TIME MEMORY TRADEOFF ATTACK ON DATA **ENCRYPTION STANDARD (DES)**

Project report submitted in fulfilment of the requirement for the degree of

Bachelor of Technology In **Computer Science and Engineering** By

Anmol Mahajan (151222)

Under the supervision of

Dr. Suman Saha



Department of Computer Science & Engineering and Information Technology Jaypee University of Information Technology Waknaghat, Solan-173234,

Himachal Pradesh

Certificate

Candidate's Declaration

I hereby declare that the work presented in this report entitled TIME MEMORY TRADEOFF ATTACK ON DATA ENCRYPTION STANDARD(DES) in fulfilment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering/Information Technology submitted in the department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat is an authentic record of my own work carried out over a period from August 2018 to December 2018 under the supervision of Dr. Suman Saha, Assistant Professor (Senior Grade), Computer Science and Engineering/Information Technology.

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Anmol Mahajan, 151222

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Dr. Suman Saha Assistant Professor (Senior Grade)

Computer Science and Engineering / Information Technology

Dated: 1/12/2018

ACKNOWLEDGEMENT

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend our sincere thanks to all of them.

I am highly indebted to Dr. Suman Saha for their guidance and constant supervision as well as for providing necessary information regarding the project and also for their support in completing the project.

I would like to express our gratitude towards our parents and Jaypee University of Information Technology for their kind cooperation and encouragement which helped us in completion of this project.

Mine thanks and appreciations also go to our colleague in developing the project and people who have willingly helped us out with their abilities.

TABLE OF CONTENT

1.	INTRODUCTION	Page No.
	1.1 Introduction	1-4
	1.2 Problem Statement	5
	1.3 Objectives	5
	1.4 Methodology	6
	1.5 Organization	6
2.	LITERATURE SURVEY	6
	2.1 A Cryptographic Time Memory Trade Off 2.2 Rigorous Time/Space Trade Off for Inverting Eulerion	6-7
	2.3 Cryptanalytic Time/Memory/Data Trade Offs for Stream Ciphers	8
	2.4 A Time-Memory Trade Off using Distinguished Points	8-9
	2.5 Breaking Ciphers with COPACOBANA-A Cost-Optimized Parallel	9
	Code Breaker	

2.6 Making a Faster Cryptanalytic Time-Memory Trade-Off9-102.7 TMTO Implementation on COPACOBANA10-11

3 SYSTEM DEVELOPMENT

3.1 Analytical	11-14
3.2 Computational	15-17
3.3 Experimental	17-19
3.4 Mathematical	19-20
3.5 Statistical	21-23

4 ALGORITHMS

4.1 Feistel Function	24-26
4.2 Key Scheduling Algorithm	26-27
4.3 Data Encryption Standard	27
4.4 Trudy's World VS Real World	28-30
4.5 Hellman Analysis	30-34
4.6 Working	35-38

5. TEST PLAN

2.1 Metrics if Hellman Analysis is used	38-39
2.2 Metrics if DP Analysis is used	41-42
2.3 Metrics if Rainbow Table Analysis is used	42-43

6. CONCLUSION AND FUTURE SCOPE

44

LIST OF FIGURES

Fig No.	Title
3.1	Computational Model for project functionality
3.2.1	Hellman Analysis
3.2.1(a)	Binary Search
3.2.1(b)	$\log(n)VS(n)$
3.2.2	Analysis Of Distinguished Points under different parameters
3.2.5	Success probability for distinguished points
3.2.6	Classical Analysis
3.2.7	Rainbow table
3.2.8	Comparison between success rates of classical and rainbow tables
4.1	A Classical Feistel Network
4.2	Construction Of P-box
4.3	The Key-Mixing Step
4.4	Sample Substitution Box
4.5	Sample D-Box
4.6	DES
4.6.1	Trudy's Perfect World
4.6.2	Merges in chains
4.6.3	Intersection of chains
4.7	Hellman Table
4.8	Starting and End points of chains
4.8 (a)	Calculation of Hellman chains
4.8(b)	Storing starting and end points of chains
4.8(c)	Finding index of encrypted text
4.8 (d)	Output of key for encrypted text
4.8 (e)	Rainbow table implementation

LIST OF TABLES

Table No.	Title
3.2.3	The influence when dp-property changes in distinguished points
3.2.4	The influence when chain length changes in distinguished points
3.2.9	Predicted Coverage
3.2.10	Comparison between the three techniques
4.9	Table with different success probabilities

ABSTRACT

Cryptography plays a very important role in a wide range of areas from preserving our privacy on the online world to providing military a secure methodology to undertake their operations. In the vast history of this field the art of encoding or encrypting the messages has held the utmost importance. We have come so far. Ahead of using the secret hieroglyphics to the digitally advanced automated computational phase.

Today digital security is taken for granted. But there are also some more aspects that need to be considered. These algorithms always tend to have some inherent sense of imperfection in them which can be exploited in one way or other to break the system. This could risk the whole system that we have built and can have some serious consequences.

The art and science of finding such faults is called crytanalysis.DES is also one such encryption algorithm that had its share of inbuilt faults and many cryptanalytic methods have been built to break it. Some of these include Time Memory Data Trade-Off Attack, Differential Analysis, Linear Analysis and Differential-Linear Attack. Today, breaking DES is a task of only about a few hours but the cost of the required hardware is very high. In this project we are going to cryptanalyse the DES algorithm using the famous Time Memory Trade-off Attack.

1. Chapter 1

1. Introduction

Humans the most civilized and advanced species on the planet has progressed due to his knowledge and the ability to successfully pass on this knowledge to his future generations. Communication was the primary means of achieving the same. Initially humans exchanged information using gestures. Slowly, he started carving information into objects in the form of images, sculpures etc. to remember it for a longer term. Speech was a golden gift endowed to humans.

But with time the need to protect the information, from leaking to the unwanted sources also emerged. The science corresponding to adding the confusion and diffusion to the message has also developed a lot and is now called cryptography. Cryptography is actually an old science which has seen about 4000 years of development. The credits to the origin of the word "Cryptography" go to the Greeks. The word means "secret writing". In the earliest times the use of this science was mainly limited to the military purposes. The earliest example or proof that exists today is from 2000 B.C. found in Egypt. This is the ancient hieroglyphic which was considered as the sacred writing since people were unable to decode the images and symbols embedded on it. Later a different tablet was found on which the meaning of these symbols was mentioned against their native language. Turned out that this was a simple substitution cipher. The military has used cryptography since a long time. It has been proposed that the brave army of Spartans also used this art to communicate messages to their generals. They shaved their slave's head, wrote the message on the bald head and waited for his hair to regrow. Then they sent the slave to the required place, where receiver shaved his head and recovered the message. The Romans are also posited to know something about the art of cryptography. King Caesar himself is credited with the invention of the famous Caesar cipher. This is a simple substitution cipher which involves the shifting to the plaintext by the number mentioned in the key. Hence, it can be said that the substitution ciphers ruled the classical age of cryptography.

As military used this science to exchange important messages the opponents wanted to break the cryptographic systems and gain the hidden information. This lead to the development of something called cryptanalysis. Cryptanalysis can be defined as the analysis of the existing cryptographic systems and using the knowledge gained to break the system with the gain of access to the secret information. The Arabs are credited with the invention of cryptanalysis. The most prominent method being the one developed by the famous mathematician Al-

Kindi as earlier as in 820 AD. He was the first to analyse the monoalphabetic substitution ciphers and invented the frequency based analysis. This advancement turned out to be most important till Second World War. As most of the existing systems were just substitution ciphers most of them were susceptible to the above attack. So need to develop more advanced cryptosystems emerged. Such system did not release until mid 1400's.A polyalpahbetic cipher was created by Leon Arberti. He is often referred as "The father of western cryptography". His method involved the use of a cipher disk which consisted of a movable inner disk. With movement this disk could map the inner text to any required plaintext. Until now the use of a secret key was obscure and unknown. In the mid 1500's Blaise De Vigenere came up with an idea that would revolutionize the way cryptography was done. His scheme involved repeating the key to adjust to the length of the plaintext and performing simple addition modulo 26. The simplicity of the method along with the confusion and diffusion it offered was remarkable. Vigenere Cipher was secure late until 1800's.In 1865 Friedrich Kasiski developed a method called KASISKI TEST through which the Vigenere cipher was broken .The cryptography remained approximately the same till the twentieth century.

But by the advent of the industrial revolution and the technological era, the amount of information exchanged experienced an exponential growth. Hence, an urgency in raising the level of the technology involved in the exchange of messages. Moreover the World Wars played an important role in the advancements in this field. In 1917 the British Army intercepted a message intercepted a message sent by the German Army to the Mexican Government. The encrypted message was soon decrypted by the British intelligence. In the message the German Foreign Minister Arthur Zimmerman addressed the Mexican ministers pleading them to join the war against the United States Of America. This instigated the American leaders and lead them to participate in the World War. Now the Americans were aware of the importance of maintaining secrecy and developed a better cryptosystem, One Time Pad. One Time Pad consisted of generation of a random key with length equal to the length of the plaintext and XORing the two to get the required ciphertext. This scheme was proposed by Gilbert Sandford Vernam in 1917 only. Similar things also happened in the second world war. The Japanese had invented a machine called PURPLE and encrypted messages using this machine only.USA had developed the decryption algorithm. their Unaware of this the Japanese messaged their fleet posted near USA about the arrival of their

military chief. The American army seized this opportunity and killed a very important ,passionate leader. But the enigma machine saga is the most popular and heroic.In 1932 German engineer Arthur Scherbius developed a machine called Enigma which was capable of performing various encryption operations. Machine turned out to be quite handy in the world war for communicating messages, secretly. The British were able to intercept many messages but were unable to break these messages. So, they employed a team of great cryptographers and mathematicians consisting of greats like Alan Turing to break the codes. Eventually, made many observations and figured out many weak points. They found out that a letter in plaintext will never map to the same alphabet in the ciphertext. Using this and many similar facts the British were able to break the enigma machine and gathered massive amount of information about the actions of the German army, which they used to eventually defeat the German propaganda.

Since the world war the science of cryptography has improved exponentially. Computers have become part and parcels of our life. We all have a life online and it has become very important to secure it against any potential threats. Hence, many popular cipher schemes have been developed. This includes many cryptosystems. The old block ciphers, stream ciphers to the more advanced elliptical curve cryptography, lattice based cryptography and even quantum cryptography. All of these provide us with a secure online life. The algorithms like DES (Data Encryption Standard) were used initially to encrypt the data online. But the processing power and the limitless resources are the challenges it was unable to keep up to. Having a key size of only 56 bits until 1990's the scheme was broken using brute force and differential analysis. Due to the importance that these algorithms hold, we require them to be impervious against any kind of attack. Hence, we analyse these algorithms against all possible threats.

One such common threat is the brute force attack. This involves plugging in every possible key and looking for the right key. But due the large size of the key involved our processing power turns out to be feeble. So, cryptanalysts always wanted to develop a better method to brute force. A method more efficient and something that could be used universally to crack any kind of scheme. In 1980 Martin E. Hellman proposed something that seemed like the answer. A simple proposal claiming to establishing a middle way between the memory used and the processing power. A trade of between the time and memory. His solution claimed to reduce the time of brute force from N to N^{1/3}. The proposal was eventually accepted and is a famous method to cryptanalyze any algorithm or cryptographic scheme. Cryptanalysis is an

important part of cryptology. We need our solutions to be perfect and want to be the first to discover any fault in them, if it does. What will happen if a person having evil intents finds out some faults or loopholes in our cryptographic scheme? Financial crisis, inconsistent information, leakage of information related to national security.DES has a similar history. DES was implemented as the security algorithm in every system from 1970's.The Hellman proposal though practical took a long period to successfully challenge DES and force it to be replaced. The DES served as the encryption standard late until 1990's when NIST finally called for replacing it.

Types of Attacks on DES

1. Brute Force Attack

This is the most basic type of attack on any kind of cipher in which we try every possible key in turn. Number of possible keys are determined by the length of the key and this also determines how feasible this approach will be. Even before DES was adopted as a standard, a lot of questions were raised regarding adequacy of its key size because its key size was too small. This showed the need for a replacement algorithm.

2. Differential Cryptanalysis

Earlier known to IBM and NSA and kept secret, Differential Cryptanalysis was rediscovered in 20th century, 1980s by Eli Biham and Adi Shamir. Chosen plaintexts are needed to exploit all 16 different rounds by Differential Cryptanalysis. Data Encryption Standard wa made resistant towards Differential Cryptanalysis.

3. Linear Cryptanalysis

This attack was discovered by Mitsuru Matsui and required 2⁴³ known plaintexts. This was the first ever Linear Cryptanalysis on DES. In 1994, Multiple Linear Cryptanalysis was proposed and was further refined by Biryukov. In 2000, a chosen plaintext variant was also introduced of linear cryptanalysis with reduced data complexity.

4. Improved Davies' Attack

It was a special technique suggested by Donald Davies which was further improves by Biham and Biryukov. The most overwhelming kind of DA needs 2^50 plaintexts and has a success probability of 0.51.

1.2 Problem Statement

Data Encryption Standard served as the encryption standard for nearly 30 years. Explain how this algorithm was finally broken and demonstrate a Time Memory Data Trade Off Attack on it.

1.3 Objectives

GOAL: To perform Time Memory Tradeoff Attack on DES with high success probability.

METRICS:

Time taken by the system to perform the attack.

The success probability involved.

The amount of memory required.

Objective 1:

Implement and optimize the implementation of Data Encryption Standard.

Deadline: October 2018

Expected Speed: 2²⁰ encryptions per second on normal machine (Intel i5 2.7 Ghz clock speed).

Objective 2:

Implementation of the offline phase of Time Memory Trade Off on Data Encryption Standard.

Deadline: December 2018

Memory Required: 34357938369*4bytes [140 GB for a success probability of about .86]

Objective 3:

Extend the implementation using rainbow and perfect tables. Deadline: February 2019

Objective 4:

Implementation of the online phase of Time Memory Trade Off on Data Encryption Standard.

Deadline: April 2018

Expected Time: O(log(m)+t)

1.4 Methodology

In the final step of our project we will create a Graphical application and merge it with the attack's implementation. The visuals will contain a button for preparing a table for a new plaintext or for performing the online phase. After performing the attack the key will be displayed. High processing power is required for our system. Hence, we will be needing GPU's which will be from some cloud online services.

2. Chapter-2 LITERATURE SURVEY

2.1 A Cryptographic Time Memory Trade Off

2.1.1 Author: Martin E. Hellman

2.1.2 Year: 1980

2.1.3 Summary:

This was the first paper to introduce the idea of finding a middle way between the amount of memory used and the time required to brute force the entire scheme. This paper proposed to break down the task of brute forcing into two parts that is firstly the pre computation step to make a table offline and then the online phase to search for the key actually used.

2.1.4 Advantages:

2.1.4.1 First paper to introduce the concept of using Time Memory Trade Off.

2.1.4.2 Reduced the brute force time by a significant amount.

2.1.5 Disadvantages:

2.1.5.1 False Alarms in the online phase.

2.1.5.2 Merged chains and loops created problems and increased the amount of total memory required.

2.2 Rigorous time / space tradeoffs for inverting functions

2.2.1 Author: Amos Fiat, MoniNaor

2.2.2 Year: 1991

2.2.3 Summary: The initial proposal of Hellman claimed that TMTO could be extended and used for inverting one way functions also. But TMTO remained limited upto block ciphers only and its use for inversion was realised in this paper, which proposed an effective methodology for achieving this.

2.2.4 Advantages:

2.2.4.1 First paper to extend the concept of using Time Memory Trade Off upto inversion of the one way functions.

2.24.2 Reduced the brute force time by a significant amount.

2.2.5 Disadvantages:

2.2.5.1 The proposal was just a conceptual one and they were unable to prove it with practical results.

2.3Cryptanalytic Time / Memory / Data Tradeoffs for Stream Ciphers

2.3.1 Author: Alex Biryukov and Adi Shamir

2.3.2 Year: 2000

2.3.3 Summary:

This was the first paper to propose a method for implementing TMTO successfully on stream ciphers. The idea was to use low sampling resistance for tradeoff attacks on stream ciphers.

2.3.4 Advantages:

2.3.4.1 First paper to introduce the concept of using Time Memory Trade Off on the stream ciphers.

2.3.4.2 The time for the pre computation phase decreased. Moreover, the total need for the data decreased.

2.4 A Time-Memory Trade off by use of Distinguished Points

2.4.1 Authors: Francois-Xavier Standaert, GaelRouvroy, Jean-Jaques Quisquater

2.4.2 Year: 2002

2.4.3 Summary:

The first paper to implement the proposal of Rivest of using distinguished points. In this method rather than generating the chains of same length, we fix the end points of chain and keep on generating the chain until any of these endpoint is achieved.

2.4.4 Advantages:

2.4.4.1 First paper to extend the concept of using distinguished points for performing Time Memory Data Trade Off.

2.4.4.2 The algorithm gave us better parameters and reduced the total memory references and made the implementation easier.

2.5 Breaking Ciphers with COPACOBANA-A Cost-Optimized Parallel Code Breaker

2.5.1 Authors: SandeepKumar, ChristofPaar, JanPelzl, Gerd Pfeiffer

2.5.3 Summary: This paper was the first to propose the design of a specialised machine called COPACOBANA to break DES. The machine was capable of doing so in about 2 days. The estimated cost of the machine was about \$10,000 which was considered more than affordable.

2.5.4 Advantages:

2.5.4.1 First paper to propose an adequate design for a specialised machine.

2.5.4.2 The designed turned out to be very efficient and cost effective.

2.6 Preparing a Faster Cryptanalytic Time-Memory Trade-Off

2.6.1 Author: Philippe Oechslin

2.6.2 Year: 2003

2.6.3 Summary: Introduced the concept of rainbow table. The rainbow table is a variant of the time memory trade off method which deals with the various problems its ancestor had by using different reduction function at each step in the chain. Philippe Oechslin showed that performance of the TMTO can be increased drastically using different reduction function.

The probability of collision between two chains decreased to 1/t. Philippe Oechslin further extended his work and worked to find the optimum values of the parameters m, t and l and worked to create something called perfect tables. Perfect tables have 0 merges and no memory is wasted.

2.6.4 Advantages:

2.6.4.1 First paper to propose the idea of rainbow tables. Perfect table was also introduced in this paper only.

2.6.4.2 Gave proper procedure to choose the various parameters involved in performing Time Memory Data Trade off.

2.6.4.3 Tackled the problems of false alarms, merges and loops originally prominent in the Hellman paper.

2.7 Time Memory Tradeoff Implementation on Copacobana

2.7.1 Author: Stefan Spitz

2.7.2 Year: 2007

2.7.3 Summary: Major objective of this research work was to provide a systematic method which can lower the time required to break the cipher and extract the original key from the block cipher. Its implementation was on the popular Data Encyption Standard (DES). Stefan used specifically designed hardware Copacobana to attack on the DES in the least time possible.

2.7.4 Advantages:

2.7.4.1 This particular utilization of only one task gives the odds to make this progressively difficult procedure an easier one and expends less time of breaking the DES.

2.7.4.2 Provided the predictability to improvise the structure of some specific tasks which can be included using FPGA's.

3. Chapter-3 SYSTEM DEVELOPMENT

3.1. Computational

The computational model will consist of the use of cloud computing services. The algorithm will be uploaded to the cloud where high performance hardware will run our algorithm till the creation of the tables.



3.1.1

3.2. Mathematical

Hellman Analysis

The equation for each step :

 $C=S_k(P)$

The output generated in this step will be of 64 bits. So we reduce it to the key size i.e 56 bits.

 $f(K)=R[S_k(P)]$

The probability of success is given by the formula:

 $P(S) \!\!> \!\!=\!\! (1/N) \!\!\sum_{i=1}^{m} \!\!\sum_{j=0}^{t-1} [(N \! - \! it)/N]^{j+1}$

The approximate count of false alarms per table is given as:

 $E(F) \le mt(t+1) / 2*N$

Hellman suggested to follow the relation:

M=m*t

mt²=N

 $m=t=N^{(1/3)}$

Breaking the DES via this method was reduced to 2³⁸.

m - number of starting points

t - length of chain



Fig. 1. Construction of the function f.



Fig. 2. Matrix of images under f.

3.2.1

Alterations in DES:

DES (Data Encryption Standard) is one of the initial encryption algorithms that involved 64 bits of plain text being encrypted using key of 56 bits. DES involves total of 16 rounds with partition of plaintext into two equal halves named left and right part respectively. Similarly, key is also divided, permutation and combination boxes are used and finally halves are XORed and stored respectively for the next round.

I have implemented DES in C++14 and to improve its speed, I have stored S- boxes, Permutation boxes and Combination boxes, instead of calculating them every time. Such an implementation has improved the speed of our DES algorithm to 2^{20} encryptions.

Sorting Function:

In my approach of implementation of Time Memory Trade Off Attack, I used the already present inbuilt sorting function of C++14. The inbuilt C++14 sorting function is comprised of an introspective sorting which is the overall mix of three types of sorting namely quick, heap and insertion sort. Such sorting is formulated in such a way that it overall tries to perform in a better way, covering all kinds of cases, that is ranging from best case to the worst possible case.

Insertion sort If elements in array or list are less than or equal to 16, Insertion Sorting is used

Heap sort performs its basic functions such as heapify in the algorithm

Quick sort is mainly used to find the pivot element, that is the point from where partition will happen

Therefore, Introsort is an optimized combination of all the above mentioned sorting algorithms for dealing with average complexities provided as solution in best and worst possible cases.

Algorithm:

Sort(A, A+size) depLimit = 2*floor(log(len(A))) insort(A, depLimit)

insort(A, depLimit): b = len(A) if b<=16: insertionSort(A) if (depLimit == 0): heapsort(A) else:

// using quick sort, the
// partition point is found
c = partition(A)
insort(A[0:c-1], depLimit - 1)
insort(A[c+1:b], depLimit - 1)

Why binary searching algorithm is preferred over other searching algorithms?

Binary search:

Let us assume that we are provided with a array which is already in a sorted fashion comprising of n values in total and we are looking for a particular value in this array. Therefore, in binary searching, we divide the provided list of elements in two partitions, and look whether the middle values is bigger or smaller than the value we are looking for. If target value is large then look in the right portion of the list and if target value is lesser then look in the left portion of the list. Repeat the above procedure until we find the required target value in the list.



Figure 3.2.1 (a)

Why we preferred binary searching algorithm?

The main reason behind using binary search algorithm is its overall lower complexity in comparison with linear searching algorithms. The overall time complexity of binary search is $O(\log(n))$ and for linear searching it is O(n*n). In best case situations, binary search

provides the results in O(1). Moreover, binary search uses no extra space for finding the target value in the list.

Figure 3.2.1 (b)

Why Hashing Algorithms were avoided for this purpose?

We know that hashing algorithms are fast as there have overall linear time complexity. But there are two parameters due to which we cannot use Hashing in Time Memory Trade Off Attack, that is large number of key value pairs stored and the manner in which these chains are stored for dealing with memory and time constraints. During the execution of Time Memory Trade Off Attack, we keep track of millions or billions of encrypted texts in precalculation part which overall leads to a lot of values to be stored and kept track of. If we include hashing in TMTO then we again need to keep track of hashvalues of billions of calculations which is not at all feasible. Such a procedure will surely hike up the space complexity to a whole new stage and keeping track of such a large number of values only for looking for one encrypted key is not at all a good practical approach.

Why we preferred iteration over recursion in our binary searching approach?

Although iterative as well as recursive binary search have similar time complexity $(O(\log(n)))$ but there is major difference in other parameters such a overall length of written code, overall complexity of the written code and of course, space complexity. The major difference between space complexities of iterative and recursive methods is that the space complexity of iterative solution is O(1) but in case of recursive implementation it can go up to O(n).

In Time Memory Trade Off Attack, we need to evenly go for both time as well as space complexities for our calculations. Therefore, it is advantageous to implement iterative binary searching algorithm and prefer it over recursive binary searching algorithm.

Expansion and reduction function in implementation of rainbow tables:

Now, there was a major problem with the solution proposed by Martin Hellman on TMTO that there was always same reduction function being used for at each point for storing values of encrypted texts in chains. In Implementation of TMTO over DES, we were using the reduction function as ignoring last 8 bits of 64 bits output of DES and storing only 56 bits. But, using same reduction function over and over again at each point in all chains can lead to different problems in real world scenario such as loops, circles ad merging of chains.

In rainbow tables, we use reduction function as an iterative function on the outcome of DES to reduce the number of sample output bits from 64 bits to 56 bits and passing these 56 bits as the key for net DES function in chain, keeping the plain text same all the time. For implementation of TMTO on other encryption algorithms such as triple DES, we can use an expansion function in same manner.

In expansion function, we use an iterative function on the outcome of 3DES for the expansion of the size of bits from 64 bits to 112 bits and passing these 112 bits as input for next 3DES function in the chain, keeping the plaintext same all the time.

Why different reduction / expansion functions on each point in rainbow tables?

If different reduction or expansion functions in rainbow tables are not used, then the properties of rainbow tables are compromised, that is rainbow tables will be same as the tabular forms of hash chains that are overall not so much good in efficiency for tabular forms of larger sizes.

Distinguished Points Analysis

The original algorithm involves fixing the length of the chain and also the number of the starting points for the table. We pre compute r tables by picking r distinctive cover capacities. For each veil work m distinctive begin focuses (recognized) will be chosen

randomly. For every begin point a particular chain will be registered till the point that a Distinguish Point is experienced or the point that the chain length is t + 1. Just begin focuses emphasizing to a Distinguished Point in under t cycles will be given away with the relating chain length, the others will be disposed of. In addition, if a similar DP is a last point for a lot of chains, at that point just the chain of maximal length will be given away.

This includes less amount memory unpredictability in comparison with Hellman's trade off. Precomputation algo: Create (r) tables with (SP, EP, l)-triples, sorted on the basis of endpoints.

1. First a distinguish point property is to be chosen .The property will have d as its order.

2. Choose r number of different mask functions. Each of these mask function will further generate a different function which can be used for reduction.

3. As the probability of achieving the specific endpoint or distinguished point is quite less the length of the chain can be very large. Hence, we choose a max length variable t.

- 4. Start a continuous loop from 1 to r
- (a) Fix, or say, still any m initial starting point
- (b) For j = 1 to m, l = 1 to t
 - i. Calculate f (Starting Point j).
 - ii. If f (Starting Point j) is a distinguish point then keep track of triple (Starting Point j , End Point j = f (Starting Point j), l)

and iteratively consider next j.

iii. If l > t "let go" Starting Point j and consider following j.

(c) Now arrange the given triplets in sorted fashion on the basis of endpoints. More than one such triplets can be achieved. So, choose the one which has the maximum length.

(d) for every table store the value of the maximum l: lmax

Searching algo: Given Cipher = Encryption K (Plain text) find K.

1. Start a continuous loop from 1 to r

(a) Find lenmax

- (b) Y = gi(C).
- (c) For I from 1 to lenmax

- i. If the given Y is a Distinguised Point
- A. If Y is present in table number i, then do
- Consider the givenSP (i) and len l in that particular table.
- But if j is less than 1
 - Calculate parent $\tilde{K} = f-1-j$ (Starting Point 1).
- If $C = Encryption \tilde{K}$ (Plain text) then of course $K = \tilde{K}$: STOP.
- If C 6= Encryption \tilde{K} ey (Plain text), consider the upcoming value of i.
- B. Otherwise consider upcoming value of i.
- ii. Declare Y = f(Y).

The probability of reaching DP is given by the formula: $P_2(l) = \prod^{l-1}_{i=0} (1 - 2^{k-d}/2^k - i)$

Choosing the average chain length β

$$P_2(l) \simeq (1 - \frac{2^{k-d}}{2^k - \frac{l-1}{2}})^l$$

$$P_1(l) = 1 - \prod_{i=0}^{l-1} \left(1 - \frac{2^{k-d}}{2^k - i}\right)$$
$$P_1(l) \simeq 1 - \left(1 - \frac{2^{k-d}}{2^k - \frac{l-1}{2}}\right)^l$$

Figure 3.2.2

Average length of chain can be computed as :

$$\beta = \frac{\sum_{l=t_{min}}^{t_{max}} l.P(DP.in.exactly.l.iterations)}{\sum_{l=t_{min}}^{t_{max}} P(DP.in.exactly.l.iterations)}$$

$$\gamma = \sum_{l=t_{min}}^{t_{max}} P(DP.in.exactly.l.iterations) = P_1(t_{max}) - P_1(t_{min} - 1)$$

Numerator can be estimated as follows:

$$\begin{split} &\sum_{l=t_{min}}^{t_{max}} l.P(DP.in.exactly.l.iterations) \\ &= \sum_{l=t_{min}}^{t_{max}} l.(\prod_{i=0}^{l-2}(1-\frac{2^{k-d}}{2^k-i}) - (\prod_{i=0}^{l-1}(1-\frac{2^{k-d}}{2^k-i})) \\ &\simeq \sum_{l=t_{min}}^{t_{max}} l.((1-\frac{2^{k-d}}{2^k-\frac{t}{2}})^{l-2} - (1-\frac{2^{k-d}}{2^k-\frac{t}{2}})^{l-1}) \end{split}$$

Where t is the average of maximum t and minimum t Equation can be rewritten in simpler terms as follows:

$$\sum_{l=t_{min}}^{t_{max}} l.((1-x)^{l-2} - (1-x)^{l-1})$$

 $x = \frac{2^{k-d}}{2^k - \frac{t}{2}}.$ Here,

$$\sum_{l=t_{min}}^{t_{max}} l.((1-x)^{l-2} - (1-x)^{l-1})$$

= $t_{min}.(1-x)^{t_{min}-2} - t_{max}.(1-x)^{t_{max}-1} + \sum_{l=t_{min}-1}^{t_{max}-2} (1-x)^{l}$
= $(1-x)^{t_{min}-2}.(t_{min} + \frac{1-x}{x}) - (1-x)^{t_{max}-1}.(t_{max} + \frac{1}{x})$

At last, the average length of chain can be given as follows :

$$\beta \simeq \frac{(1-x)^{t_{min}-2} \cdot (t_{min} + \frac{1-x}{x}) - (1-x)^{t_{max}-1} \cdot (t_{max} + \frac{1}{x})}{\gamma}$$

Firstly the property of distinguished points can be evaluated as follows,

DP-property	Length region (log_2)	Experimental β (log ₂)	Theoretical β (log ₂)
DP-11	9-13	11.2140	11.2140
DP-12	10-14	12.2137	12.2139
DP-13	12-14	13.0965	13.0966
DP-14	13-15	14.0967	14.0966
DP-15	11-18	15.0771	15.0836

3.2.3

Similarly, influence of chains can be observes as

DP-property	Length region (log_2)	Experimental β (log ₂)	Theoretical β (log ₂)
DP-13	10-13	11.9790	11.9987
DP-13	12-14	13.0965	13.0966
DP-13	13-16	14.0107	13.9955

3.2.4

Probability of finding key using table t different keys with m different rows is

$$P_{table} \ge \frac{1}{N} \sum_{i=1}^{m} \sum_{j=0}^{t-1} \left(1 - \frac{it}{N}\right)^{j+1}$$

The overall probability of success with use of 1table only is provided as

$$P_{success} \ge 1 - \left(1 - \frac{1}{N} \sum_{i=1}^{m} \sum_{j=0}^{t-1} \left(1 - \frac{it}{N}\right)^{j+1}\right)^{\ell}$$

	classic with DP	rainbow
t,m,ℓ	4666, 8192, 4666	4666, 38'223'872, 1
predicted coverage	75.5%	77.5%
measured coverage	75.8%	78.8%

3.2.5

Solution space for probability of success with 99.9%, maximum size of 220 seconds and memory size of 1.4GB

Table os size m x t is provided as

$$P_{table} = 1 - \prod_{i=1}^{t} \left(1 - \frac{m_i}{N}\right)$$

where $m_1 = m$ and $m_{n+1} = N\left(1 - e^{-\frac{m_n}{N}}\right)$

$$\begin{array}{c} t \\ m \downarrow \begin{bmatrix} k_{1,1}^{1} & f_{1} & f_{1} & \dots & f_{1} & k_{1,t}^{1} \\ k_{m,1}^{1} & f_{1} & f_{1} & \dots & f_{1} & k_{n,t}^{1} \end{bmatrix} \\ m \downarrow \begin{bmatrix} k_{1,1}^{2} & f_{2} & f_{2} & \dots & f_{2} & k_{1,t}^{2} \\ k_{m,1}^{2} & f_{2} & f_{2} & \dots & f_{2} & k_{m,t}^{2} \end{bmatrix} \\ \vdots & \vdots & \vdots \\ m \downarrow \begin{bmatrix} k_{1,1}^{t-1} & f_{t-1} & f_{t-1} & \dots & f_{t-1} & k_{1,t}^{t-1} \\ k_{n,1}^{t-1} & f_{t-1} & f_{t-1} & \dots & f_{t-1} & k_{m,t}^{t-1} \end{bmatrix} \\ m \downarrow \begin{bmatrix} k_{1,1}^{t} & f_{t} & f_{t} & \dots & f_{t} & k_{m,t}^{t} \end{bmatrix} \\ k_{m,1}^{t} & f_{t} & f_{t} & \dots & f_{t} & k_{m,t}^{t} \end{bmatrix}$$

3.2.6

And rainbow table of overall size mt x t can be constructed as

3.2.7

3.2.8

Comparison between success rates of classical and rainbow tables

Following are the estimations for classic tables with different end points and for rainbow tables after calculating keys for 500 different password hashes

	classic with DP	$\operatorname{rainbow}$	
t,m,ℓ	4666, 8192, 4666	4666, 38'223'872, 1	
predicted coverage	75.5%	77.5%	
measured coverage	75.8%	78.8%	

Cryptanalysis statistics with tables yielding 99.9% success rate. We can observe from the middle column easily that rainbow table notes a reduction of 12 times in terms of calculations.

	classic with DP	rainbow	ratio	rainbow sequential	ratio
t, m, ℓ	4666, 7501, 23330	4666, 35M, 5	1	4666, 35M, 5	1
cryptanalysis time	101.4s	66.3	1.5	13.6s	7.5
hash calculations	90.3M	$7.4\mathrm{M}$	12	11.8M	7.6
false alarms (fa)	7598	1311	5.8	2773	2.7
hashes per fa	9568	4321	2.2	3080	3.1
effort spent on fa	80%	76%	1.1	72%	1.1
success rate	100%	100%	1	100%	1

3.2.9

To know the number of distinguishable chains, we just need to observe overall count of distinct particular points in the ending segment

$$\hat{P}_{table} = 1 - e^{-t\frac{m_t}{N}}$$
 where $m_1 = N$ and $m_{n+1} = N \left(1 - e^{-\frac{m_n}{N}}\right)$

Algorithms:

In this project we are working to break a classical encryption technique called DES. DES stands Data Encrypt Standard. It is a symmetric key algorithm which is partially software and hardware implementation. The algorithm was proposed by Horst Fiestel who at the time worked at IBM in 1970's. The algorithm incorporates Fiestel networks and is a modification of Lucifer cipher scheme. The algorithm served as the standard encryption block cipher algorithm from 1977 to 1998 when it was replaced by AES. The major problem faced was the small size of the key and the possible differential cryptanalysis.

DES is the original square figure—a calculation that takes a settled length string of plaintext bits and changes it through a progression of convoluted activities into another ciphertext bitstring of a similar length. On account of DES, the square size is 64 bits. DES likewise utilizes a key to redo the change, with the goal that decoding can apparently just be performed by the individuals who realize the specific key used to scramble. The key apparently comprises of 64 bits; in any case, just 56 of these are really utilized by the calculation. Eight bits are utilized exclusively to check equality, and are from that point disposed of. Thus the viable key length is 56 bits. DES is never used natively. Hence, some mode of operation like Cipher Block Chaining, output feedback etc. are always used. The process of decryption is similar with the only difference being the opposite order of the used keys.

DES is based on something called the Feistal network. The network consists of smaller blocks which individually instil some confusion and diffusion. The total input bits are divided into two equal parts (here 32 bits each). They are both treated differently with the left bits going undisturbed to the right portion of the next Feistel block. The right bits are first fed into a function called Feistel function and are then XORed with the left 32 bits. This output is then take to the left block of the next round.

A Classical Feistel Network

4.1

Feistel function:

The feistel function works on half of the bits at a time. Each undergoes through four main steps.

1. In this step the input's 32 bits are expanded into 48 bits. This is done by the use of an permutation expansion box. The input is first divided into eight four bits blocks. Now, each of these four bits are expanded into 6 bits.

4.2

2. The above 48 bits are XORed with the 48 bits of the key. It is to be noted that the input key size was only 56 bits. So, here we use a pseudo random generator to actually produce the required key. The step is known as the key-mixing step.

In this step a substitution box is used. This is used to convert the 48 bits output to 32 bits. The input size is divided into 6 blocks, each of 8 bits. Now we have a two dimensional matrix containing 4rows and 16 columns. The first and the last bit are used to find the row and the middle four bits are used to locate the column. Each cell has a 4 bit value and we substitute the given 6 bits with these 4 bits. After doing this for all 8 blocks we will have a resulting size of 8*4 i.e. 32 bits. It is to be kept in mind that we need to prevent any kind of linearity in the construction of these s-boxes. Moreover, for all 8 blocks of data different s-box is used.

4. Now the output 32 bits are spread out using a permutation block. The basic functionality of a p-box is to scramble the output and to distribute the effect of the s-boxes to a wider range. This is called the permutation step.

R1=L0

 $L1=f(r)^L0$

Here, R1 that is the right portion stores the value of previous left part of plaintext and L1 stores the output of previous left part XORed with overall computation of function f(r).

The Key-Mixing step

_		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
	1	00	15	07	04	14	02	13	10	03	06	12	11	09	05	03	08
	2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
	3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

Sample Substitution Box

4.4

Key Scheduling Algorithm

For each function in the feistel round we need a key of 48 bits. As a lot of such steps involving different keys can be there we cannot use the input key directly. Hence, we need a key scheduling algorithm.

Steps involved in key scheduling:

1. The 56 bits input key bits are separated into two parts. These keys have a size of 28 bits.

The bits are shifted by some positions. The first, second, ninth and sixteenth positions are shifted by 1 bit to the right. All the remaining bits are shifted by two bit positions to right.

3. Now a D-box is used. The bits convert 28 bits of the key into 24 bits. The left and the right part are now combined to give us the 48 bits which are used for the encryption process.

14	17	11	24	01	05	03	28
15	06	21	10	23	19	12	04
26	08	16	07	27	20	13	02
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

Figure 4.5

The first three rows of the table are used for the compression of the left part of the key while the lower three keys are used to reduce the size of the right part.

Data Encryption Standard

This algorithm is built by the use of Feistel network. Sixteen Feistel blocks are used in DES. The block size for this algorithm is 64 bits. Hence, DES is nothing but Feistel rounds applied 16 times.

Trudy's Perfect World

Initially, when Time Memory Trade off Attack was just conceptualised, nobody thought about the problems that can occur within the chains, such as merging of chains, detection of loops and circles in chains.

Trudy, a researcher provided his thoughts in this proposed solution, which are now considered as Trudy's Perfect World. According to this, there cannot be any overlaps in the chains and every cipher text is in different chain.

Trudy gave a simple concept that suppose we consider DES. Here, length of chain is 56 bits and suppose we find 2^28 chains of lengths also ranging to 2^28. So, easily we can say that it will acquire a total memory of space for 2^28 different pairs.

Also, time required to find the solution will also be around 2^28 for each attack. Every time, we can find an endpoint from the starting point in 2^27 steps and in the same way we can look for K in 2^27 steps again. So, in this way, there will be no chance of failure of attack.

Figure 4.6.1

As described in the picture above, according to Trudy's Perfect World scenario, there cannot be any loops, overlaps, merges or circles in the chains. Every chain is perfect and every cipher text exists only in one chain, and only one single time

Real World Scenario (exactly opposite to Trudy's perfect world)

In real world implementation of chains, chains are not so well behaving. There can be loops, circles, overlaps and also merging in chains. Due to these real world exceptions, we can find the encrypted text we are looking for in the endpoints (EP) but this doesn't guarantee that we will also find the solution, that is, the key in that particular chain.

Figure 4.6.2

In real world, to avoid such situations, one can reduce merging by opting for different reduction functions at each point in the chains. In this way, there may be possibilities of intersection in the chains, but they will not overlap and go on merging and winding with each other.

Reduction function can be of any type. Say, one reduction function can add some binary string format to the output before storing the output in the chain. Other type of reduction function can be possibly the one that shuffles the output in random fashion at each point in the chain.

Figure 4.6.3

Hellman Analysis

In this approach we choose m start points and fix a chain length represented by size t. Now as discussed before due to the problems like merges and loops we need to construct large number of tables.

How is the table constructed?

Pre-computation Phase

TMTO is specifically a known plaintext attack. So we have a value P for plaintext. We consider these m start points as the key. So at start point we encrypt the plaintext using this key.

 $C=S_k(P)$

Now the output of the above step is of 64 bits. But the size of the key is 56 bits. So, we use a reduction function.

 $f(K)=R[S_k(P)]$

S is the encryption algorithm.

R is the reduction algorithm.

Hellman proposed this function as the one dropping the last eight bits of the ciphertext produced.

Now, this algorithm is applied t times and the result is obtained correspondingly. As storing all of these pairs will be a tremendous memory overhead we only store the first and the last column as pair.

Diagram below shows a pictorial view of how chains are being created, and the creation of chains starting from a starting point say SP, to the end that is the end point of the chain, named as EP.

$$SP_{1} = x_{10} \xrightarrow{f} x_{11} = f(x_{10}) \xrightarrow{f} x_{12} = f(x_{11}) \xrightarrow{f} \dots \xrightarrow{f} x_{1t} = f(x_{1(t-1)}) = EP_{1}$$

$$SP_{2} = x_{20} \xrightarrow{f} x_{21} = f(x_{20}) \xrightarrow{f} x_{22} = f(x_{21}) \xrightarrow{f} \dots \xrightarrow{f} x_{2t} = f(x_{2(t-1)}) = EP_{2}$$

$$\vdots$$

$$SP_{m} = x_{m0} \xrightarrow{f} x_{m1} = \xrightarrow{f} x_{m2} = \xrightarrow{f} \dots \xrightarrow{f} \dots \xrightarrow{f} x_{mt} = f(x_{m(t-1)}) = EP_{2}$$

$$\vdots$$

$$SP_{m} = x_{m0} \xrightarrow{f} x_{m1} = \xrightarrow{f} x_{m2} = \xrightarrow{f} \dots \xrightarrow{f} \dots \xrightarrow{f} x_{mt} = f(x_{m(t-1)}) = EP_{2}$$

Figure 4.7

Algorithm:

The algorithm for offline phase is as follows:

•M random points of 56 bits are selected. These serve as the starting point or key as in DES.

- For each starting point a chain of length T is formed.
- The starting point is taken and is fed into the algorithm.
- If the output size is different than the required size (56 bits as in DES) the output bits are truncated.
- The same procedure is repeated T times.
- The starting and the end point are saved.
- The above steps are repeated for M different start points.

Figure 4.8

Online Phase:

Algorithm

Algorithm for online phase is explained as follows

- This phase involves searching the table for the required ciphertext.
- Firstly, the ciphertext is truncated to 56 bits.
- Now the end points of the various chains are searched for the same ciphertext bits.
- If the same bits are encountered the chain is regenerated to find the t entry and the output of this t after applying the reduction function is the key.
- If the bits don't match then the reduced ciphertext bits are encrypted and reduced p time 1<=p<=T.
- Each time they are checked against the required ciphertext's bits.
- When a match is found the chain is generated T-p times and we derive the key from there.

```
Hellman suggested to follow the relation:

M=m*t

mt^2=N

m=t=N^{(1/3)}

Breaking the DES via this method was reduced to 2^38.
```

Improvements by Perfect Table:

In 2003 Philippe Oechslin showed that the performance of the TMTO can be increased drastically b using different reduction function.

The probability of collision between two chains decreased to 1/t.

Rainbow chains have the following advantages:

• There is an overall reduction by a factor of t in table look ups using rainbow tables, if compared with original implementation proposed by Martin Hellman.

•Therefore, merge free tables can be created using rainbow tables. Still, there can be collisions in the chains.

• There cannot be any loops in rainbow tables as reduction function changes iteratively at every point in the chain. This saves our time as we don't have to look for loops in our chains and remove them, and also, all values of encrypted texts can still be covered, and that too, without any loops or circles.

Philippe Oechslin further extended his work and worked to find the optimum values of the parameters m, t and l and worked to create something called perfect tables.Perfect tables have 0 merges and no memory is wasted.

Given M (memory available), N and P(required probability

of success), following are the optimal metrics that lowers down the overall time of cryptanalysis:

l=-ln(1-P)/2

m=M/l t=(-N*ln(1-p))/M Using the above parameters we tried implemented TMTO for DES : P=.86 l=1 m=34359738368 t=262144

Working of our Offline phase implementation

In offline phase, I created a C++14 function of Data Encryption Standard (DES) that takes input of plain text and key in string format and outputs the result, that is encrypted text in string format to the main function.

Along with DES function, I have also implemented other useful functions in my code such as binary to hexadecimal conversion and vice versa, conversion from string to integer and also, iterative binary search algorithms.

I have prepared a double dimensional matrix of 200 rows and 200 columns which represents 200 chains of 200 lengths each. After all this computational work, I created another array of 200 rows and 2 columns to store the starting and ending points of the chains. Then I dumped the 200 chains to save memory and sorted the array with respect to ending points of the chains so that I can perform binary search later.

Now, there is a field present to enter the encrypted text whose key you want to search for and using binary searching algorithm, one can easily find in logarithmic time that whether

encrypted text is present in last column or not, and on which index. This is the overall implementation of offline phase of time memory trade off attack.

F:\desmid\des_mic	l\bin\Debug\des_mid.e	xe							
Chains of length	200 each are as	Follows:							
			16400020004004	FERE200244CD4724	2561020004607050	000000000000000000000000000000000000000	055077400500004	D220205002640550	4220040007674212
		70004E0E12CE77E6		C30036D34ACDA724	366846330664647636	9070930037403932	03EZ77403FDC9394		4230046007074213
10000000107040000				9D101C020000A0C4		00D3E0FA10701329	320CD7E033F3AD27		
10F05CFEA100EDD5	A3D7DE93E0C2A66F			DF3A627AF9240491		CA7551E72142570A	23000000000000000000000000000000000000	40A29FE070F6142C	
AA1200A399339201	AE1AE7E1005E5652	440751 9CT 881 5DAL	53634060030606623	E400C6400020P141	E49055E7E32C96C2		204000046504600	00C920A111128072	61EA7E73DD41E3ED
21005656677660045	647CD9D0EE1ED522	6440002CEC141 CZAZ		220000620115600	E2004DECE7202200	A1E0E2D14A720A22	7699500050046603	D15500E0D20000E6	
	54470080910100333	ADEE/07700E10100	A20274505902EE16	2574242594665164	60EAE62ED79EAD47	DD/DC70620E2002D	2040E740646E1D47	C100210601706110	0910017E110201E
AD0010E20A6502D5	7044404000004040	4021407709212108	C64407EC94EEEEE7		9C725D1110E62DA5	6200100251006102	E46704E0DD000074	0DA260C0E0012ACE	EA70D202EA272524
400010100000000000000000000000000000000				7450702500020062	A001E05E70A0D74E	605670201500103	A7D206C640E15D5E	A0200C019912ACL	
106000000000000000000000000000000000000		971EE62ED56C072A		A01AD7C7A0211057	AD2550EA26DDA411	00107A30A123A120	7012002007221440	26521606410200	ESESENCEONSSCOSD
	E0E070CAEA6A20E6	E1ACA70550000A70	400400000000000000000000000000000000000	270000000000000000000000000000000000000	AC1EAE01E70EE4D2	100150000000000000000000000000000000000	250006ED45D1D22A	D00D363060EDE03A	C27C56E400050E61
AD2E651E2727D07A	E1AADE4EE115030E	0050625512025709	E2E562AE70D221019	19000LA991L9409	666AD900D7E4A962	1076069E01439439	255005020000000000		DOD/ECD62569100E
DOCD42D0ECE25072	55000E2005200054	6DA0A60202EDBECC		CESE1DEDED165100	E7044ADE41656044	E76000520A420420	6010000000570400	02040510D25DE6E0	CAD99066057050C22
C199E272A0DC620D	02070200202003000034	6C20275P0CE2461D	E26D22465E50050E	DA7200012EC17C11	21 AD120EAED64D2A	1910535244250850	201405950855784449	C2412EAED67C7EEE	AE05E5CA6A64A25C
201000000000000000000000000000000000000	0PDA72P49550C6EA	20000606660221061		2222002361460000	DSDER5780C720EE5		016200/00200601A	A9ED325C60065016	14E6222EE6CE000E
898E88E724C2848E	CRED44030588624R	049CC86049R1698R	D/9E970C9CE5E6E9	FA95583718056208	529CEE01D3700761	398CE/RDE00C51/7	1E2545010D43B78C	EC009BC72D0895E7	46C615E74E164202
CCA22E/R3777ED99	R5F4925364FFFR09	R450DDEC795CB452	CEC8E63138E5B549	228D32E767B563D4	9232CF1294R1771R	CE5EE6238C8D4995	0D54200108438762	D98DB880E2E10474	24E366454BEBACDC
E2R7774E7C2E0DEE	E40C04440B3809C4	253007RE138839//	0/004E/CCE22BE22	R816CD2CA0R3E7D7	0RA37C73EC1ECR12	733602D4558R68DF	376502/09C1E88B3	760/3046/4930860	RA15E35E77D15A3D
72861D2C7E0E2D/A	0088EBECE/032605	34D4044CD898C326		82C9C233865CE814	946B3B7496BC7825	FDA73502//AA1CF21	3/38E9/858D9455E	R/AF9/0E8893/563	6/29605D6986D/49
1656C8ER03E50587	0E680880744E7E84	55D13E9353E92B3E	CR1DE319D597C9R4	D044C1694DDE2E61	24566C42D823B5E6	E31E464E6CC8DE26	295C/185C/FEE89E	9/011E//DE8D29E1	839633F1F2384948
1050000000505150507	0100000014427204	5561525555152652	010001000070004	DONHCIOSHDDLLI OI	2490004200290910	LOTI HOM OCCODELO	2550410504211052	5401124401002521	05505521 25015110
4FA349FF3DF0C99C	ED895ECE38B08451	C9D31A3BBC9440BC	C174191085D6BA70	7642216FBA1F64C7	21B5117BD5943CDC	6A404C02FFF85F75	81BE4B777ED31B83	28CCA7062BA12D23	43484976CF2923D2
8FF29713BA8F6C22	FFF2FC312F308F2F	8A23B9BFF8BCB24F	D01A32E58A7E2ACE	7AD45FFB1704D49F	E5EDD979BA5E2DD7	56DD491DE54B0AC1	F454D0B5BD5BBB71	B87B274EB73C7D6C	2BE0C71A66A1724B
B881D70805D7F68F	7A57CEA50D18CE51	543E53EED363BC64	192C41A8607AD7C9	C3255A36119589AD	D9B637B2215C0A58	FAF8DØB8C2DØ1AC3	4E53BCED8CEDA083	FD1D564B87F70B96	C13D1B99BB1E9EA5
8277571BFF571090	FRF92F50F2D6DBDF	D49733A9F271C665	8233E650E7E3D62E	C348A4805473EAE2	F0FF4C2C360F0627	B6C9849701B8D6DA	FFFF17F01BBB9ABF	9F06852F63FD8081	FC9FF080483DCB61
1DF491B2F68F6779	1135602C776E8E25	7CACD722C59B2200	2D9987BE2D6D75DE	9082472A70F28F59	9F9321186712BC67	5BB447005221027D	6FB2AD4391F794F2	2FFCD7288679FF93	F98C77749A1C9A8A
335F37ACD3142211	2C7AF9B7FBF0A412	EEC650AD2251D2B0	83AF70D4AC6F4F2D	7DB3EE500025209B	D935B852D25B29D3	E336EDD7B0DE3E1C	88923DFF5910FB94	FA673D2DF449AFFA	3110CD4B225D4D65
F3458B3FC90CF4A5	F1108FDC943D942B	6543FB093F681D74	5A0697A2BCAC427A	DF4606789923B620	A4703C9390B69F8C	04C72108FA5BF02B	2C8250D65FF2F3D2	306C3C72AA58D649	7A40F44945CFCA85
3823F88F77BF4098	64968A214FDF1FB3	957F0BF77D0C2495	4C7BA041109C713E	2E29B627DBACD08B	986305ADF0F88580	CB57E4A609B4C916	C6B84483C6C98FB6	FCB8730D530FD91A	069F4BC41A5425BC
FCF6154428678113	64B12497BF1A9560	7484074408FFD2B3	3BDCE4088E810143	A29D08BDD66AA069	5F8B65831066D4A3	AD04F049AD7B5077	D0222F116B9153AF	83B6CDC0F2FF0CC6	E96E0AB87EDA2A03
F7646CC26BBC9BFB	22B65E6AEC32801E	574203E02345B68E	5C1917367566854D	5070B5E6EE3E3E9C	6F8A2C9265AD78BD	63B233E44CA08DC5	9198BAFD530054CA	3F84FACFF7FF2AFD	62C9FF9F7AB937C6
A7B44511427FFB9B	9097953F3818F5D2	5A797B1358D79AF1	Ø8D5466BAFADF3FC	AA0CABE590B9206E	9FDAC86C814928F6	90DE05867B380A95	1800101067502280	D36995360706D730	4BB24429CB213625
96F95C9262B396DC	81BE9FF198790600	58267A42A5622799	8162E064D3D51549	C41A0AF40BC85766	A8796B48167C1FCE	6579BDCE334B3778	C42F70938083ABB8	94B590B95C4930DE	2D057186406FA169
01815959C70C7611	187A78C2DC954414	9C817BA847BDDB4A	0FC4E459A3273B69	5E5337D74BF2A1A8	61C0EC13114A97A8	88DAD41B5FC9E22E	DD4F321A30BAD643	C9B2AF01A0BAD600	EF91B5B54AF0A5FA
DEC9032D4C45BBF2	3DA36067B34AA486	43D685CAA288AE7C	863503AB4F89D50D	3BEEC5B55E9CDCEB	AB3BBB2AFEBFA4E4	F2B12CF2C274C66C	469BE620BF7CCBB9	30FB1F333A7B4C8A	5B4AD19A3A1FC4E7
335A4202953BBD87	6DB9F1246884EC9F	00122DCA5B1D1394	4571303657201357	31C7CAFA24134CCD	44C7480A8A973E48	35C048E025018C59	5E2625B9FB6E1DB6	D8AFA1588BEE16FE	485051C932387D98
20C7FBFADBFC1156	396CB2BC3F7A9DCB	36D495A083851450	C9730F31F78FA6E0	8E1C3BB0C75EFA10	BED4BA21AD2584D6	21E145EF7BE2B22D	A7D80612633F5BD8	9AAF899350C4932A	CBF6D4F926262EE3
298149801F21493C	673CFCA0E02477F6	65DD3C832CBEB1E9	09C637DA351C62BC	0A620C81EBE70598	13EB3BA8FDF9218C	21CBEFBDFBA0951B	0CB3FF11312EF94E	80D2FB92A00ACA8F	4FC3CAE09F13292E
46FC86CDBA2F1D41	EDA0E1112BC0E1C9	0C1FF74AA1D362A0	16DCCBD4D4DF6549	38D02FD14816FAFA	35C8EA099D46C371	73E60A6402268100	BCC1E24F647953F2	7C2A971150F84043	4B377C25DCEDA969
6864E176CAA436AE	EF09256E35C78E67	1F8764752932B447	5272E7775E8AE1A6	C7A1655A0EA92A0F	E935B589D15E2D06	8AC962F738BBCAFC	F725E8BBE55CCB5B	F76EF31B423A1F16	9B5E2985FCC31F0D
40300F995A95FAB5	648A07277C1273F1	AA49341323C7827F	60BC914C18DF6D11	974ED9A8DA21F056	1992715BD0C257F0	E7B2477151C22A8D	16139F1E8928899B	FD7744B9A4319FF9	861BB584DB0C43AC
A290FEC684CF648A	2FACF50412DF4615	F2AAF86BCBD61A00	4552C75271EAA4D2	78EFDC8952E95C02	2C4244696D04921E	E01D922277749998	F6A694EC85590359	387C6B3CE789C89B	BF93B9A8348BCDA1
6C653A222B575C49	2B9E2770A8A432F1	8132D3A6CD5B60EE	CFEF7E906A4A796E	AFE3CDC9B0E5DE32	BFD8BFF2FF028CD1	16091CECA9A83FA2	ADECE09399BE8B5A	D6A84435C814D212	A90DB35FE196B6F0
18E4870E0A26D5D5	ØAC12FØFDE592A27	ØBEB6373DDE50BD5	742B17EEB57E5076	E8F444D482972C1F	89C4E6553F9433CE	030D0FEB225A1CFD	B0448DEE698C5B7F	1F81FAA37B858FC4	689BBC2E53E42F7E

Creation of 200 chains with each chain containing 200 values are shown below

Figure 4.8(a)

Computation of binary search over the array containing first and last elements of chains in a sorted fashion based on end points

Figure 4.8(b)

Working of our Online Phase implementation

In online phase, file handling is used to transfer the data from offline phase to online phase. Here, one can enter the index at which the encrypted text is located to compute that particular chain from starting point to (t-1) length as the (t-1)th index will hold the key for that particular encryption.

💶 F\desmid2\bin\Debug\desmid2exe
14AEFEE690A390B3 D68AA0993DA780EE
6488000AD0105568_DAA802D388499902
2332D58CABBA14E9 DC7C2C3035F45FD5
963CDE21ED642FEA DF7980DF06420654
38A58F1FB5931DC1 E0EC708BBED756AF
DC4195ED09A8C085 E189E2289EE6FACF
4C0628282369F0E4 E2DE558AF4B75EDD
14428CA7A3F14E04 E304D6AA3E9A6724
80F27F50546E5F1B E345EFD160B43C2C
F631FB9860C1DCF1 E4150E27D99FBE7E
9E82982497CF7CCC E4A374F8AD6AC73C
3E25A15A3663E5E1 E5F2EC82A99CAB03
67477401FC04B797 E80B5788BE58069E
D6CF877580C486CA_E856ACCE9549D57B
4D3954192CF8E977 EA426931488620F3
4F8CC9212249C4FC EBC9E2E26344F69A
062001F888EE4020 EC3888F1E58C8648
C304F80FE44D6F49_ECD7FE7B45A11290
C9A819AF33809609 EE7A32EB728E5838
S66727/9501640106 EEA0940(A9/1/1-18
90.344799(020704 EE3283253225324884
NASHUJJAHU / L/07/0 E/80043UU/JUU8U3
UF 10617 / BUDE 1096 F012040LEFF0C3FF
03A/0/07/1/30/00/3 *03A/0023/1/00/3
GC1880245E098A2 F2100ED89925895
884CF0436A557332 F3843F0FD29C2146
684290E120C15108 F3DEFE0180EC98D4
2C308BAAA3955AA1 F6E81AA86E781DE1
C58C77429C5673C9 F7704DF1F69CE81F
960102001073CC45 FA6798F15FDF2886
CE8C1BC572F399F1 FB56FB3A1CD5D518
A34D4DCD09849481 FD0D2E9692C27104
7386E90F95EE1730 FDA07ACE6D341A1E
6804AB73B550CADA FF96ABE11BBBAD9B
enter the index where ciphertext matches in the table.
Starting point of the chain is: AND/F21854EC6342
Required output is: 55950070M&IFC44
Process returned 0 (0x0) execution time : 5.743 s
Press any key to continue.

Figure 4.8(c)

But if the user enters -1, that is the encrypted text is not present in the endpoints, then we need to compute DES on that encrypted text again and look for the output in the endpoints.

We have to repeat this step till we find the encryption among the endpoints. Also, we need to keep a track of number of times encryption is computed again and again, say n. Then, we need to compute the chain length up to (t-n)th position as the required key is present at that index. This is the overall implementation of Online Phase in our time memory trade off attack.

Figure 4.8(d)

Our Rainbow Tables Implementation

In implementation of rainbow tables, the major change as compared to normal solution proposed by Martin Hellman is in the reduction function. In this implementation, we need to change the reduction function at each step so that we cn avoid loops, circles and merging of chains.

F:\rainbow\bin\De	ebug\rainbow.exe									
0011111101000100 3F44B8CE0AD4DC	10111000110011	100000101011010	010011011100							
Chains of length	100 each are	as follows:								
3F44B8CE0AD4DC 60 07B9497CD5CAD4 20	B4E60E3607C6F 624C6A1C6E01A	B32935D0C16650 DA070F9D9690C6	410C7D2242E75E 562801A8DB8020	D4B3828D9575F0 8E1C3D93C261AA	47888EB707FA3D 39F2FD8D9900F4	671D1A96488A86 08DEFDCDDFD580	120B4FB7F5B138 81377BADC23744	165CD3002DCECE 74095B998F48A0	275268A4E61F06 726AEE78787747	
E18D107B1C11C8 B	A36C6150A9370	D120626FDD68D5	D05A14B8A0BE50	61450AB8B3CC8D	8AE8D9534B8754	F65F7B38D47F4E	C01F5F161A1FEC	E619D7E7C9D8CB	A3E893BDCA6E71	
9BEAFFB60623EE 2	A13840D8775CE	AA6C378A731A94	A7F1017D3E1154	A5057647558B94	10F381F711D8F3	35D51EA094BB06	59BF51248F4433	838BAE0800A635	132B9FD9E3A319	
90FE5155984BD2 5	CC3371A42226F	2D8B52CE63E411	42217F1B87A012	BA18AA75E460A6	A624887A0BA7E3	4834304047AAB4	FD0B69BE3A390B	ECC14B0262AC2E	5E59A723996A61	
D8AA1DB948A222 4	FCC8B991CA1C2	F4E86F0D432C9F	8AE48C8353C62A	1DCCØED85758FF	A7B7F295F84729	F9DEB59566F34B	9AA7513636BFCD	1758D59DB8A927	2F0779FF6CC8F1	
076E89742DF611 8	8B86DC109BB7E	726A0B93EFCAFE	B58BACAB44AD41	1D6128EB4CE6C7	3EB0688C589F20	EF8EA4C9641CC0	6B654F7D88B15C	AD6118A20FFD9A	D10E88C58D0A5C	
E93454913E9DEA 3	A202975616BAE	3136AC4FC9EA42	92D46D0767A009	511348F978C12C	77956E2DE9A848	309AE9CED4BCDE	0BDA7F4F93D099	F94891E777B393	D723D2A484C462	
A14E07E9055E20 6	1A480F306755F	DØC26ED3D2ACDE	F139C0D3602EE0	C66C16B9CF5F1B	4079170B1F4C75	6BC2EDC8E5ECE3	CBC9880B073302	ØC2F43493252E6	9904E066D50828	
0B6842B3977A9B 1	889E4686F120E	29A84C70F1ACA4	F4/B2810/6A984	CABFFE/52B7/D8	B16A6BAC597AE4	5525E11595FB66	A/4825E95A646D	F730DEB8F263A2	640E4E5BFCFAD5	
198F128FDE3C67 B	EAF6CC30B8EFB	29B723A8DA89C2	131F4642EC27EC	9DC45D6B5870F3	1CFFD4D31D5AE1	D4FC6132CD4ABF	679A5017BEA361	997D68E92B5C44	AFE69F0DE16778	
F51A89D8E0CDCA 6	9B29C6A9AE19F	FDC2CD9F05EE84	4891FC72E68815	F5090B96A764B2	F6D3B17A9A50BA	F98845D360D01F	B9A2AC8117B6A4	A96102F2D2FD1C	ECAA3BE45E04F3	
0BE4FD4BDFD800 F	5853D7C70EEFC	0C5E5D0B5CBB64	5499529972CBDA	6E79BD2543CA16	DA644932E90540	46102000030136	8DA7F5BB54CB57	189FF5EAEB76C1	C728ABCBF6171E	
2EE8D443285E96 E	00A5A211101BD	4856394983C7C2	B823995FFA5613	7139BBFBA8F5A8	79C1A4A3BD5061	16BFC5AD176EBC	89BB05B4C1FA58	695CFECA0B817D	04F446DAA3AD75	
C0927D99674595 F	293299458BBB1	46A02AE2489700	41EF1C8C6DAF87	2ED3331EC954FC	D9DE2E8A7F9363	7146BC23060ASA	E4F14371206A9C	AB96BBEBEB1791	58F2000B437AA0	
7C5B3236D9FB1E D	7CF64BF77767F	22BCE29300941E	FF65EC9D4D1E30	22E5B07C14ABD1	48DED8E237B901	19F33BB65F460E	3FDF00475EA218	98253F6D20E2C6	B24933273F0A70	
6CC3B81F871FFB 3	CC12CD6BFF138	D53C9618B51BA9	69770E3EA2B5D5	D811D68FE8C032	79C5AF82FACAD9	F858B53A3A591D	9079442EFF41E7	19C24607F9349B	D32B81703BEBCE	
47256A69F66FF0 1	FDD20FAC01DFF	4DCBB408085D79	EAF1CFA9C35BBB	B1B09A0A07365E	AFD256992DC377	18572C226BC1A9	6DDEC04DCD3258	9AD39100457F12	29D31374F3238D	
3A748C34D3B543 5	62CA09DED6E9E	BBDB5AD7D92407	AD9B10E61CD2B7	5D5679FCFADD37	55DCA8A44DF705	E8DØAF5BDF9C45	71586F8A748E5B	230A5DD8A88ECA	655AF57DC8434E	
56CEF0450266CF C	4F4AAC7A638B0	C79BDE9AA5810B	9384ABC8EBB687	E0135C434AFA58	CF9CDEA3659CC7	69430CC4109A32	48DE93034376E6	1922A5B5A1B86B	57F73C4C984731	
099B4815CE5E73 A	12190D0569F2A	64B0C9EA0C492C	6CCEF85D385433	76061684523328	34C376BD436DA9	C70A47F99770AF	DECB9BF315DF87	2285BF7043FDD6	78C85AD8CC76E7	
39F0582980B112 B	9812B326BF69F	6BBB1A4717B29E	51FBCA05266D93	CACD75A667F0CA	7BF24F59F8E02D	CC2DBB7F55D3D9	E6E5EB9433C896	FC46A56CCD7729	CE2BA21E973C26	
902E129A596B61 1	3A8D1924631BB	2B8686A621155C	3E07C923A766BF	75B984A99725E2	82850D9CFE33ED	18EAAD72381F18	C2C213A31364B0	E9EF1029B16C06	C64B47430062AA	
284E9D1A459477 A	EA01263E1F6E7	2A82E66F911DC7	9709E7439C5C71	60D9C7AB871CE7	FAF7885ACB5795	468C78C97093FB	2B8F1A4EB7F91F	56FBE1BFA9C15B	BF33679B98885E	
0B54EA80126FDC 8	28AD96C72114B	33C361F914BC44	E497025B4A0F09	3CE42C3B184B3A	C7ADE2D0B403C2	6D19230A0A7474	F66E5E02596CAC	D8159FD9C144B9	04A316FF0EF44D	
6216C12CD31554 7	924CD42EEDF60	40FA0AB0A0E1D3	9EBC11BA1378ED	304F13FE48BD40	BCABD1BE4459A4	669E72C5F0943B	65F7F8A1677DC3	240ACA540AF404	3831F0870BD3A8	
2E92E3335BADØE A	DBA382D58957A	D7DB5927500276	4B955FE4EF1A27	6B74657B2B8555	C2386C0FD779EA	99DEF6111CD1B6	4E52EDD6ADE7C1	7F92F2BA3BA0CC	D4DD538FF6BC1C	
8C3DA484438219 6	70CC4B3797A34	2E39450181A689	11A3D8D06FDE71	Ø8CC81C3ECØA49	AD0768E1511087	42CC75580812C8	B2FEE621C0599F	AA59601592FA3F	399386E36A311D	
EZAZ136F/EZ32B B	E11A5CF7E4B69	208C950C8C2D85	DAB1DA86FA012B	F43C0AC5F08092	BF02B01511C/A4	7A87BA80438325	310013FAC4810E	134C2/0/94AF58	830BCB63852389	
EA8FBF23583597 9.	317B63DF361E9	E1C8ECCC913610	115E73E6855E1F	FDE0F6AA5E0DA8	887D41C9EC1C4A	22E58E95319F02	AA85FA189BCEE6	8E1363160430D7	136DC184C47BA1	
273A86CCF1C980 1	D24B46C4F1A70	E81266FD84AB84	3FDA30891EFDEA	92EFBE1EC5FE12	C56BFCDE77012E	4984613CD2F665	B2ABB28AD7E7EC	BCF78882F71BFF	D28422D1CF36C8	
9C40E490648948 3	EAF 3AAF 0009DA	F4B31D1A8D6B58	02B692FEB82422	5088EFD219DFF4	6873A05229438C	E7FFCD9020E448	CC15083A13BE27	29C5F7AE065E1B	4CE11111C4CA24	
A20015D7A2B074 1	809333307B25E	79AA901854D591	897B94CF8D1BE0	916A8C2935C367	77DA8640A158FA	83B5747C9E6CC6	315479A7912C6C	C92EA73F27EC34	625F640B42CA48	
EA125ECEC2B561 2	B9ADC1D4C5E53	A928DDB9A11546	6CE25FB96EE37F	AC1651C8DFB759	D237B88047C59B	FD1F2D5AA35CBB	89B64B0F14FCEB	82B15ADFFAC463	BCCFC4F494FB15	
867028876926CA 8	E439C0ED41F30	76CE950357C6BA	61E7489EB121F2	94183793562502	AB6BAC8D275F96	FA06226CB2E433	D76286E80A57AD	F73C0D1C0AEDAA	0CAAD8825A3912	
71F07821E984F8 3	005D2685D7B16	88E5646C79340D	2C267E719DE5DB	F40231694744A7	D8FE231D8E476C	F3BAF0B71E7A43	604ADCF91271ED	59DDBF287BD6BA	62525BC9144C3F	
6E6DED9BB2CBF0 1	C8407611B18AE	F95D355273D222	3E3FDE77BC65C0	D2C82DC439911E	98D2638F233245	74DF6A3A57612F	F4ABD59A28D3FB	17B15816CEA207	02C2F8041BDB51	
B438EE512D6F6F 2	SASSBFB13CSA6	EØA58CAB7AC3AB	460A0F5486C19B	F2725626DDE9CC	FCD061E99E6458	24ABE18184663C	ØDEEDØ2978FEB5	009EE1F687642D	E9ED64CEBFEFE5	
652D1BB0D4E8DF 2	71EFDB99C3BA2	A765D23E0A98AC	01FDCDE3ABAA16	4FEFF0210BD275	3006FE961A6659	0678B876B4FA1A	9DCDCCAAD092C5	DC3B6E94335F88	7171D5DB4E4846	
A602B08809450C 3	0F02CEA1EA797	82F6A33D496BE8	00A9E1467DB063	95AD46ED0FA6CE	AFE65822DF43A5	6E81128CA4E392	DFAA8C41B589C1	B07ED5CC4B13F6	BD5B11AC278EB3	

Here, we have taken a counter or iterator, and after calculation of encryption at each step, we are adding the binary forms of both iterator and calculated encryption and then we are storing the final result in the array representing chains.

Now, because reduction function is changing at each step, there is no chance that two chains can have same output after reduction function at same point in respective chains, so these chains simply cannot merge. Also, due to different reduction functions, there are minor or less chances of getting loops or circles in the chains.

Success Probability

Consider 3 different parameters say a, b and c where a are the number of random starting points for every F, b stands for number of different encryptions in each chain and c stands for number of different tables or random functions F taken into account.

If a, b and c are chosen as 2^k/3 then the probability of success is near about 0.55 in lower bound. With this lower bound of probability, it requires about 'a*b' amount of memory and 'b*c' about of time for each TMTO attack. Also, for pre calculation, that is offline phase, it requires about 'a*b*c' amount of total work to be done.

Success probability can be calculated using the same approach as used for occupancy problem, that is, if you have to put n different balls in m different boxes, what are the expected numbers of boxes with at least one ball in them?

If the size of the key is considered as k bits length, then the success probability can be given as follows:

Probaility (success) = $1 - (e^-abc/k)$ Where a^b^c is the amount of work required to be done in offline phase.

mtr	P(success)
0	0
2^{k-5}	0.03
2^{k-4}	0.06
2^{k-3}	0.12
2^{k-2}	0.22
2^{k-1}	0.39
2^k	0.63
2^{k+1}	0.86
2^{k+2}	0.98
2^{k+3}	0.99
∞	1.00

The overall ranging of probability of success is defined in the table below:

Table 4.9

5.Test Plan

The proposed system needs a high processing power. We are interested in implementing the perfect rainbow table method. This method involves creation of the table for which the DES encryption algorithm has to be used several times. As per the proposed scheme we choose the appropriate parameters and create this table. For the creation of the perfect rainbow table approximately about 13 days will be consumed.

The expected encryption rate for our DES implementation is about 2^{36} encryptions/sec. This will generate a large number of encryptions. The total size of these encryptions will be about 2^14 GB. Do we intend to store all of these? No. (Even if we do, we won't be able to) !! We will just be choosing a limited number of starting points with a fixed chain length. We will save only the pairs consisting of the starting point and the end point.

But we have another question in front of us. How are we going to achieve 2^{36} encryptions/sec. Our optimised implementation of DES incorporates bit slicing. A technique suggested by Philipp Grabher, Johann Großsch adl, and Dan Page in 1996 which can significantly increase the speed. With our optimized implementation and bit slicing we achieved a speed of about 2^{22} a Intel i5, fourth generation machine. Now for increasing the speed we would be needing dedicated GPU's and need to code them for our requirement. The preferred system would be a workstation consisting of 8 GTX 1080i GPU's working simultaneously. Where can we get these? Provided the situation we don't have a workstation

with the requirements mentioned we can hire a cloud system to do this work for us. After considering several services we have decided to go with **vscaler cloud services**. The cost involved certainly is a major factor involved here.

Following metrics for analysis should be used:

- 1. Money
- 2. Time
- 3. Success Probability
- 4. Availability of resources

Now different approaches could result in different results.

Metrics If Hellman Analysis Is Used

1. **Money**: In Hellman Analysis the problem of merges and loops prevails. So we need to create a large number of tables. Because of this the amount of memory required increases immensely. The cost for the solution as mentioned in the original solution was about

\$3.5M.But with time the cost of various hardware components has decreased and the native implemented solution will cost about \$12000.

2.**Time**: The pre-processing phase of the Hellman method takes a longer time due to the various problems mentioned above. Nonetheless, the solution decreases the total time required to $N^{2/3}$ for the pre-processing phase. The time consumed in the online phase is equivalent to time of searching. This is about O(logN+t).The time as per today is equivalent to the time involved in the pre computation phase which is about 100 days.

3.**Success Probablity**: The probability of success as proposed in the original paper is (m*t)/N.[m-the number of start points, t-the length of chain, N-The total search space].But merges and loops decrease the success probability to some extent. If we take sufficiently large m and t then a success probability of a range between .63 and .75 can be achieved.

Metrics If Distinguished Points Analysis Is Used

1.**Money**: In Distinguished Point the problem of merges and loops still prevails though to a lower extent. We need to create a comparatively smaller number of tables. The cost for the solution as mentioned in the original solution was about \$100k. But with time the cost of various hardware components has decreased and the native implemented solution will cost about \$9000.

2.**Time**: The pre-processing phase of the Hellman method takes a longer time due to the various problems mentioned above. Nonetheless, the solution decreases the total time required to $N^{2/3}$ for the pre-processing phase. The time consumed in the online phase is equivalent to time of searching. This is about O(logN+t). The time as per today is equivalent to the time involved in the pre computation phase which is about 47 days.

3.**Success Probablity**: The probability of success as proposed in the original paper is (m*t)/N.[m-the number of start points, t-the length of chain, N-The total search space].But merges and loops decrease the success probability to some extent. If we take sufficiently large m and t then a success probability of a range between .7 and .85 can be achieved.

Metrics If Perfect Rainbow Tables Are Used

1. Money: Perfect Tables tackle the problem of merges, loops and false alarms. This helps in reducing the amount of memory required by a large amount. Moreover less work needs to be done even in the process of creating the table. The total amount for setup is as minor as 1000\$.

2. **Time**: The pre-processing phase a comparatively less time because of lack of merges and loops. But some time has to be spent on keeping a check on these merges. Yet it is comparatively smaller. The total time consumed in the precomputation phase is about 13 days for a success probability of .86.

3. Success Probability: The success in the range of .1 to .999 can be achieved.

Data Required:

Time Memory Trade Off is essentially a chosen plaintext attack. So, we need to know a plaintext in advance. Generally we try to exploit our existant knowledge. This included exploiting our knowledge of various protocols. We know that some things are already fixed as the part of the protocol. This helps up to attack all the messages and data falling under a particular protocol.

6. Chapter-6 CONCLUSIONS

6.1 Conclusions

Thus so far in our project we have done and covered the research part. We read and collected all the required information to perform this. Moreover, we have constructed a robust framework which will help us to proactively deal with the various problems. Having developed a good workflow we can perform the project provided we are provided with the necessary resources.

6.2 Future scope

Being a generalized attack, it has already been established that we can perform the Time Memory Tradeoff attack on most of the algorithms present out there. Hence, this approach can be easily applied to any problem with a defined search space.

There are different encryption algorithms present and being used such as 3DES and AES on which naïve attack approaches such as Brute force attacks and attacks storing all possible results do not provide efficient search solutions.

But with Time Memory Trade Off Attack, we can easily compute the time that a much better attack can take to break such encryption algorithms. If found that algorithm can be attacked in a feasible possible time, we can head towards a new, safer and much secure encryption algorithm and should replace the existing one in real world scenario as usage of such an algorithm simply means we are compromising with security provided in real world cases, ranging from banking encryptions to our daily used social media passwords.

REFERENCES

1.https://en.wikipedia.org/wiki/Data_Encryption_Standard

2. https://en.wikipedia.org/wiki/Time/Time_memory_tradeoff

3. https://ee.stanford.edu/~hellman/publications/36.pdf "A cryptanalytic Time-Memory Trade-Off"

by Martin E. Hellman

4. https://perso.uclouvain.be/fstandae/PUBLIS/2.pdf "A Time-Memory Tradeoff using Distin. guished Points"

5. https://eprint.iacr.org/2008/054.pdf "Variants of the Distinguished Point Method for Cryptanlytic Time Memory Trade-Off"

6. https://perso.uclouvain.be/fstandae/PUBLIS/74.pdf "Time Memory Trade-offs"

7. https://crypto.junod.info/jacm08.pdf "Characterization and Improvement of Time Memory

Trade-Off Based on Perfect Tables"

8. https://lasec.epfl.ch/pub/lasec/doc/Oech03.pdf "Making a faster Cryptanalytic Time Memory Trade-Off"