Implementation and Analysis of the DES Algorithm: A MATLAB Approach

Nikeel Ramharakh (2433669)

School of Electrical & Information Engineering, University of the Witwatersrand, Private Bag 3, 2050, Johannesburg, South Africa

Abstract: This report dives into the implementation and analysis of the Data Encryption Standard (DES) algorithm using the Matlab Integrated Development Environment (IDE). DES is notoriously used as a symmetric-key block cipher. It remains foundational in cryptography despite updated encryption standards today. This report includes the method and reasoning behind the implementation, the results obtained compared to other peer reviewed results and an overall analysis of how breakable the ciphertexts are. After practical exposure to implementation of the DES algorithm, a high-level overview can be obtained. A brief discussion of challenges faced when optimizing DES in Matlab is included. Ultimately, this report offers insights into DES operation and its relevance in modern cryptography.

Key words: Cryptography, Data Encryption Standard (DES), Integrated Development Environment (IDE), Matlab.

1. INTRODUCTION

Cryptography is the science of communication security and is essential to protecting the integrity of sensitive data. DES is the fundamental basic symmetric-key block cipher that has been used for decades. Symmetric cipher refers to the cipher using the same key for encryption and decryption. DES is an old encryption and decryption method that revolutionised cryptography when brought to fruition. At the time, commercial cryptography was not reliable or safe. Thus, a standard was needed.

Decades later, DES is still useful in modern cryptography because it provides insights into basic encryption concepts and acts as a standard by which to compare more recent cryptographic algorithms. Matlab is used in this research to investigate DES implementation and experiment attributes of DES, such as the Avalanche effect. Through analysis and the development of a new model, the laboratory aims to shed light on how DES functions. The intention is to explore its advantages and disadvantages as well as its applicability in the context of modern cryptographic standards via testing and validity checks.

The addressing of various key objectives and questions are included in this report encompassing the implementation of both encryption and decryption algorithms. Experiments are conducted to validate the implementation against known plaintext and ciphertext pairs obtained from reputable online references. This helps confirm adherence to the DES standard.

Additionally, ciphertexts generated by the algorithm are analysed using classical cryptographic breaks such as frequency analysis to evaluate DES's resistance to such techniques. The efficacy of different keys for a consistent plaintext in encrypting data is examined. Discussions on associated vulnerabilities or weaknesses are included and an overall assessment is made of the performance of the DES algorithm.

2. BACKGROUND

DES was first developed in the early 1970s under the leadership of IBM cryptographer Horst Feistel and later used by the US government through the National Bureau of Standards (NBS). The NBS solicited proposals for cryptographic algorithms to protect data during transmission and storage [1]. For secure data transfer, DES has long been a cornerstone of contemporary cryptography. This cryptographic technique relies on a single shared key for both encryption and decryption processes, ensuring that the same key is used by both the sender and the receiver. This key is securely exchanged between the communicating parties prior to the encryption process [2]. Understanding DES is still crucial despite the age and eventual replacement by more sophisticated encryption protocols. Future improvements of DES will be determined after use in the Matlab environment through experimentation and evaluation.

2.1 Success Criteria for DES Implementation

A number of important goals must be met in order to successfully implement the DES algorithm. First and foremost, the implementation must faithfully mimic the DES encryption and decryption procedures as outlined in the algorithm's specification. Splitting of the permutation of the key and input data and final permutation procedures must all be handled correctly. Substitution tables should be accurately encoded to rid the risk of error.

In addition, the solution must exhibit resilience by safely encrypting and decrypting information. This guarantees that the original hexadecimal message can be precisely extracted from the ciphertext. The solution outcomes must also be efficiently executable with minimal memory usage. Finally, the implementation must endure cryptographic attacks like differential and brute force cryptanalysis [3]. This demonstrates the shear strength against sensitive data being compromised and other security risks.

2.2 Significance of the Key

The key serves as a crucial parameter that determines the conversion of plaintext into ciphertext during encryption and vice versa during decryption. It is a binary string of 56- bits. The significance of the key lies in its role in ensuring confidentiality and security of the encrypted data. Security lies in the key. By altering the key, users can generate a vast number of different encryption schemes. This makes it computationally difficult for protected data to be decrypted by unknown and unauthorised adversaries without knowledge of the correct key. Therefore, the key in DES is fundamental in providing a secure communication channel for sensitive information transmission. Moreover, it allows encryption and decryption to be simplified, repetitive and is therefore adequate for hardware implementation.

3. ALGORITHM IMPLEMENTATION

The Data Encryption Algorithm (DEA) and Data Decryption Algorithm (DDA) were developed using a total of two functions and two .m files in which the different functions were called. Each .m-file is responsible for the encryption and decryption algorithms respectively.

3.1 Subkey Generation and Key Scheduling

The initial 64-bit hexadecimal key undergoes conversion into binary and an initial permutation to obtain a 56-bit binary key which satisfies the system requirements. This conversion removes the 8th bit for every 8-bytes being the parity bits. Subkey generation is achieved through first splitting the 56-bit key into two 28-bit halves. Each half undergoes round-dependant circular shifting operations leftwards to produce 16 subkeys. Each 48-bit subkey is used in a different round of encryption. There is a compression permutation that allows the two 28-bit halves to become a 48-bit subkey. Key scheduling ensures that each round of encryption uses a unique and distinct subkey and thus enhancing the overall security of the encryption process. In the Matlab function, a cell array is employed to hold all 16 subkeys. The reason for this design choice will later be explored.

3.2 The Inputted Plaintext

The input plaintext in the DES algorithm is provided in hexadecimal format in which each character represents a nibble of data. This hexadecimal representation allows for a compact and readable way to express binary data. Before encryption or decryption, the hexadecimal plaintext is converted into its binary equivalent, ensuring compatibility with the DES algorithm, which operates on binary data. This conversion process involves decoding each hexadecimal character into its corresponding 4-bit binary representation, resulting in a binary plaintext that can be processed by the Data Encryption Algorithm.

3.3 The Data Encryption Algorithm

This function is the core of the encryption and decryption operations of the DES algorithm. Passing in a 64-bit input, it immediately separates it into two left and right 32-bit inputs. Application of the expansion permutation (E-box) to the right half of the 64-bit input takes the right half from 32-bits to 48-bits. This is followed by a bitwise XOR operation between this expanded half and the subkey for that specific round. The result for this XOR operation will then experience substitution table (S-box) handling. Each S-box substitutes eight 6-bit input blocks with 4-bit outputs based on predefined substitution tables. After substitution, the function applied a final permutation (P-box) to the resulting 32-bit block. Finally, the result from this was XORed the left half of the 64-bit input. The result of this XOR operation becomes the right half of the input for the next round and the right half from the previous round becomes the left half for the input of the next round. This process is repeated for each round except the final round of the DES algorithm, with the number of rounds determined by the encryption or decryption process, normally being 16. In the 16th and final round, the reverse occurs. The result of the XOR operation becomes the left half of the output and the right half of the output is the previous round's left input half.

3.4 The Data Decryption Algorithm

The DDA uses the established DEA function and subkey generation function. The .m-file differs only slightly from the encryption .m-file. The encryption utilizes a for loop to increment from one to sixteen rounds. The converse occurs for the decryption with the subkey generation function taking in an index that decreases from 16 rounds to the one. The overall for loop still increments from one to sixteen rounds. The other subtle change involves the renaming of the plaintext variables into ciphertext variables to better distinguish input from output. Therefore, the functionality of the algorithms are virtually the same except for the round increments.

4. OUTCOMES OF LABORATORY EXPERIMENTATION

4.1 Subkey Generation Function

A given 16 character hexadecimal key input is converted and depicted in 64-bit binary. This binary representation is passed through an initial permutation table. After which, the `generatesubkey.m` function is called receiving the permuted 56-bit input key and the index. The .m-file with the developed function is included for ease of reference.

4.2 *Unique Subkeys*

During the testing of three keys to ascertain whether they would generate distinct subkeys, the following results were obtained. The blocks of subkeys can be located in the Appendix.

- The observation that the input 1F1F 1F1F 0E0E 0E0E yielded only one unique subkey which indicates a weakness in the encryption process. A single unique subkey suggests a lack of robustness in the encryption algorithm, potentially enabling attackers to exploit patterns or vulnerabilities in the encryption scheme.

- The discovery that the input 1FFE 1FFE 0EFE 0EFE resulted in two unique subkeys over sixteen rounds indicates a condition known as semi-weakness in the encryption process. While semi-weak keys offer more security than weak keys, they still pose a vulnerability in the encryption algorithm.

- The outcome of obtaining four unique subkeys for sixteen rounds from the input 1FFE FE1F 0EFE FE0E suggests a relatively stronger encryption compared to scenarios with fewer unique subkeys.

4.3 *Utilizing a 64-bit Input and Subkey to Produce Two 32-bit Output Blocks*

The `DEA.m` function performs the expansion permutation, XOR operation with the subkey, substitution using S-boxes, permutation using a P-box, and final XOR operations to produce the output blocks. For the given input 133457799BBCDFF1 the function produces two 32-bit output blocks, with the first block being 85E81354 and the second block being 0F0AB405 after completing sixteen rounds of the encryption process. It's noteworthy that the index `'i'` in the function can be adjusted to correspond to any round value.

5. TESTING VALIDITY OF RESULTS

To facilitate this validation process, a table is be constructed below documenting the comparison between the provided ciphertexts and the corresponding plaintexts. These were obtained from credible sources with both encryption and decryption algorithms being tested. This rigorous examination aims to verify the algorithm's adherence to DES standards and enhance confidence in the reliability of the algorithm.

Table 1: Multi-sourced cross-referenced ciphertexts for various plaintexts and keys implementing DEA.

Plaintext	Key	Ciphertext
4E6F7720	01234567	3FA40E8A
69732074	89ABCDEF	984D4815
13345779	01234567	85E81354
9BBCDFF1	89ABCDEF	40F0AB405
4E6F7720	01234567	3FA40E8A
69732074	89ABCDEF	984D4815
123456AB	AABB0918	C0B7A8D0
CD132536	2736CCDD	5F3A829C
C0B7A8D0	01234567	123456AB
5F3A829C	89ABCDEF	CD132536

The first example plaintext was sourced from the Handbook of Applied Cryptography and consisted of the ASCII bytes of the phrase "Now is the time for all" [4]. A full conversion of this plaintext was accessible on the website and yielded identical ciphertext to that produced by the Matlab implementation. Subsequently, the next plaintext underwent testing via a conversion acquired from a reputable source [1]. Finally, a website offering a step-by-step conversion and implementation of the DES algorithm in Python was utilized for encryption and decryption of the final example inputs and outputs [5].

6. EVALUATION OF THE AVALANCHE EFFECT PARAMETER

The best way to describe the Avalanche effect would be to draw parallels to the butterfly effect. Small changes to the plaintext can have a significant change to the ciphertext. This relates to the diffusion and confusion applications to the algorithm. Diffusion was applied through many permutation tables and confusion was applied through the substitution tables. The purpose of the diffusion property is to ensure that each plain text bit affects numerous cipher text bits, dispersing the statistical structure of the cipher text. It is akin to several plaintext fragments influencing every ciphertext [6]. Similarly, the purpose of the confusion property is to guarantee a highly intricate statistical link between the key and the ciphertext [7]. To adequately demonstrate the effect in the implemented algorithm, the plaintext of 0123456789ABCDEF and key of 0000000000000000 were selected. The ciphertext obtained was 617B3A0CE8F07100. A singular bit change in the key being 1000000000000000 led to ciphertext being 3BEA022C038228AE. The significant transformation in the ciphertext showcases the Avalanche effect. A minor change in the key input yielded a substantial alteration in the output. This behaviour aligns with the anticipated standards for a secure encryption algorithm. Additionally, it's crucial to recognize that the observed output shows the presence of a high level of diffusion and confusion within the encryption algorithm.

7. CLASSICAL CRYPTOGRAPHIC BREAKS ASSESSMENT

A frequency analysis is performed to properly assess the robustness of the algorithm to classical cryptographical breaks. Five random and different keys are chosen and the plaintext to ciphertext transformation is represented in the table below. The sample plaintext used was 24336692346951AB.

Table 2: Frequency Analysis of five differing keys and the same plaintext

Plaintext	Key	Ciphertext
24336692	01234567	0B256E7A
346951AB	89ABCDEF	2560D454
24336692	FEDCBA98	7F7394FF
346951AB	76543210	8F91E14C
24336692	AABB0918	40ADD3F3
346951AB	2736CCDD	0D2F7EF3
24336692	3FA40E8A	8F374B37
346951AB	984D4815	5B98AA6E
24336692	2736CCDD	0DE2C98F
346951AB	5F3A829C	71C90660

Three figures were generated with each illustrating the frequency distribution for plaintext, keys, and ciphertext with varying keys while maintaining a constant plaintext. All figures are depicted in the Appendix and all table column variables were combined rather than being related row by row. Changes in the key were opted rather than changes in the plaintext to adequately see a key-ciphertext change correlation. Moreover, to examine the effect of using weak keys compared to stronger keys.

Interestingly, the peaks observed in the frequency distributions do not align. The ciphertext has a flatter distribution which shows DES's immunity to frequency analysis. This immunity can be attributed to the confusion introduced through the substitution tables employed in the DEA algorithm. The weak keys exhibited two prominent peaks similar to that of the plaintext, but lacked a coherent relationship. This further highlights the robustness of DES against classical cryptographic breaks like frequency analysis and potentially the Kasiski method.

8. CHALLENGES AND IMPROVEMENTS

Minor challenges in the development stage was changing between strings and arrays. Strings were mainly used as strings made the reading of the hexadecimal values easier and kept consistency among functions. However, computation became problematic as XOR operations were required to occur in a bitwise manner and many string conversions were made in the final stages of the `DEA.m` function. One of the main challenges encountered in the implementation was ensuring accuracy in the subkey generation process. During decryption, it was observed that incorrect subkeys were generated when the function was run for all sixteen rounds. Having the the subkey generation function run sixteen times to obtain each subkey in the overall for loop proved futile. It posed a significant obstacle to the decryption process as the wrong plaintext would be generated. The inconsistency in subkey generation likely occurred due to the design's complex interaction between the key schedule and the DEA.

To eliminate this issue, the decision was made to generate all subkeys upfront and store them in a cell array. By generating and storing all subkeys for all rounds, the decryption process became more efficient and streamlined. This improved the overall performance and reliability.

9. CONCLUSION

The DES implementation in Matlab showcased integrity, accuracy and resilience against classical cryptographic attacks such as frequency analysis. The refinement of the algorithm is highlighted by its immunity to simple monoalphabetic breaks and its ability to accurately portray the Avalanche effect. The strength of DES lies in the key, ensuring that encrypted data remains confidential and is easy to decrypt when in possession of the correct key. Ultimately, the implemented DES algorithm in Matlab met the real-world standards across multiple dimensions and this report further solidifies its position as a cornerstone in cryptography.

REFERENCES

[1] J. O. Grabbe. "The DES algorithm illustrated.", 2010.

[2] O. Reyad, H. M. Mansour, M. Heshmat, and E. A. Zanaty. "Key-Based Enhancement of Data Encryption Standard For Text Security." In *2021 National Computing Colleges Conference (NCCC)*, pp. 1–6. 2021.

[3] E. Biham and A. Shamir. "Differential cryptanalysis of DES-like cryptosystems." *Journal of CRYPTOLOGY*, vol. 4, pp. 3–72, 1991.

[4] A. Lamoureux and M. Bodewes. "64 des full example with all the stages.", Oct 2018.

[5] GfG. "Data Encryption Standard (DES): Set 1.", Sep 2023. URL `https://www.geeksforgeeks.org/data-encryption-standard-des-set-1/`.

[6] S. Ramanujam and M. Karuppiah. "Designing an algorithm with high avalanche effect." *IJCSNS International Journal of Computer Science and Network Security*, vol. 11, no. 1, pp. 106–111, 2011.

[7] R. S. Krovi and P. Jetty. "ANALYSIS OF AVALANCHE EFFECT–A MODIFIED DES CIPHER." *researchgate,(URL) https://www. researchgate. net/publication/286100145*.

Appendix

A Outcomes of Laboratory Experimentation

A1 Frequency Analysis

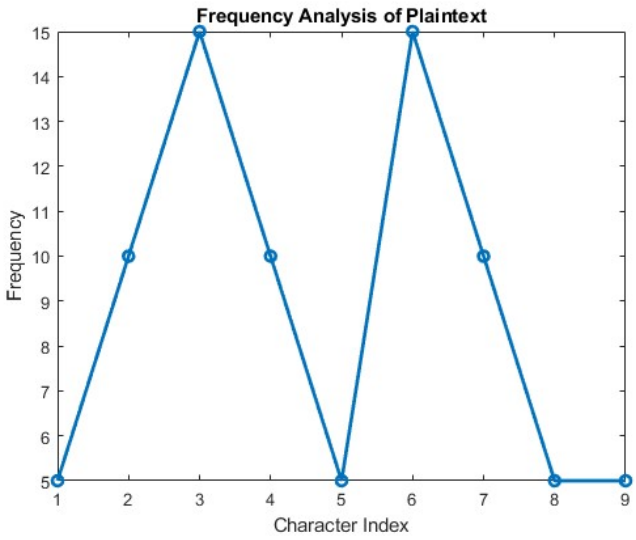

Figure 1: Frequency Distribution of Constant Plaintext

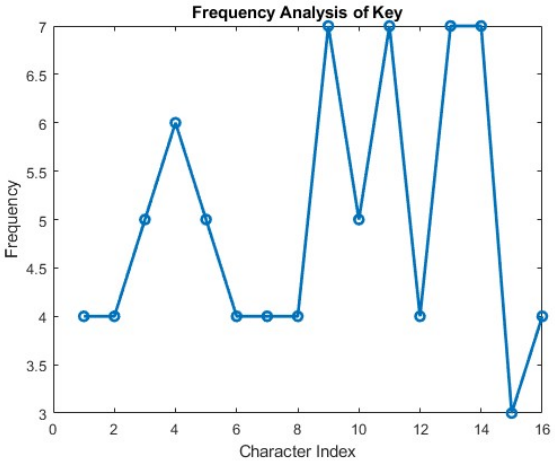

Figure 2: Frequency Distribution of Differing Encryption Keys

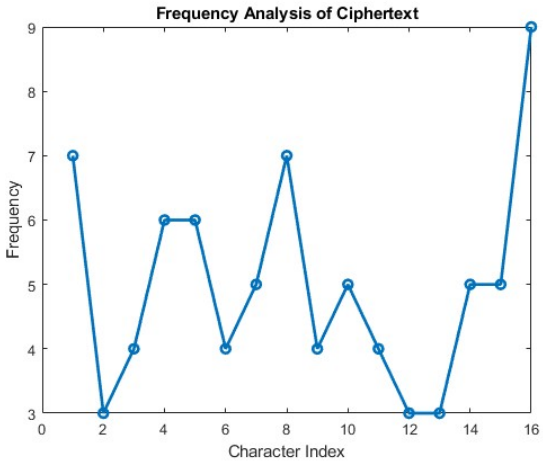

Figure 3: Frequency Distribution of Ciphertext

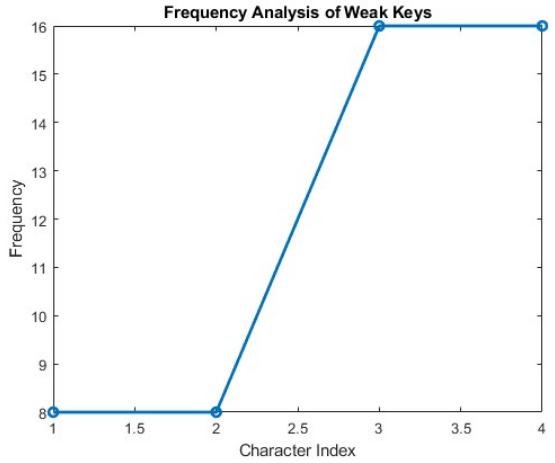

Figure 4: Frequency Distribution of Weak Encryption Keys Combined

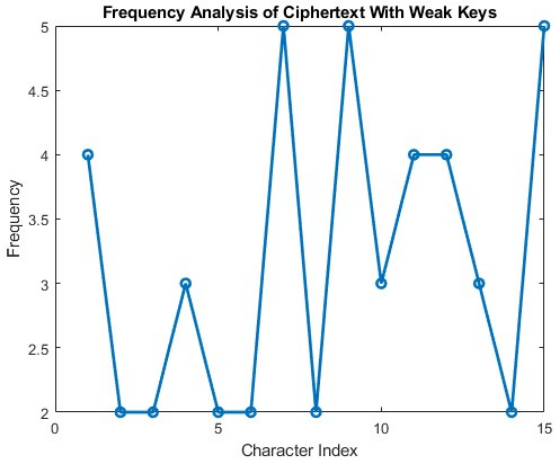

Figure 5: Frequency Distribution of Ciphertext With Weak Keys Combined