.H., 2021. An improved chacha algorithm for securing data on IoT devices. *SN Applied Sciences*, *3*(4), p.429.

Rajaprakash, S., Basha, C.B., Muthuselvan, S., Jaisankar, N. and Singh, R.P., 2020, December. RBJ25 cryptography algorithm for securing big data. In *Journal of Physics: Conference Series* (Vol. 1706, No. 1, p. 012146). IOP Publishing.

Jaichandran, R., Shalini, K.S., Leelavathy, S., Rajaguru, J.J. and PRADEEP, K., 2020. Securing generalized data using RB23 algorithm. *Advances in Mathematics: Scientific Journal*.

Jain, D.K., Mohan, P., Lakshmanna, K. and Nanda, A.K., 2022. Enhanced data privacy in cyber-physical system using improved Chacha20 algorithm.

Rajaprakash, S., Jaishanker, N., Basha, C.B., Muhuselvan, S., Aswathi, A.B., Jayan, A. and Sebastian, G., 2021. RBJ20 cryptography algorithm for securing big data communication using wireless networks. In *Intelligent Sustainable Systems: Selected Papers of WorldS4 2021, Volume 2* (pp. 499-507). Singapore: Springer Nature Singapore.

Bagath Basha, C., Rajaprakash, S., Nithya, M., Sunitha, C. and Karthik, K., 2022, March. The Security Algorithm BR22-01 Used to Secure the COVID'19 Health Data. In *International Conference on Deep Sciences for Computing and Communications* (pp. 345-354). Cham: Springer Nature Switzerland.

Zahoor, A. and Kaur, S., 2021. A review of algorithms for secure data transmission in iot devices. *Data Driven Approach Towards Disruptive Technologies: Proceedings of MIDAS 2020*, pp.83-95.

Rajaprakash, S., Muthuselvan, S., Jaichandaran, R., Sai Nithankumar, M., Sai Rathankumar, M., and Gupta, V.K., 2023. The Security Algorithm ES-BR22-001 Used to Secure the Health Data. In *Intelligent Sustainable Systems: Selected Papers of WorldS4 2022, Volume 2* (pp. 549-557). Singapore: Springer Nature Singapore.

Basha, C.B., Rajaprakash, S., Aggarwal, N., Riyazuddin, M.D., Sirajuddin, M. and Gole, S.B., 2024. An Innovative Cryptography Safety Algorithm Called S-RSB-23 for Protecting Data Using Machine Learning Algorithm. *International Journal of Intelligent Systems and Applications in Engineering*, *12*(2s), pp.503-510.

SHOBANA, R., JAICHANDRAN, R., SRISATHVIK, M., PRAKASH, M.B. and BHUVANESH, P., RB22 ALGORITHM FOR NEW SECURITY SYSTEM.

De Santis, F., Schauer, A. and Sigl, G. (2017). ChaCha20-Poly1305 Authenticated Encryption for high-speed Embedded IoT Applications. [online] IEEE Xplore. doi:https://doi.org/10.23919/DATE.2017.7927078.

Juels, A. and Ristenpart, T. (2014). Honey Encryption: Encryption beyond the Brute-Force Barrier. IEEE Security & Privacy, 12(4), pp.59–62. doi:https://doi.org/10.1109/msp.2014.67.