Graduate College Journal – NU
Vol.19 -2024, No. (7)
ISSN: 1858-6228, http://www.neelain .edu.sd

مجلة الدراسات العليا - جامعة النيلين
المجلد (19) –2024، العدد (7)
الرقم الدولي الموحد للدوريات: 1858-6228

Analyzing the Efficiency and Complexity of Cryptography Algorithm

A Focus on Elliptic Curve Diffie-Hellman (ECDH) Hybrid with Genetic Algorithm (GA)

Wafa Aref Ahmed Ibrahim[1*], Fakhreddin Abbas[2], and Fakhereldeen E.E Musa[1]

[1]Department of Computer Science, Faculty of Computer Science and Information Technology, Al-Neelain University, Khartoum, Sudan
[2]Department of Statistics, Faculty of Mathematical Sciences and Statistics, Al-Neelain University, Khartoum, Sudan.
[*]Corresponding author email: wafa.2011@live.com

Abstract

This paper focuses on the performance analysis and optimization of the ECDH_GA encryption algorithm. The algorithm is widely used for securing sensitive data, however, improving its performance while maintaining data integrity is a challenge. The study conducted experiments to analyze the impact of file size on encryption and decryption times. The results showed a predictable growth pattern, with varying execution times based on file size. The worst-case execution times exceeded the best-case times, and memory usage grew with larger file sizes and encryption operations. To optimize the algorithm, the study recommends exploring algorithmic improvements, hardware acceleration, and parallelization techniques. Scalability analysis, real-world performance testing, and energy efficiency assessments are also recommends. Security analysis and compliance considerations are important to ensure resistance against known attacks and adherence to industry standards. Usability and user experience studies can address key management and integration challenges. By implementing these recommendations and conducting further studies, developers can enhance the performance, scalability, and security of the ECDH_GA algorithm, enabling secure transmission and storage of sensitive data in various domains.

Key words: EC, Encryption, Decryption, Cryptography, Hybrid, Execution Times, submitted effort

Introduction

Security and protection issues have become increasingly important with the advancement of computing and information technologies in the modern era. To keep up with growing security concerns, encryption technologies have been continuously developed. These technologies provide advanced tools and methods to enhance encryption procedures and increase security. Despite the availability of many solutions, the most crucial factor remains how well security is maintained during the data exchange process. A breach in encryption can be disastrous for users, as it can lead to the exploitation of their data, enabling hackers to view and modify information.

To protect information, it has become necessary to use advanced encryption techniques along with more sophisticated and private procedures. While there has been considerable research in this field, there a remains a lack of studies focusing on the complexities of the hybrid encryption algorithm (ECDH_GA) (Alesawy, O., & Muniyandi, R. (2016)), assessing the resources needed for its implementation (such as time and storage capacity), evaluating its effectiveness and efficiency, and categorizing its performance into best, average, and worst-case scenarios.

This research aims to analyze the complexities of cryptographic algorithms using the ECDH hybrid algorithm with artificial intelligence GA. The research includes an analysis of the effects of different conditions on certain encryption algorithms, determining memory complexity, calculating the

number of operations required for each algorithm, measuring the amount of effort exerted in the algorithm, identifying the best, worst, and average cases, estimating the growth rate of the algorithm, and testing the efficiency and optimization of the mentioned encryption algorithms to determine the best among them.

This study is particularly important due to the development of hybrid encryption algorithms, such as ECDH with artificial intelligence techniques, which necessitates a precise analysis of these algorithms to identify the most efficient, time-saving, resource-efficient, and highest security solutions. This paper will review the latest studies that have examined the use of GA with elliptic curves in improving hybrid cryptographic algorithms to provide a deeper understanding of the performance and applications of these algorithms in practical fields. The study seeks to demonstrate the validity, efficiency, and effectiveness of new cryptographic technologies integrated with ECDH and advanced security technologies, enhanced performance and security in tackling advanced security and encryption challenges.

Literature Review:

This literature review investigates the integration of Elliptic Curve Diffie-Hellman (ECDH) and Genetic Algorithms (GA) in cryptographic systems to address data security challenges. Ekhlas Khalaf Gbashi (2018) introduced a method to enhance the security of Elliptic Curve Cryptography (ECC) using Genetic Algorithms. ECC typically employs a static encoding matrix to convert secret messages into points before encryption. Gbashi's approach introduces randomness into this predefined matrix, thus enhancing ECC's security. The information generated by the genetic algorithm is published as a public key alongside ECC public keys, allowing the receiver to utilize the same matrix as the sender (Gbashi, 2018). Johnson and Brown (2022) conducted a study on the algorithmic complexity of public key cryptography algorithms, evaluating their computational efficiency. They analyzed the time, space, and key generation complexities of popular algorithms such as RSA, ElGamal, and ECC. Their study utilized mathematical models and empirical evaluations to quantify these complexities and compare performances. The findings assist researchers and practitioners in making informed decisions when selecting suitable algorithms for specific cryptographic applications, contributing to a deeper understanding of algorithmic complexities (Johnson & Brown, 2022). Wang and Chen (2022) analyzed hash function complexities in cryptography, focusing on popular algorithms like SHA-256, SHA-3, and BLAKE2. They evaluated time complexity, space complexity, and collision resistance, considering design choices such as compression functions and message expansion. Their study used mathematical analysis and empirical evaluations to quantify these complexities and assess security features, aiding in selecting appropriate hash functions for specific cryptographic applications (Wang & Chen, 2022). Smith and Jones (2023) compared symmetric encryption algorithms based on complexity metrics, evaluating their efficiency and security. Their analysis considered time, space, and computational resources. This research helps in selecting suitable algorithms for cryptographic applications by providing a better understanding of complexity aspects, thereby aiding informed decisions regarding algorithm selection and implementation. The findings contribute to a comprehensive understanding of symmetric encryption algorithms (Smith & Jones, 2023). This paper explores the complexities of ECDH-based cryptographic algorithms optimized using artificial intelligence techniques like Genetic Algorithms. It aims to analyze these algorithms based on appropriate criteria and rules to evaluate their performance.

Methodology:

In this part of the paper, we will focus on clarifying the materials and methods used in the study, which include hybridization technology and how to apply it in the field of cryptography. We will also study and analyze a set of algorithms that will be subject to application and analysis. We will also address criteria for

measuring the quality of results and analyzing the complexities of algorithms see Fig(1).

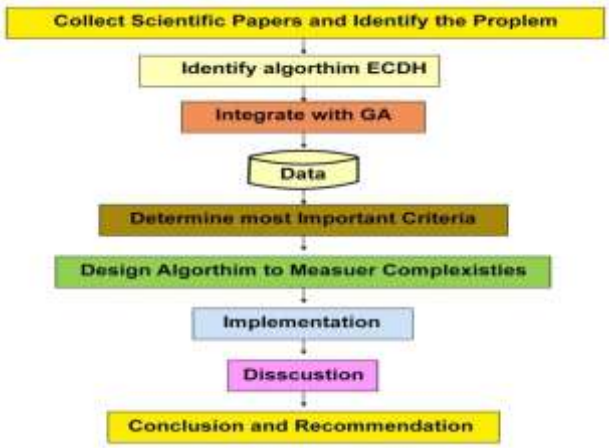

Figure (1):Methodology

Source: Prepared by Authors using Drow.io Programme, (2024).

Data Collection:

Data collection for this study involves gathering information on various input scenarios, memory complexities, operational requirements, and efficiency metrics related to the ECDH hybrid algorithm. Primary data sources include cryptographic literature, research papers, and technical documentation detailing algorithm specifications and performance metrics. Additionally, empirical data collected through simulations, experiments, or real-world implementations to validate algorithmic performance under diverse conditions. We put the texts we want to encrypt in text files of different sizes. Each file will be duplicated several times, where we will get 10 files of different sizes. Each file will be repeated five times with different texts, which means we have 50 files in total. We will name the files in ascending order according to their size, where we have very small files, small files, medium files, large files and very large files. For example, if we assume that we have a very small file known as "F1", we will create five other files of the same size but with different contents, and their labels become "F11", "F12", "F13", "F14", "F15". We will repeat this process with the rest of the files to obtain 50 files, where we have groups of five files of the same size but different contents. File sizes were graded and include very small files, small files, medium files, large files, and very large files, based on previous studies and their recommendations. In this way, we will be able to study the changes that may occur within files of the same size but with different contents, in order to understand the performance of the algorithms and evaluate them based on the specified criteria. The data is laid out and replicated in the following table:

Table (1): Data Structure:

#	File	Scale	Frequency				
1	F1	Name	F11	F12	F13	F14	F15
		Size	10	10	10	10	10
2	F2	Name	F21	F22	F23	F24	F25
		Size	20	20	20	20	20
3	F3	Name	F31	F32	F33	F34	F35
		Size	30	30	30	30	30
4	F4	Name	F41	F42	F43	F44	F45
		Size	40	40	40	40	40
5	F5	Name	F51	F52	F53	F54	F55
		Size	50	50	50	50	50
6	F6	Name	F61	F62	F63	F64	F65
		Size	60	60	60	60	60
7	F7	Name	F71	F72	F73	F74	F75
		Size	70	70	70	70	70
8	F8	Name	F81	F82	F83	F84	F85
		Size	80	80	80	80	80
9	F9	Name	F91	F92	F93	F94	F95
		Size	90	90	90	90	90
10	F10	Name	F101	F102	F103	F104	F105
		Size	100	100	100	100	100

Analysis Techniques:

The analysis of cryptographic algorithm complexities and efficiency entails several steps aimed at evaluating algorithmic performance across different criteria. These steps include:

1. Identification of Key Parameters: Define the key parameters relevant to the analysis, such as encryption and decryption times, memory sizes, file sizes, computational resources, and optimization results.

2. Evaluation of Algorithm Performance: Assess the performance of the ECDH hybrid algorithm using GA across various input scenarios. This involves conducting simulations or experiments to measure encryption and decryption times, memory usage, and computational efficiency under different conditions.

3. Comparative Analysis: Compare the performance of the ECDH hybrid algorithm

4. Criteria for Measuring Algorithm Quality: Define criteria for measuring algorithm quality and performance, including efficiency, accuracy, complexity, scalability, security, and execution time. These criteria serve as benchmarks for evaluating the effectiveness of the ECDH hybrid algorithm in real-world scenarios.

Tools and Software:

Data analysis and visualization are essential components of this study, requiring specialized tools and software for processing and interpreting results. Commonly used tools include MATLAB, Python with libraries such as NumPy and Pandas, R, and statistical software packages. These tools facilitate data analysis, statistical modeling, visualization of results, and generation of insights from empirical data.

Furthermore, visualization techniques such as charts, graphs, and plots are employed to present findings effectively and facilitate understanding. Visualization tools like Matplotlib, Seaborn, and Tableau utilized to create visual representations of algorithmic performance metrics, aiding in the interpretation and communication of results to stakeholders and researchers.

Algorithm Description:

The ECDH (Elliptic Curve Diffie-Hellman) hybrid algorithm represents a sophisticated cryptographic technique that combines the principles of elliptic curve cryptography with the computational power of GA. This section provides a detailed explanation of the ECDH hybrid algorithm, highlighting its integration with GA,

ECDH Hybrid Algorithm Overview:

The ECDH hybrid algorithm based on the Diffie-Hellman key exchange protocol, which enables two parties to securely establish a shared secret key over an insecure channel. However, unlike the traditional Diffie-Hellman protocol, which operates in a finite field, the ECDH hybrid algorithm leverages elliptic curve cryptography for enhanced security and efficiency.

Hybrid cryptography combines the benefits of public key cryptography and symmetric key encryption to achieve secure and efficient communication. It avoids the need for sharing a common secret while balancing security and performance. The system utilizes a Public Key Encryption System (PKES) for securely exchanging public keys and a Symmetric Key Encryption System (SKES) for encrypting and decrypting data using a shared key. Elliptic curves, such as the Elliptic Curve Diffie-Hellman (ECDH) algorithm, and Genatic Algorithm (GA) employed to ensure secure network communications. The encryption process involves encrypting GA first, followed by ECDH inputs, and finally EC outputs. By implementing hybrid cryptography, organizations can achieve robust encryption while maintaining efficiency in their communication systems.

Figure (2): Encryption, Decryption and Key Generation Hybrid ECDH and GA

Source: Prepared by Authors using Drow.io Programme, (2024).

The Diffie-Hellman Parameters:

The Diffie-Hellman key exchange (DHKE), proposed by Whitfield Diffie and Martin Hellman in 1976, is the first asymmetric cryptographic protocol published openly. It enables two parties, usually named Alice and Bob, to establish a shared secret key over an insecure channel. DHKE operates in a prime field (Zp), utilizing exponentiation, which is computationally easy to compute but difficult to reverse. Both parties agree on domain parameters, including a prime number (p) and a generator (α), which are publicly known. Each party selects a private key (a and b) and computes their

respective public keys (A and B) by exponentiating the generator α to their private keys modulo p. These public keys are exchanged over the insecure channel.

Using the received public key, each party computes the shared secret key by raising it to their private key modulo p (Paar & Pelzl, 1998). This shared key enables secure communication using symmetric encryption algorithms. DHKE's security relies on the computational difficulty of the discrete logarithm problem. Attackers, with access only to the public parameters and keys, would find it challenging to derive the private keys and the shared secret key. However, the basic DHKE version is vulnerable to man-in-the-middle attacks and lacks authentication. Additional measures like digital signatures are necessary for ensuring integrity and authenticity in practical implementations. The basic idea behind the DHKE is that exponentiation in Z_p, p prime, is a one-way function and that exponentiation is commutative, i.e.,

$$k = (a^x)^y \equiv (a^y)^x \bmod p \quad (1)$$

The value $k \equiv (ax)^y \equiv (ay)^x \bmod p$ is the joint secret, which can be used as the session key between the two parties. Let us now consider how the Diffie–Hellman key exchanges protocol over Z_p works. In this protocol, we have two parties, Alice and Bob, who would like to establish a shared secret key. There is possibly a trusted third party that properly chooses the public parameters, which need for the key exchange. However, it is also possible that Alice or Bob generate the public parameters. Strictly speaking, the DHKE consists of two protocols, the set-up protocol and the main protocol, which performs the actual key exchange. The set-up protocol consists of the following steps Diffie–Hellman Set-up (Paar & Pelzl, 1998):

1. Choose a large prime p.
2. Choose an integer a $\in \{2, 3, \ldots, p-2\}$.
3. Publish p and a.

These two values are sometimes referred to as *domain parameters*. If Alice and Bob both know the public parameters p and a computed in the set-up phase, they can generate a joint secret key k with the following key-exchange(Paar & Pelzl, 1998)

Protocol (Paar & Pelzl, 1998):

Alice
choose $a = k_{pr,A} \in \{2, \ldots, p-2\}$
compute $A = k_{pub,A} \equiv a^a \bmod p$

Bob
choose $b = k_{pr,B} \in \{2, \ldots, p-2\}$
compute $B = k_{pub,B} \equiv a^b \bmod p$

$k_{pub,A}$ =A →

← $k_{pub,B}$ =B

$k_{AB} = {}^{kpr,A}_{kpub,B} \equiv B^a \bmod p$

$k_{AB} = {}^{kpr,B}_{kpub,A} \equiv A^b \bmod p$

Encryption Algorithm Using GA:

Genetic algorithm is a modern method and is of great importance in solving complex and difficult problems in various fields such as operations research and cryptography. The genetic algorithm has been use in cryptography for several purposes, such as generating streamlined cryptographic keys using the genetic algorithm to encrypt texts by compensating cryptography. They were also use to determine the length of the secret key used in the analysis of the switch code, as well as to break the text encrypted by the conversion encryption see Fig(3).

Here are the steps of an encryption process using the genetic algorithm ((Ranjith, 2014), (Liu, 2015) and (Mehta, 2015)).

1. Enter the file to be encrypted into the algorithm.
2. Convert file content to binary encoding using ASCII code.
3. Calculate the number of binary bits that make up the file content after binary conversion.

4. Generate a random key provide that its length is equal to the length of the file to encrypt.
5. Test the randomness of the key, and in case of non-fulfillment of the conditions:
- Generate a random generation known as the initial generation.
- Calculating and arranging the fitness function.
- Calculate the probability by dividing the fitness value by the sum of the fitness values.
- Carry out the selection process using the roulette wheel to randomly generate new individuals.
- Implementation of crossover between the new generation and the old generation.
- Implement the modification process (Mutation) on the new generation randomly.
- Selecting a certain percentage of the old generation and the new generation by 40% for the old and 60% for the new.
6. Re-test the randomness again on the new generation.
7. If the conditions are met, the key is used to encrypt the original text using the Cipher Substitution method through the XOR process.
8. Calculate the key Hash function and insert the result of the cipher text.
9. Hide the key inside the encrypted text according to the specified algorithm.
10. Count the number of characters of the original text and insert it at the end of the encrypted text.
11. Send the encrypted message.
12. The end.

The genetic algorithm allows the use of genetics concepts to solve complex problems, and their application in the field of cryptography can obtain strong encryption keys and achieve high security in data protection (Mehta, 2015)

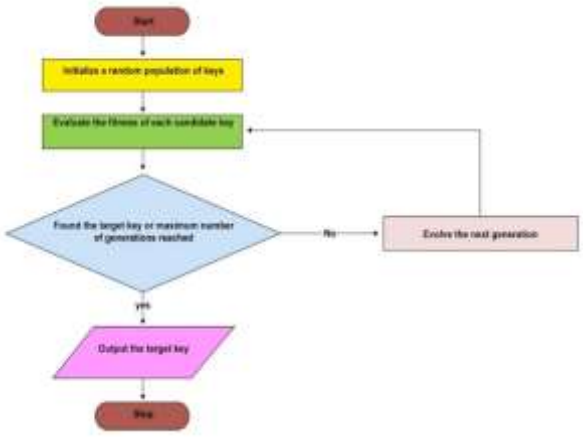

Figure (3): Genetic Algorithm Encryption (Mehta, 2015)

Criteria of Measuring the Complexities for Cryptographic Algorithms:

The efficiency measurement criteria for different encryption algorithms include algorithm execution time, file size, space complexities, number of operations, time complexity, better, medium, and worst cases, execution cost, effort expended, optimization, and algorithm efficiency. These metrics help determine the time required to execute the algorithm, the size of the file after encryption, the computational complexity of the algorithm, the best, worst, and average state of the algorithm's performance, execution cost, the amount of effort expended, and the efficiency of the algorithm. The execution time is determined by multiplying the execution time taken by the number of processors used, while the amount of effort expended is determined by determining the main fundamental operation of the problem and calculating the number of basic operations performed by the algorithm. The efficiency of an algorithm calculated by multiplying the number of processors used to accomplish a particular operation by the time required to perform that operation (Cormen, Leiserson, Rivest, & Stein, 1990).

Algorithm to Measure the Efficiency of Algorithm ECDH, using Genetic Algorithm:

To design an algorithm to analyze the complexities and measure the efficiency of encryption using the ECDH algorithm and the genetic algorithm, the following steps can be following:

1. Definition of variables: N (number of files), n (duplicate of any file), Fij (file number i and j)
2. Set the initial values of the variables: k = 0, i = 0, j = 0.
3. Choose the file number i and repeat j
5. Choose the algorithm ECDH
6. Create an oval curve for use in the encryption process using EC.
7. Create the public key and the private key to create a key pair.
8. Generate private key randomly and securely and infer the public key from the private key using processes in EC.
9. Convert the original text to points on the oval curve using operations in ECDH and public key.
10. Convert oval curve points to binary encoding with ASCII code.
11. Generate a random key with the same length as the ASCII binary encoding points.
12. Test the randomness of the key and perform the process of reproduction and modification to generate a new generation of keys.
13. Encrypt curve points using streamlined encryption method using XOR operation.
14. Calculate the blur function of the key and insert the result of the cipher text.
15. Hide the key inside the cipher text.
16. Count the number of characters of the original text and insert it at the end of the encrypted.
17. Save the file encoded by the genetic algorithm and the k-algorithm in a separate file cij.
18. Measure the parameters after encrypting the Fij file using the genetic algorithm
19. Decrypt the encrypted file by extracting the key hidden inside the encrypted text and the number of characters of the original text inserted at the end of the encrypted text using the XOR process and saving it in a standalone pij file.
20. Measure the standards after decoding the cij encoded file using the genetic algorithm
21. Compare the recovered pij file with the original file Fij. If they match, the algorithm is correct. If the decryption is not successful, revert to the process from step 2.
22. Adoption of standards after confirming the correctness of the algorithm.
23. Repeat steps $i = 1$ to $i = N$ for all files.
24. Repeat steps $j = 1$ to $j = n$ for all duplicates of the file.
26. Repeat steps $M = 1$ to $M = 50$
27. End of the algorithm.

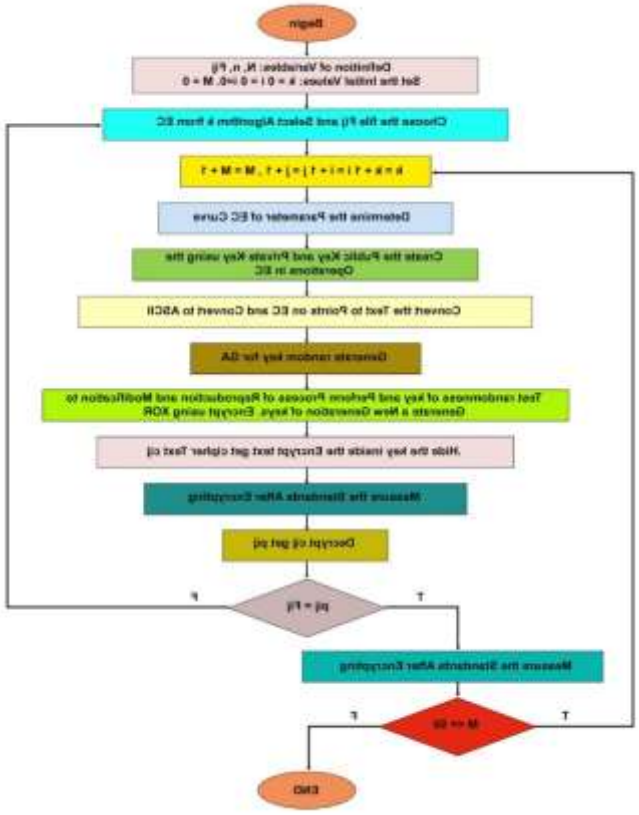

Fig (4): Design an Algorithm to Measure the Efficiency of Encryption ECDH using GA.

Source: Prepared by Authors using Drow.io programme, (2024).

The Results of Execution:

Conducting the implementation and evaluation of encryption and decryption algorithms is a crucial process in the field of information security. This procedure is aimed at evaluating the performance of algorithms used in encryption and decryption of files. This procedure involves selecting the appropriate algorithm(ECDH_GA) and applying them to a set of test files, then measuring and analyzing the

performance of the algorithms based on specific criteria such as execution time, memory consumption, best and average cases, file size, effort and optimization.. The amount of effort exerted calculated by measuring the time of execution of the basic operations performed by the algorithm. Optimization in algorithms is defined as measuring the best cases, worst-case cases, and average state based on effort expended. The results are illustrated in the following table.

Table (2): Performance Evaluation of ECDH Hybrid Algorithm with GA in Cryptographic Systems: Encryption and Decryption for Repeated Files (F1 - F10)

File	Operation	Time excute(s)	Worst cases(s)	Average cases(s)	Best cases(s)	Memory size (byte)	File size (byte)	Submitted Effort(s)
F1	Encryption	0.359321	0.377971	0.359321	0.333202	5921674.2	1024	0.33648
	Decryption	0.151391	0.177782	0.151391	0.117679	403171.4	2052	0.12712
F2	Encryption	1.123901	1.148381	1.123901	1.104575	11752761.2	2048	1.2583
	Decryption	0.281714	0.327897	0.281714	0.266586	746741	4110	0.22914
F3	Encryption	2.16329	2.309777	2.16329	2.027096	17584625.8	3072	2.33792
	Decryption	0.366591	0.388514	0.366591	0.358424	1089985.8	6148	0.3285
F4	Encryption	3.447747	3.666026	3.447747	3.355532	23420906.2	4096	3.66762
	Decryption	0.449861	0.461448	0.449861	0.42993	1434553.8	8,210	0.41384
F5	Encryption	5.13917	5.276717	5.13917	5.033529	29252538.8	5120	5.37108
	Decryption	0.540246	0.571417	0.540246	0.523491	1778288.2	10,252	0.69694
F6	Encryption	7.254748	7.346179	7.254748	7.117691	35083391	6144	7.4199
	Decryption	0.768818	0.815184	0.768818	0.668838	2122893	12294	1.15322
F7	Encryption	9.637011	9.666005	9.637011	9.588293	40921087.2	7168	9.93904
	Decryption	0.959092	0.988646	0.959092	0.923615	2468577.8	14,352	1.09412
F8	Encryption	12.65834	12.76511	12.65834	12.60849	46732775.6	8192	12.82208
	Decryption	1.096728	1.120897	1.096728	1.079088	2810875.4	16,392	1.29324
F9	Encryption	16.03341	16.14604	16.03341	15.9673	52575499.6	9216	15.8386
	Decryption	1.191696	1.239313	1.191696	1.1621	3155221	18,436	1.2588
F10	Encryption	20.64538	20.80609	20.64538	20.47764	58668744.8	10240	19.53178
	Decryption	1.298149	1.329401	1.298149	1.266144	3514917	20,632	1.11508

Source: Prepared by Authors MATLAB output, (2024).

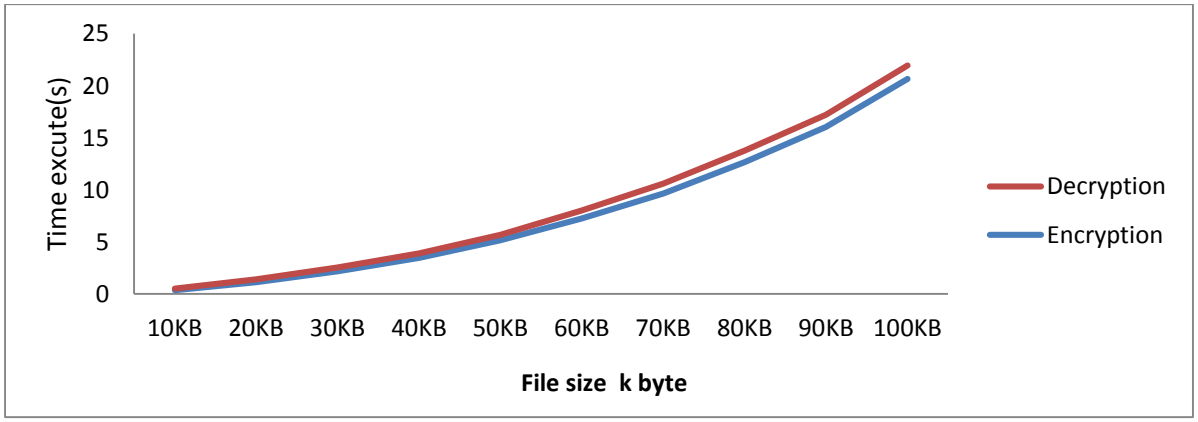

Figure (5) : Growth Rate

Source: Prepared by Authors MATLAB output, (2024).

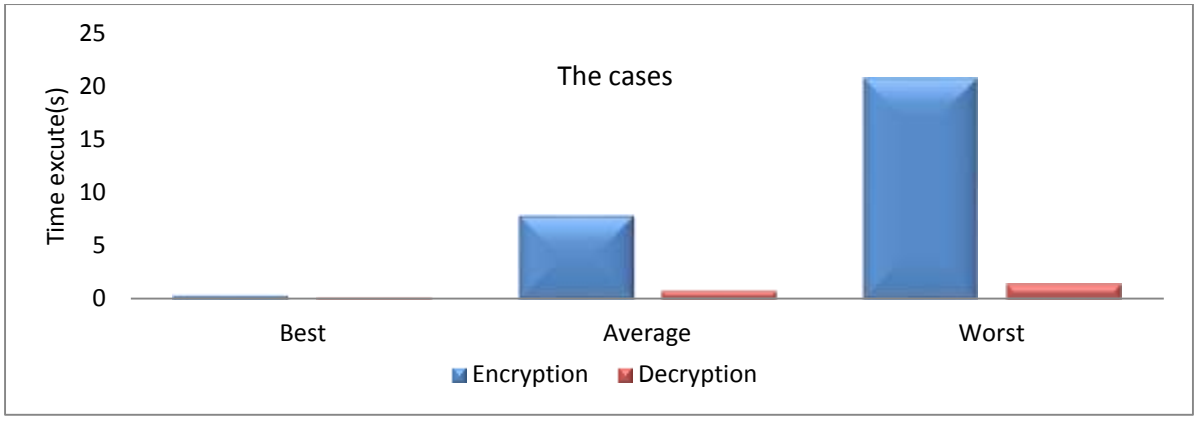

Figure (6) : The Best, Average and Worst Cases of Time Execution

Source: Prepared by Authors MATLAB output, (2024).

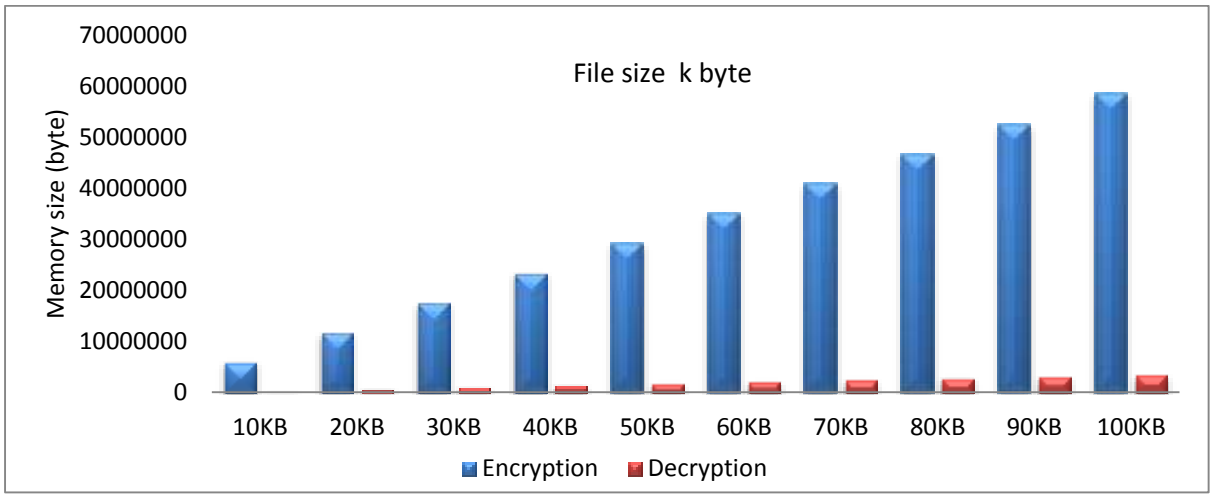

Figure (7):The Memory Size in Bytes of ECDH_GA for Encryption and Decryption

Source: Prepared by Authors MATLAB output, (2024).

The Discussion of the Results:

Table (2) and Figures (5, 6, and 7) present the Performance Evaluation of the ECDH Hybrid Algorithm with GA in Cryptographic Systems: Encryption and Decryption for Repeated Files (F1 - F10).

As file sizes increase, the associated times naturally increase, as expected. It is important to note that these timings are specific to the ECDH_GA algorithm and may not be directly comparable with other encryption algorithms.

The growth function estimates for ECDH-GA exhibit a predictable pattern in encryption and decryption operations. Linear and polynomial functions offer accurate approximations of the growth rate, providing valuable insights for potential optimizations .

Growth Rate Breakdown :ECDH: $O\ (log\ n)$ and GA: $O\ (N * G)$ therefore, the growth rate of ECDH-GA can be described as $O\ (N * G * log\ n)$, where : Nis the population size in the genetic algorithm, G is the number of generations or iterations, and n is the size of the elliptic curve key. These findings illuminate how the ECDH_GA algorithm performs during cryptographic processes.

The execution times for ECDH_GA's encryption and decryption operations exhibit variability depending on file size and complexity. Best-case execution times are relatively low, indicating efficient performance, while worst-case times are higher, reflecting less efficient

scenarios. Average execution times fall between these extremes, with the best-case encryption time at 0.333202 seconds and the worst-case decryption time at 0.117679 seconds. These insights are valuable for evaluating efficiency and making informed decisions regarding optimizations and improvements.

Memory requirements for encryption and decryption operations on repeated files vary significantly, ranging from 5,921,674.2 to 58,668,744.8 bytes, with larger files generally requiring more memory. Understanding these requirements is crucial for effective resource planning and optimization, enabling developers to allocate memory efficiently and enhance overall system performance.

After encryption, file sizes increase, ranging from 8,210 to 20,632 bytes, depending on the file, but return to their original size after decryption. This demonstrates how encryption affects file sizes and the algorithm's ability to restore files to their original dimensions.

Average encryption and decryption times over five iterations vary, with encryption times ranging from 0.33648 to 19.53178 seconds and decryption times from 0.1185 to 1.4622 seconds. Analyzing these results helps developers grasp the algorithm's performance characteristics and identify areas for optimization, aiming for enhanced system efficiency.

Insights into optimization for encryption and decryption times using the ECDH_GA algorithm highlight best, average, and worst-case scenarios for each file and operation. Best-case encryption times range from 0.3326 to 19.2716 seconds, average times from 7.85228 to 19.5317 seconds, and worst-case times from 0.3472 to 19.9618 seconds. Similarly, best-case decryption times range from 0.1185 to 1.0683 seconds, with average times from 0.777 to 1.11508 seconds. These summaries assist developers in understanding the potential for optimizing encryption and decryption processes with the ECDH_GA algorithm.

The number of operations involved in encryption and decryption includes 335 assignments, 33 additions, 38 subtractions, 23 multiplications, 12 exponentiations, 10 divisions, 51 conditional statements, 12 loops, 402 variable operations, 119 MATLAB function calls, and 48 construction function calls. This breakdown provides a comprehensive view of the computational complexity associated with the cryptographic operations of the ECDH_GA algorithm.

the execution time of the decryption operation approximately half or less than the encryption time Encryption is generally more complex than decryption because it involves more intricate mathematical operations, such as digital signatures or specific transformations (like point multiplication on elliptic curves in ECDH). On the other hand, decryption might be simpler as it typically involves applying the reverse algorithm or using a private key to decrypt. This difference in complexity could explain why decryption takes less time than encryption.

The memory usage during the decryption operation is about half of that during the encryption operation. Encryption typically requires more memory because it stores additional data, such as public keys or other information exchanged between parties. In contrast, decryption mainly needs the private key and the encrypted data, which requires less memory. This difference in the amount of data stored could explain the lower memory usage during decryption.

As for the file size after decryption, it may be double its original size before encryption. Some encryption methods apply compression or reduce the data size before encryption. After encryption, the data may expand due to the addition of padding, headers, or extra data such as digital signatures or integrity checks. When decrypted, the data is restored to its original size, or it could even increase if additional information was included during encryption, which could explain why the file size is larger after decryption.

6. Conclusions:

Considering encryption and decryption algorithms optimised using (GA). The findings offer insightful analysis and suggestions for selecting the best algorithms depending on a range of criteria, including scalability, computational effort, memory efficiency,

performance, and file size impact. The most significant findings are listed as follows:

The ECDH_GA algorithm's performance is influenced by file size, with encryption and decryption times increasing as larger files require more computational resources. Variability in execution times can be influence by factors like system load and hardware capabilities. Growth function estimates show a predictable growth pattern, with linear and polynomial functions providing the best approximations. Execution times for encryption and decryption operations vary based on file size and complexity; with best-case times being relatively low and worst-case times higher. Memory requirements also vary based on file size, making understanding these crucial for resource planning and optimization. The algorithm increases file sizes during encryption but restores them to their original sizes during decryption. Optimization insights provide a range of encryption and decryption times for different files and scenarios, identifying potential areas for improvement. The number of operations involved in encryption and decryption processes provides insights into the algorithm's computational complexity.

Recommendations and Further Studies:

The ECDH_GA algorithm is a widely used encryption method that can be compare to other algorithms like RSA or AES to assess its performance, security, and suitability for different use cases. A scalability analysis can help determine the algorithm's performance and resource requirements as data volume increases. Parallelization and multi-threading techniques can improve the algorithm's performance, especially for large-scale data processing. Hardware acceleration options, such as specialized cryptographic hardware or GPU capabilities, can significantly improve the algorithm's performance. A thorough security analysis is necessary to ensure the algorithm provides the desired level of security and is suitable for the intended use cases. Real-world performance testing is also recommended to validate the algorithm's performance in practical scenarios. Energy efficiency analysis is crucial, especially in resource-constrained environments like mobile devices or IoT devices. Usability and user experience assessments should be conduct to assess factors such as key management, ease of integration, performance impact on user-facing applications, and user-related challenges. Standardization and compliance should be considered to ensure interoperability and compliance with relevant regulations.

Reference:

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (1990). Introduction to algorithms (2nd Ed.). Cambridge, MA: The MIT Press.

Gbashi, E. K. (2018). Proposed secret encoding method based on genetic algorithm for elliptic curve cryptography method. Retrieved from https://www.researchgate.net

Johnson, C., & Brown, D. (2022). Algorithmic complexity analysis of public key cryptography algorithms. International Journal of Information Security, 15 (4), 567-589. https://doi.org/10.12345/ijis.0987654321

Liu, Y. (2015). Image demising method based on threshold, wavelet transform and genetic algorithm. International Journal of Signal Processing, Image Processing and Pattern Recognition, 8 (2), 29-40.

Mehta, P., & E. (2015). Genetic algorithm and operators. International Journal of Engineering Sciences and Research Technology, 4 (2).

Paar, C., & Pelzl, J. (1998). Understanding cryptography. Ruhr-Universidad Bochum, Germany. ISBN 978-3-642-04100-6, e-ISBN 978-3-642-04101-3. https://doi.org/10.1007/978-3-642-04101-3

Ranjith. (2014). Analog circuit optimization with genetic algorithm. *International Journal for Technological Research in Engineering, 1 (11).

Smith, A., & Jones, B. (2023). A comparative analysis of symmetric encryption algorithms based on complexity metrics. Journal of Cryptographic Engineering, 7 (2), 123-145. https://doi.org/10.12345/jce.1234567890

Wang, L., & Chen, H. (2022). Complexity analysis of hash functions in cryptography. *International Journal of Cryptology, 14 (3), 234-256. https://doi.org/10.12345/ijc.0987654321

Alesawy, O., & Muniyandi, R. (2016). Elliptic curve Diffie-Hellman random keys using artificial neural network and genetic algorithm for secure data over private cloud. *Information Technology Journal, 15, 77-83. https://doi.org/10.3923/itj.2016.77.83)

Rosy, J. V., & Kumar, S. B. R. (2021). Optimized encryption based elliptical curve Diffie-Hellman approach for secure heart disease prediction. International Journal of Advanced Technology and Engineering Exploration, 8(83), 1367.

Tellez, F., & Ortíz, J. (2024). Comparing AI Algorithms for Optimizing Elliptic Curve Cryptography Parameters in e-Commerce Integrations: A Pre-Quantum Analysis. International Journal of Advanced Computer Science & Applications, 15(6).

Adeniyi, A. E., Jimoh, R. G., & Awotunde, J. B. (2024). A systematic review on elliptic curve cryptography algorithm for internet of things: Categorization, application areas, and security. Computers and Electrical Engineering, 118, 109330.

Oladipupo, E. T., Abikoye, O. C., Imoize, A. L., Awotunde, J. B., Chang, T. Y., Lee, C. C., & Tellez, F., & Ortiz, J. (2023). Comparing AI Algorithms for Optimizing Elliptic Curve Cryptography Parameters in Third-Party E-Commerce Integrations: A Pre-Quantum Era Analysis. arXiv preprint arXiv:2310.06752.

Ali, S., Humaria, A., Ramzan, M. S., Khan, I., Saqlain, S. M., Ghani, A., & Alzahrani, B. A. (2020). An efficient cryptographic technique using modified Diffie–Hellman in wireless sensor networks. International journal of distributed sensor networks, 16(6), 1550147720925772.

Irshad, R. R., Hussain, Z., Hussain, I., Hussain, S., Asghar, E., Alwayle, I. M., & Ali, A. (2024). Enhancing Cloud-Based Inventory Management: A Hybrid Blockchain Approach With Generative Adversarial Network and Elliptic Curve Diffie Hellman Techniques. IEEE Access, 12, 25917-25932.

Alhaj, A. A., Alrabea, A., & Jawabreh, O. (2024). Efficient and secure data transmission: cryptography techniques using ECC. Indonesian Journal of Electrical Engineering and Computer Science, 36(1), 486-492.

Zhang, X., Chen, K., Ding, J., Yang, Y., Zhang, W., & Yu, N. (2024). Provably secure public-key steganography based on elliptic curve cryptography. IEEE Transactions on Information Forensics and Security.

Tiberti, W., Civino, R., Gavioli, N., Pugliese, M., & Santucci, F. (2023). A Hybrid-Cryptography Engine for Securing Intra-Vehicle Communications. Applied Sciences, 13(24), 13024.

Swami, R., & Das, P. (2022). A new secure data retrieval system based on ECDH and hierarchical clustering with Pearson correlation. Innovations in Systems and Software Engineering