CRYPTANALYSIS OF SOME BLOCK CIPHERS

Submitted by

Abdul Ghani Haji Naim

for the degree of Doctor of Philosophy

of the

Royal Holloway, University of London

2014

بسم الله الرّحمن الرّحيم

This work is dedicated to my parents, my wife, Norbahyah Kumalawati Haji Kadir and my family.

Declaration of Authorship

I, Abdul Ghani Haji Naim, hereby declare that this thesis and the work presented in it is entirely my own. Where I have consulted the work of others, this is clearly stated.

Signed .....(Abdul Ghani Haji Naim)

Date:

Acknowledgements

I would like to thank Pengiran Dr Hj Nor Jaidi Pengiran Tuah and Dr Seyed Mohamed Buhari for their help in programming. I would like to thank Prof. Peter Wild for his patience, guidance and comments for my thesis work. I would also like to thank my current supervisor, Prof. Chris Mitchell for his patience, understanding and guidance. I would also like to thank my PhD examiners, Assoc. Prof. Konstantinos Markantonakis and Dr. Chan Yeob Yeun for their valuable comments. Last but not least, a special thanks to my wife, Norbahyah Kumalawati Haji Kadir, for her full and tireless support throughout the making of this thesis. I could neither continue nor complete this thesis without all your help.

Abstract

This thesis concerns the cryptanalysis of block ciphers and we look at two important examples: the Data Encryption Standard DES [80] and the cipher RC5 proposed by Rivest [84]. Although these ciphers may have been superseded by recent advances, there are lessons to be learnt in the art of cryptanalysis by studying them. The first half of our thesis focuses on the reduced variant of DES *i.e.* the 8-round version. We discussed the implementation of DES and various cryptanalytic attacks on 8-round DES such as, differential cryptanalysis, linear cryptanalysis, Differential-Linear cryptanalysis and also the use of multiple linear approximations in Differential-Linear cryptanalysis. By performing these cryptanalytic attacks on a PC, we were able to gain insight into the processes/steps involved in performing cryptanalysis and we were able to gauge the feasibility of the attacks with respect to the computing power of a normal PC. We discovered a different implementation when we used multiple linear approximations in Differential-Linear cryptanalysis of 8-round DES which gives experimental result comparable to previously known result based on similar attack. The second half of our thesis focuses on a comparative study of cryptanalytic attacks on both DES and RC5 block ciphers mainly concentrating on the use of distinguishers to determine the strength or weakness of the block cipher based on known statistics.

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

In this Chapter, we give the motivation for our research. We describe the state-of-the-art in block cipher cryptanalysis. We also describe the contributions of this research and present its overall structure.

1.1 Motivation

Advancement in computer technology has significantly thrusted block ciphers to the world stage. Ever since DES (Data Encryption Standard) was first publicly adopted as an encryption standard, we have been trying to improve security through the use of different encryption algorithms while at the same time trying to find security weaknesses in these algorithms by inventing or discovering ingenious cryptanalytic attacks. The importance of encryption algorithms, particularly block ciphers, is more evident when we surf the Internet. Safe transactions in e-commerce through the internet would not be possible without the existence of encryption algorithms. Since the Internet is worldwide, knowledge about encryption algorithms and security, in general, also spread throughout the world. The global impact of block ciphers has made cryptology, the science of secure communications, very important.

Cryptology has two main branches – cryptography and cryptanalysis. Cryptography is the study of how to design algorithms that aim at providing security goals such as confidentiality, authenticity, integrity and other security-related goals for data transmitted in insecure communication channels. Confidentiality protects data from leaking to unautho-

rised users. Authenticity provides assurance regarding the identity of a communicating party, which protects against impersonation and nonrepudiation. Integrity protects data against being modified (or at least enables modifications to be detected).

In modern cryptography, there are two types of cryptography used, *i.e.* secret-key (symmetric) cryptography and public-key (asymmetric) cryptography. In secret-key cryptography, a single secret key is used in the encryption algorithm for encrypting or decrypting messages; the sender encrypts a message using a secret key and the receiver uses the same key to decrypt the message. In public-key cryptography, introduced by Diffie and Hellman [35] in 1976, each participating party has a pair of keys, one called the public-key and the other called the private-key; the public-key is typically published in a trusted directory, while the private-key is kept secret. When using a public-key encryption algorithm, the sender uses the public-key of the receiver to encrypt the message, and the receiver uses his/her private-key to decrypt the message.

Cryptanalysis is the study of how to evaluate or break cryptographic algorithms. It helps in the design of more secure cryptographic algorithms because we can learn from the weaknesses that we discover in cryptographic algorithms and so develop new designs that avoid these weaknesses.

The block cipher is an important primitive in secret-key cryptography. The main purpose of a block cipher is to provide confidentiality for data transmitted over an insecure communication channel. A block cipher can also be used to build other secret-key cryptographic primitives, such as stream ciphers, hash functions, message authentication codes (MACs), and cryptographically secure pseudorandom number generators. Block ciphers are also widely used as a fundamental component in public-key cryptography, information security, network security, computer security and other security applications. It is therefore important to investigate the security of a block cipher algorithm against various cryptanalytic attacks.

1.2 Block Ciphers

The term "block cipher" refers to an encryption algorithm that operates on "blocks" of data. Typically, the blocks of data are 64, 128 or 256 bits in length and they are transformed into blocks of the same size with the use of a secret key. The block cipher

encrypts a block of input *plaintext* P into a block of output *ciphertext* C using a secret key k. This is typically denoted as $C = ENC_k(P)$. The reverse process, called decryption, uses the same secret key. This is denoted as $P = DEC_k(C)$. There are two important parameters for a block cipher, *i.e.* its block size, denoted by b and its key size, denoted by κ. The block size b determines the space of all possible permutations that a block cipher might generate. The key size κ determines the number of permutations that are actually generated. So, for a block cipher with key size κ, there are 2^κ possible keys and each key specifies a permutation of 2^b inputs. There are $(2^b)!$ different permutations on b-bit input blocks which is approximately $2^{(b-1)2^b}$, by using Stirling's approximation. For a typical block size b and key size κ, a block cipher only provides a tiny fraction of all the available permutations using a highly structured method. However, we expect a good block cipher will be able to disguise this and we expect a randomly chosen key will be able to "select" a seemingly random permutation from among the $2^{(b-1)2^b}$ possibilities. Apart from that, we also require that keys that are related in some way should yield permutations that have no discernible relation between them. For a secure block cipher we expect no exploitable information about the encryption process to leak. Such information might include information about the choice of key, information about the encryption or decryption of inputs or information about the permutations generated using different keys.

Much of the work in block ciphers can be attributed to the work of C. E. Shannon, particularly the landmark paper [87] which introduced the idea of *confusion* and *diffusion* for practical cipher design. These are still the most widely used principles in block cipher design. The idea of confusion is to make the ciphertext statistics depend on the plaintext statistics in a manner that is too complicated to be exploited by the cryptanalyst while the concept of diffusion is to make each digit or bit of the plaintext and each digit or bit of the secret key to influence many digits or bits of the ciphertext. Block ciphers are designed to provide sufficient confusion and diffusion. It is the job of the designer to come up with a mix of components that will help achieve the goals of providing both confusion and diffusion. It turns out that the basic operations of *substitution* and *permutation* are particularly important in achieving these goals. Most block ciphers contain some combination of substitution and permutation. However, the exact forms of substitution and permutation vary greatly for different block ciphers. Substitution is often used as a way to provide confusion within a cipher. Permutations are often used to contribute to the good diffusion

in a cipher. The mix of substitution and permutation is an important component of most block cipher designs. An important class of block ciphers called *substitution/permutation networks* or *SP-networks*, consist of the repeated application of a carefully chosen substitution, a carefully chosen permutation, and the addition of key material. While the user supplies the encryption key, it is a good design principle to reuse as much of that key material as often as possible throughout the encryption process. It is the role of the *key schedule* to produce a series of *round keys* to each round of encryption and these round keys are computed from the user supplied encryption key.

So far, we have only discussed the role of the cryptographer, which is to design cryptographic algorithms. Now, we turn our attention to the cryptanalyst who tries to examine what the attacker is trying to do. In the most extreme case, the cryptanalyst will want to recover the user supplied secret key. However, the cryptanalyst may be satisfied with much less than that. With this in mind, it is possible to establish a hierarchy of possible attacks [60]:

1. TOTAL BREAK: The attacker recovers the user supplied key k
2. GLOBAL DEDUCTION: The attacker is able to find an algorithm A which is functionally equivalent to $ENC_k(\cdot)$ or $DEC_k(\cdot)$
3. LOCAL DEDUCTION: The attacker can generate the ciphertext corresponding to a previously unseen ciphertext
4. DISTINGUISHING ALGORITHM: The attacker can effectively distinguish between two black boxes: one contains the block cipher with the randomly chosen encryption key while the other contains a randomly chosen permutation.

An attacker achieving a total break can achieve all the other goals. These attacks have been ordered so that achieving any given goal automatically achieves those that follow. Conversely, if an attacker is unable to distinguish between the implementation of a block cipher and a randomly chosen permutation then we have, in some sense, achieved an ideal block cipher.

In 1883, Kerckhoffs (1835 - 1903) laid out six requirements for a usable field cipher [54]. These have been reinterpreted by comentators and the term *Kerckhoffs' Assumption* or

Kerckhoffs' Principle is now used to refer to the assumption that the cryptanalyst knows every detail of the encryption mechanism except the user-supplied secret key.

We might also assume that the cryptanalyst has access to different types of data. As we look at attacks in more detail it will become clear that we need to specify exactly the kind of data required in an attack. Not all kinds of data are equally useful, nor equally available. Attacks can be classified according to the type of data that they require as given below [62]:

1. CIPHERTEXT ONLY: The attacker intercepts the ciphertext generated by the encryption algorithm. Here, the attacker will rely on some knowledge of the plaintext source.

2. KNOWN PLAINTEXT: The attacker is able to intercept a set of n ciphertexts $c_0 \dots c_{n-1}$ corresponding to some known plaintexts $p_0 \dots p_{n-1}$ with $c_i = ENC_k(p_i)$ for $0 <= i <= n-1$.

3. CHOSEN PLAINTEXT: The attacker is able to request the encryption of a set of n plaintexts $p_0 \dots p_{n-1}$ of the attackers' choosing and intercepts the ciphertexts $c_i = ENC_k(p_i)$ for $0 <= i <= n-1$.

4. ADAPTIVELY CHOSEN PLAINTEXT: The attacker obtains the encryption of new chosen plaintext p_i for $i >= n$ in an interactive way, perhaps after seeing an original pool of n chosen plaintext/ciphertext pairs $c_i = ENC_k(p_i)$ for $0 <= i <= n-1$.

5. CHOSEN AND ADAPTIVELY CHOSEN CIPHERTEXT: The attacker recovers the decryption of ciphertexts of his/her own choosing. As in the case of chosen plaintext attack, there is also an adaptive version.

6. RELATED KEY: Like in the chosen plaintext attack, the attacker can obtain only the ciphertext encrypted with the help of two keys. These keys are unknown but the relationship between these keys is known, for example, the two keys may differ in only one single bit.

With the increase in the type of data available, the attacker has more control over the analysis of the block cipher and can devise increasingly sophisticated attacks. However, at the same time collecting data of a given type becomes more demanding as we move down

the list. Many of the attacks we differentiated according to how much data is required as well as the type of data required.

The success of different types of cryptanalysis will be measured according to the resources they consume. We have already seen that the amount, and type, of data is important. We can add some additional resources that are likely to be of interest below:

1. TIME: The time, or work effort, required to mount the attack is usually the first thing analysts and commentators consider in a new attack. Sometimes, it is the only requirement considered.

2. MEMORY: The amount of memory, or storage, for an attack is very important. Sometimes the amount of memory is so great that it creates an insurmountable bottleneck and the attack remains impractical. For example, the time to perform, say 2^{40} encryption operations is easier to accumulate than the memory to store 2^{40} results.

3. DATA: We have already seen that the type of data for an attack is important, but so is the amount of data. For instance, an attack might not require much time but it might require an enormous amount of data. If the time required to generate the data far exceeds normal usage patterns then the practical impact of the attack is limited.

The data types and resources listed above are typically the most appropriate way to quantify a cryptanalytic attack. However, when we turn to cryptographic implementation and deployment, a whole range of new attacks that can be more powerful than classical cryptanalysis become crucial. These are attacks that allow the attacker to exploit the leakage of physical information, for instance, the encryption time [68] or the power consumption [69], during the implementation of an algorithm. This area is called *side-channel analysis*, and the design of attacks and countermeasures is a constantly evolving field. However, such works take us out of the scope of this thesis.

So far, we have explained what a block cipher is, what its security relies on and the different types of analysis that can be done on it. Although there are numerous block ciphers, we will only concentrate on several block ciphers which we have analysed in this

thesis, namely, Data Encryption Standard (DES) and RC5. The following sections give brief descriptions about these particular block ciphers.

1.2.1 Data Encryption Standard (DES)

On May 15, 1973, the National Bureau of Standards (now known as the *National Institute of Standards and Technology*, or *NIST*) published a solicitation for cryptosystems in the Federal Register [80]. This led to the ultimate adoption of the Data Encryption Standard,or DES. The Data Encryption Standard (DES) was developed at IBM, as a modification of an earlier system known as *Lucifer*. DES was first published in the Federal Register of March 17, 1975. On January 15, 1977, DES was adopted as a standard for "unclassified" applications. It was initially expected that DES would only be used as a standard for 10 - 15 years; however, it proved to be much more durable. DES then became the world's most widely used encryption algorithm, particularly to protect financial information [28]. DES was reviewed approximately every five years after its adoption. It was last renewed in January 1999 but by this time, the development of its replacement, the *Advanced Encryption Standard* [82, 32, 33] had already begun.

There was a lot of controversy over the use of DES as an encryption standard ever since it was first introduced. There were two main objections over the use of DES:

1. The NSA (National Security Agency) worked with NBS (National Bureau of Standards) throughout the development of DES, evaluated the proposed DES algorithm and recommended several changes to IBM. Specifically, IBM made changes to the S-boxes, the nonlinear substitution transformations that are at the heart of the DES algorithm, to improve its security. Some critics suspected that the NSA deliberately weakened, rather than strengthened, the S-boxes, or perhaps even introduce a "trap door" that would enable the NSA to decrypt messages encrypted by DES.

2. A commonly accepted definition of a good symmetric key algorithm, such as DES, is that there exists no attack better than exhaustive key search in order to obtain the key for decrypting the encrypted message. Critics argued that the 56-bit key used in DES was too short for long term security and that the expected increase in computational power would make the 56-bit key vulnerable to attack by exhaustive key search [36, 50].

As a result of the controversy over DES, research had been mainly focused on cryptanalytic attacks which involved minimal or no exhaustive key search.

1.2.1.1 Description of DES

A detailed description of DES can be found in [80]. DES is a special type of iterated cipher called a *Feistel cipher*. In a Feistel cipher, each state u_i is divided into two halves of equal length, say L_i and R_i. The round function g has the following form:

$$g(L_{i-1}, R_{i-1}, K_i) = (L_i, R_i)$$

where

$$\begin{aligned} &L_i = R_{i-1}, \\ &R_i = L_{i-1} \oplus f(R_{i-1}, K_i) \qquad \text{and} \\ &K_i \quad is \quad the \quad i^{th} \quad round \quad key \end{aligned}$$

DES is a 16-round Feistel cipher having block length of 64 bits. Basically, DES encrypts a plaintext string x of length 64 bits using a 56-bit key, K, to obtain a ciphertext string y of length 64 bits. Prior to the 16 rounds of encryption, there is a fixed *initial permutation* IP that is applied to the plaintext. We denote

$$IP(x) = L_0 R_0$$

After the 16 rounds of encryption, the *inverse permutation* IP^{-1} is applied to the bitstring $R_{16}L_{16}$, yielding the ciphertext y *i.e.*

$$y = IP^{-1}(R_{16}L_{16})$$

Note that L_{16} and R_{16} are swapped before IP^{-1} is applied. The IP and IP^{-1} only permutes the bit positions. The application of IP and IP^{-1} has no cryptographic significance, and is often ignored when the security of DES is discussed. One round of DES encryption is shown in Figure 1.1.

Each L_i and R_i is 32 bits in length. The function f takes as input a 32-bit string (*i.e.* the right half of the current state) and a round key. The 32-bit string is then expanded to 48-bit string by and expansion mapping, E before XORing with the round key K_i

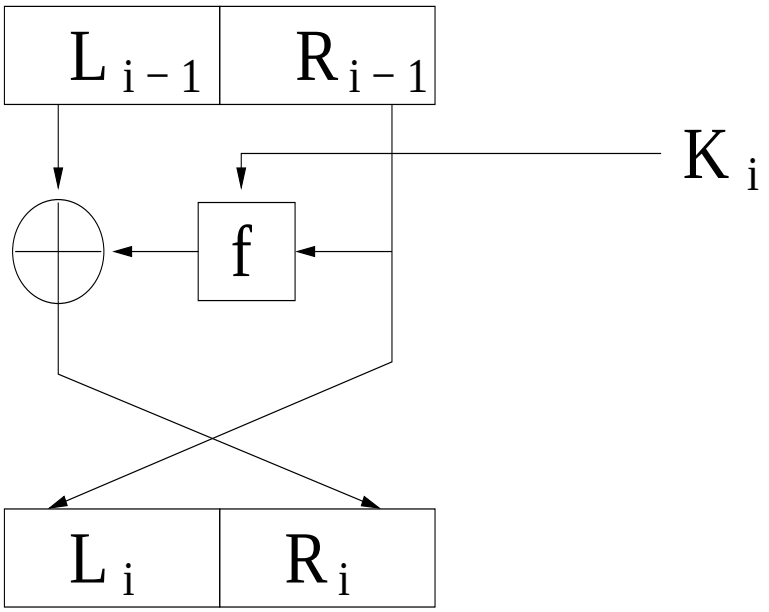

Figure 1.1: One round of DES encryption

according to the key schedule. The key schedule, $(K_1, K_2, \cdots, K_{16})$, consists of 48-bit round keys derived from the 56-bit key, K. Each K_i is a permuted selection of bits from K.

The function f is shown in Figure 1.2. Basically, the function f consists of a substitution (using an S-box) followed by a fixed permutation, denoted by P.

The function f has two inputs, *i.e.* the 32-bit input R_{i-1} and the 48-bit subkey, K_i to produce a 32-bit output which is then XORed with L_{i-1} to produce R_i. In the function f, R_{i-1} is first expanded using and expansion function E thus producing a 48-bit output $E(R_{i-1})$. By bitwise XORing $E(R_{i-1})$ with the subkey, K_i, we then produce a 48-bit output. This 48-bit output is then divided into 8 blocks of 6 bits where each block is fed into an S-box (There are 8 S-boxes altogether, *i.e.* $S1, S2, \cdots, S8$). Each S-box receives 6 bits as input and produces 4-bit output. The S-boxes are lookup tables. Thus, the output of the S-boxes depends on these lookup values. Altogether, the 8 S-boxes produces 32-bit output. Finally, a permutation, P, is applied to this output of the S-boxes (to distinguish it as all 32 bits rather than individual S-boxes) to produce the output for the function f.

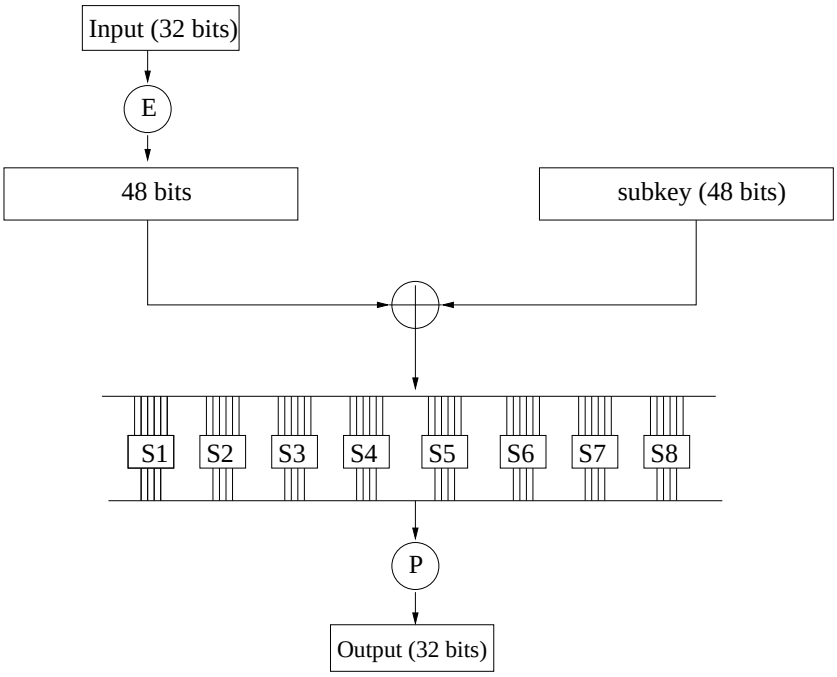

Figure 1.2: The DES f function

1.2.2 RC5

RC5 is a block cipher first introduced by Ron Rivest [84] in 1994. RC5 has been studied extensively for many years ranging from differential cryptanalysis [55, 65] to timing attacks [47]. RC5 is famous for the simplicity in coding of its encryption and decryption routines although, the key schedule is quite complex. RC5 is also the cipher for which the AES candidate RC6 [85] is based upon. A key feature of RC5 is the use of data-dependent rotations.

We will give a formal description about the RC5 block cipher by giving notation that will be used in subsequent Chapters.

1.2.2.1 Notation

To begin with, RC5 is a family of encryption algorithm determined by the parameters in Table 1.1:

The following notations are used for Chapters related to RC5:

$+$ This denotes addition of words modulo 2^w. The inverse operation, denoted by $-$, is

Table 1.1: Parameters for RC5 encryption algorithm

Parameter	Definition	Allowable Values
w	Word size in bits. RC5 encrypts 2-word blocks	16,32,64
r	Number of rounds.	0,1,. . . ,255
b	Number of 8-bit bytes(octets) in the secret key K	0,1,. . . ,255

subtraction modulo 2^w

$\oplus$ This denotes bitwise exclusive-or operation

$\lll$ This denotes the left circular rotation of a word. For example, $X \lll Y$ means the left cyclic rotation of word X by Y^{rot} positions to the left with Y^{rot} denoting the integer represented by the $log_2 w$ least significant bits of word Y. The inverse is the right circular rotation of word X by Y^{rot} positions to the right is denoted by $X \ggg Y$

In [84], a round in RC5 consists of two equations, and in each equation, one of A or B is modified while the other remains unchanged. Each equation is refered to as a half-round. One half-round of RC5 is similar to a full-round in a Feistel cipher. We let L_0 and R_0 represent the left and right half of the plaintext input, each of length w bits. We use the notation L_k and R_k to represent the left and right half, respectively, of the cipher data after (k - 1)-th half-round. Also S_k represents the k-th subkey consisting of w bits associated with (k - 1)-th half-round generated by RC5 key scheduling algorithm. We let L_{2r+1} and R_{2r+1} represent the left and right half of the ciphertext, respectively. We rewrite the RC5 encryption algorithm as follows:

$L_1 = L_0 + S_0$
$R_1 = R_0 + S_1$
for $k = 2$ to $2r + 1$ do
  $L_k = R_{k-1}$
  $R_k = ((L_{k-1} \oplus R_{k-1}) \lll R_{k-1}) + S_k$

For a binary vector x of length w, we label the bit positions from the most significant bit to the least significant bit as $w-1, \ldots, 1, 0$. We use $x[s]$ to denote the s-th bit of x and

$x[s \dots t] \quad (s \geq t)$ to denote the word of length $s-t+1$ consisting of the s-th through t-th bits of x. We use $x[s,t,\dots,u]$ to denote $x[s] \oplus x[t] \oplus \cdots x[u]$. We let $x^{rot} = x[log_2 w - 1 \dots 0]$ which is used to determine a rotation count.

In the following Section, we will give a summary of the development and state-of-the-art in block cipher cryptanalysis.

1.3 Development and State-of-the-art in Block Cipher Cryptanalysis

Over the years, there have been many techniques used in block cipher cryptanalysis. These techniques, or more commonly known as cryptanalytic attacks, are based on the attacker's access such as ciphertext only attack, known plaintext attack or attacker's access to the encryption system to generate plaintexts of chosen ciphertexts. The success of attack can be measured using the number of plaintext-ciphertext pairs or operations required to recover the secret key either fully or partially. When the number of operations required is less than 2^n (*i.e.* exhaustive key search) where n is the number of bits of the secret key, the cipher is said to be broken.

Until recently, work on block cipher cryptanalysis has been concentrated on a number of popular block ciphers such as AES [46, 73], KASUMI [17], Serpent [10, 11, 38, 56], just to name a few. However, a class of block ciphers known as lightweight block ciphers has newly emerged. These are block ciphers used in applications where low power consumption is a requirement along with hardware area constraints regarding their implementation. A survey of the different lightweight block cipher implementations were mentioned in [40] where DES and AES were included and compared with other block ciphers. Our discussion about state-of-the-art in block cipher cryptanalysis will also include work on these block ciphers.

In 1990, Biham and Shamir [4] proposed the basic differential cryptanalytic technique based on DES, which is a chosen plaintext attack. The differential properties of DES's S-boxes were outlined in [76, 77]. Since then, differential cryptanalysis has been applied to various block ciphers such as RC5 [20], HIGHT [34] and PRESENT [23], Many modifications and extensions have been proposed and analysed to improve the attacks on

various cryptographic algorithms. Differential cryptanalysis can be extended to higher order differential cryptanalysis, trunctated differential cryptanalysis and impossible differential cryptanalysis and so on. Although, these are related to the basic idea of differential cryptanalysis, they are different from differential cryptanalysis. Biham [7] outlined that differential cryptanalysis is structurally similar to linear cryptanalysis [74]. Some block ciphers can resist differential cryptanalysis but they do not necessarily resist these types of attacks. Generally, high-order differential cryptanalysis is effective against a cipher having nonlinear modules with low algebraic degree and few iterative rounds. Although, for certain block ciphers, it is hardly possible to find the high probability differential. However, if we are given the characteristics of a few bit differences, some key bits can be restored. This is just the basic idea of truncated differential cryptanalysis. The basic idea of impossible differential cryptanalysis is to crack ciphers by eliminating candidate key points which lead to a small probability check feature (this is generally zero). Lars Knudsen [61] was the one who first proposed using truncated differential in 1994. In the same year, he proposed higher order differential based on the concept of higher order derivatives. Eli Biham, Alex Biryukov and Adi Shamir [9] used impossible differential to break IDEA and Skipjack block ciphers by exploiting differentials that never occurs.

In 1993, linear cryptanalysis was developed by Matsui [74] to exploit linear approximation with high probability, *i.e.* greater than $\frac{1}{2}$. A number of interesting variants of linear cryptanalysis have been proposed including attacks using chosen plaintexts [63], non-linear approximations [66, 88] and multiple linear approximations [52, 53, 22, 27]. Zero correlation is a variant of linear cryptanalysis developed by Andrey Bogdanov and Vincent Rijmen [24] which tries to construct at least one non-trivial linear hull with no linear trail *i.e.* with correlation C exactly zero. This attack is a counterpart of impossible differential attack. To attack a cipher using integral, impossible or zero correlation attack, details of the S-Box used is not required since it is independent of the choice of S-Box used. Choosing another S-Box for a cipher will result in almost the same cryptanalytic results.

After the discovery of linear and differential cryptanalysis, cryptographers started to design ciphers which minimized both the maximum probability of differential characteristics and the maximum correlation of linear characteristics. One of these ciphers was SQUARE, designed by Daemen and Rijmen in 1997. However, during the analysis of a

preliminary version of this block cipher, Knudsen discovered that it was vulnerable to a new type of attack. This forced the designers to increase the number of rounds and the resulting cipher was published in [31], together with the new attack, which was from then on referred to as the "SQUARE attack". The SQUARE attack is not affected by specific design choices for individual components, but relies on how these components, which are considered as black boxes, are interconnected. Another interesting feature of the attack is that it is not probabilistic: if the attacker does not detect the special property which the distinguisher relies on, then he/she knows for sure that the plaintext/ciphertext pairs were not generated by the block cipher. The general technique used in the SQUARE attack has been given different names. Lucks [73] proposed the name *saturation attack*. Biryukov and Shamir [21] treated the technique as a special case of *structural cryptanalysis*, and Knudsen and Wagner [67] refered to it as *integral cryptanalysis*. The idea behind the SQUARE attack is to study the behaviour of complete sets of carefully chosen plaintexts. In order to analyse these sets, the text blocks are first split into m-bit words whose size matches the internal structure of the cipher. The different values taken by each individual word are then treated as multisets. A multiset is a list of values, each of which can appear multiple times, but the order of which is irrelevant. During the attack, a number of special multisets are considered. Multiset attacks are of particular significance today due to their applicability to RIJNDAEL. The RIJNDAEL cipher, designed by Daemen and Rijmen in 1998 [32], is a successor of SQUARE. It was submitted to the U.S National Institute of Standards and Technology (NIST) in response to a open call for 128-bit block cipher. It was, together with 14 other candidates, extensively evaluated during two years, before NIST announced in 2000 that RIJNDAEL would replace DES and become the new AES [82]. RIJNDAEL was specifically designed to resist differential and linear cryptanalysis. Multiset attacks have been shown to be the most effective in breaking reduced versions of RIJNDAEL. The SQUARE attack, which was also applicable to the RIJNDAEL structure, allowed to break six rounds out of ten. It recovered the 128-bit key using a set of 2^{32} special plaintexts, and it required a computational effort of 2^{72} steps. The work factor was later reduced to 2^{44} by performing the calculations in a more efficient way [43]. Ferguson *et al.* [43], as well as Gilbert and Minier [46], have developed more sophisticated multiset attacks that could be applied to seven rounds.

Boomerang attack was developed by Wagner [92] in 1999, which explains that an attack

is possible even if no differentials with high or low probability is present for the whole cipher. It is an adaptive chosen plaintext/ciphertext attack in which the attacker finds two short differentials with high probabilities instead of one whole differential with low probability. Boomerang uses adaptive chosen plaintext/ciphertext for which many of the ciphers that had been developed throughout the years cannot be attacked by boomerang distinguishers and key recovery attack cannot be applied. This led to to the development of its chosen plaintext variant called the Amplified attack [56]. This was later modified by Biham, et.al. [10] and named as Rectangle Attack. In [13], Biham also outlined some results from using boomerang and rectangle attacks.

In 1994, Martin Hellman and Susan K. Langford [70, 71] introduced the differential-linear attack which is a mixture of both linear cryptanalysis and differential cryptanalysis. The attack utilises a differential characteristic over part of the cipher with a probability of 1 (for a few rounds - this probability would be much lower for the whole cipher). The rounds immediately following the differential characteristic have a linear approximation defined, and we expect that for each chosen plaintext pair, the probability of the linear approximation holding for one chosen plaintext but not the other will be lower for the correct key. Hellman and Langford have shown that this attack can recover 10 key bits of an 8-round DES with only 512 chosen plaintexts and an 80% chance of success. The attack was generalised by Eli Biham *et al.* [12] which uses differential characteristics having probability less than 1. Apart from DES, differential-linear cryptanalysis has been applied to FEAL [2], IDEA [25, 26, 18], Serpent [15, 38], Camellia [94], CTC2 [39, 72], SHACAL-2 [89] and even on the stream cipher Phelix [95].

Biham [6] proposed new types of cryptanalytic attacks using related key in 1993. A line of ciphers have been shown to have weaknesses in this attack scenario [57, 58], namely IDEA, GOST, SAFER, Triple-DES, CAST, DES-X, TEA, etc. Widely used ciphers like RC4 [44], KASUMI [17], AES-192 and AES-256 [19] have been broken in this attack scenario. Lightweight block ciphers such as HIGHT [34] and PRESENT [23] are also prone to Related Key attacks. In [64], Knudsen *et al.* outlined possible attacks on the key schedule of iterated ciphers. Related Key attack can be combined with other variants of differential cryptanalysis where knowledge of the difference in keys may allow to attack more rounds as outlined in [16]. A cryptanalytic attack called the Slide Attack can be viewed as a variant of a related key attack, in which a relation of the key with itself is

exploited. Slide attacks are known plaintext or chosen plaintext attacks and thus are more practical than related key attacks since they do not require the attacker to know relations between different encryption keys.

Differential and Linear cryptanalysis or its variants have been applied to almost all block ciphers developed till today. A block cipher which is resistant to one attack can be attacked by its variants or some combinations of the variants.

The simplest approach to cryptanalysing a block cipher is exhaustive key search. The cryptanalyst wishes to find the key k that was used with block cipher E to encrypt some plaintext P to produce ciphertext C where $C = E_k(P)$. Once k is known, the cryptanalyst can find the plaintext by decrypting the ciphertext, $P = E_k^{-1}(C)$. A simple approach to finding k is known as exhaustive search: the cryptanalyst tries decrypting the known ciphertext C with each possible key in turn until the correct key k is found. In 1998, the Electronic Frontier Foundation has actually built a hardware-based DES key search machine called "Deep Crack"[41]. Deep Crack, cost US$200,000 to build, consisted of 1,536 chips each capable of searching through 60 million keys per second, and required on average, 4.5 days to recover a DES key. Recovery of the DES key was possible due to its key size, which is 56 bits. Nowadays, block ciphers such as AES, which has variable key sizes of 128, 192 or 256 bits, it is virtually impossible to mount an effective exhaustive key search using current computing power since it would take billions of years to recover the key [93].

1.4 Contributions

The following is a list of contributions which appear in this thesis:

- We managed to publish a paper titled "Multiple Linear Approximations in Differential-Linear Cryptanalysis of 8-round DES" [78], which outlines the experimental results that we found from applying Multiple Linear Approximations in Differential-Linear Cryptanalysis of 8-round DES.

- We managed to propose a new test for distinguishing a block cipher from a random permutation. This new test is modified from the SAC Test which was outlined in [48] and we called it modSAC Test. Actually, there are 3 types of modSAC

Test, *i.e.* 2-bit modSAC, 4-bit modSAC and 8-bit modSAC Tests. We used our test together with the SAC Test on the block ciphers DES and RC5. We also performed these tests on AES and KASUMI block ciphers. The beauty of these modSAC Tests is that they are fast and easy to implement in software.

1.5 Limitations of Thesis

Initially, when we started our work for this thesis, we planned to use our own implementation of an encryption algorithm to be used in applying various cryptanalysis techniques. We managed to implement our own implementation of the DES encryption algorithm. However, our implementation was much slower than some DES implementations so we used the fastest DES implementation amongst those found in [86] and use it for our cryptanalytic attacks. Important factors that are vital for any successful cryptanalytic attacks are speed and storage resource. These factors are the main reasons restricting us from performing successful attacks on, say, the full round DES encryption algorithm. We concentrated on attacking 8-round DES because we feel that attacking reduced variants of DES would require less computing and storage resources. With current technology, the feasibility of successful attacks increases tremendously compared to a decade ago. We wanted to find out what kinds of cryptanalytic attacks are possible if we have only limited computing and storage resources. Our cryptanalytic attacks are only limited to Differential Cryptanalysis, Linear Cryptanalysis, Differential-Linear Cryptanalysis, Multiple Linear Approximations in Differential-Linear Cryptanalysis, Related Key Cryptanalysis and our own proposed Distinguishing Attack using modified SAC Tests. As we can from Section 1.3, there are a lot of other cryptanalytic methods that we mentioned but did not cover in this thesis. However, the ones that are covered in this thesis like Differential Cryptanalysis, Linear Cryptanalysis, Differential-Linear Cryptanalysis and Related Key Cryptanalysis are popular attacks on block ciphers.

1.6 Organisation of Thesis

The block ciphers we are going to discuss in detail here are the Data Encryption Standard (DES) and the RC5 encryption algorithms. Since we have already given brief explanations

about DES and RC5 encryption algorithms, we will outline our own implementation of the DES encryption algorithm in Chapter 2. In Chapter 3, we discuss how we implemented differential cryptanalysis on 8-round DES. We describe the main idea of differential cryptanalysis and the use of "characteristics" in the differential cryptanalysis of 8-round DES. At the end of the Chapter, we give the result of performing differential cryptanalysis of 8-round DES for different number of chosen plaintexts. In Chapter 4, we discuss about linear cryptanalysis of 8-round DES. In Chapter 5, we explain about differential-linear cryptanalysis of 8-round DES. In Chapter 6, we focus on using multiple linear approximations in differential-linear cryptanalysis of 8-round DES. In this Chapter we are using information from Chapters 3, 4 and 5 to perform this cryptanalysis. In performing the different cryptanalyses in Chapters 3 through 6, we will be able to see how the different methods work and how these methods will be integrated to produce new methods of cryptanalysis. Also, whenever possible, we give our experimental results on performing the different cryptanalyses done on 8-round DES. Chapter 7 discusses about how to distinguish DES from a random permutation. We introduce a well-known test, *i.e.* the SAC Test, for performing the distinguishing attack on n-round DES, for $n = 1 \dots 16$. Also, we introduce our own tests which are modified versions of the SAC Test which we call 2-bit modSAC Test, 4-bit modSAC Test and 8-bit modSAC Test and compare the results from the SAC Tests with those produced from our modified tests. From these results, we produce a Distinguishing Attack Profile for DES. Since we have outlined RC5 as an example in Chapter 1, we discuss the linear cryptanalysis on RC5 in Chapter 8. Chapter 9 focuses on linear weak keys of RC5 and related key cryptanalysis performed on RC5. In Chapter 10, we perform experimental tests on RC5 similar to the tests performed on DES in Chapter 7 in order for us to produce a Distinguishing Attack Profile for RC5 so that we can compare with the Distinguishing Attack Profile for DES. We choose RC5 as the algorithm for comparison with DES because of its versatility in producing different RC5 configurations by changing its parameters. In particular, RC5 was chosen because the input and output paramaters are similar to that of DES. In Chapter 11, we summarise our work for this thesis and give suggestions for future work. We also added the experimental test results from performing our proposed technique to other well known block ciphers, *i.e.* AES and KASUMI.

Chapter 2

DATA ENCRYPTION STANDARD (DES)

In this Chapter we have written our own implementation of DES in software on a PC and we describe the implementation in Sections 2.1 to 2.3. We discuss the implementation issues and the solutions that we used in Section 2.4. Our initial thought of writing our own implementation was to look into the design and practical aspects in the creation of an encryption algorithm. We arbitrarily had chosen DES for our implementation since DES has been extensively analysed through different attacks [8, 4, 5, 74, 75, 70, 30, 42]. After we compared our implementation with several other implementations of DES, specifically some implementations mentioned by Bruce Schneier in [86], we found that our implementation was much slower than existing implementations of DES. Thus, we decided to use the fastest implementation of DES that we compared with, namely, Stuart Levy's DES implementation found in [86]. This implementation of DES will be used throughout our cryptanalytic attacks on DES in this thesis. However, in this Chapter, we will still discuss our own implementation of DES for the benefit of the reader who wants to know about how to do software implementation of DES.

Although, it is well known that by using exhaustive key search, a DES key can be found in less than a day [37], there are still lessons to be learned from studying and cryptanalysing DES and its reduced round variants. In [42], the authors look at the feasibility of using algebraic cryptanalysis on reduced round DES up to 8 rounds. In [30], the authors used algebraic cryptanalysis on the DES's S-boxes for reduced round variants

of DES. This shows that there are still methods or techniques that can be learned from studying DES and its reduced round variants.

We are going to concentrate on four cryptanalytic attacks on DES, namely, differential cryptanalysis, linear cryptanalysis and differential-linear cryptanalysis and multiple linear approximations with differential-linear cryptanalysis. Our emphasis is to give an overview of these different kinds of attacks. Although advancement in computer technology has made DES to be a thing of the past, we will be looking into the feasibility of attacks using a normal computer. We will also look at the feasibility of using parallel programming in attacking reduced round DES, in particular 8-round DES as well as the full 16-round DES.

2.1 Implementation of DES

Before any analysis and/or attack of a particular encryption algorithm is done, it is vital that we first understand how the algorithm works. So, for this reason, we made our own implementation of DES for the purpose of understanding how the algorithm works in order for us to later understand how the different cryptanalytic attacks are done for this particular algorithm. Henceforth, implementing cryptanalytic attacks on block cipher algorithms in practice. In implementing the Data Encryption Standard (DES) we need to understand that we have two tasks that we need to do. They are:

2.1 Key Scheduling

2.2 Encryption or Decryption

We must also understand that in order to achieve both these tasks we would need to define some lookup tables and several functions/subroutines so as to help us in giving an organised view of the whole DES implementation. We try as much as possible to follow the original outline of the DES [80]. The first task is Key Scheduling.

2.2 Key Scheduling

We know that DES must have 56 bits of key data in order to perform the encryption or decryption. In actual implementation, the key is usually supplied in a 64-bit block where 8 of them were not used in the actual encryption or decryption because they are parity

bits. The 56-bit key goes through a key scheduling process. This key scheduling process eventually produces 16 subkeys of 48 bits in length, so that one subkey will be supplied to each round of the DES encryption/decryption.

There are a set of tables that are defined for the Key scheduling algorithm of DES. These are the permutation tables $PC1$ (Figure 2.1) and $PC2$ (Figure 2.2) and the left circular shift table, *lrot* (Figure 2.3).

PC1						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Figure 2.1: The permutation *PC1* in the DES Key Schedule

PC2					
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
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 2.2: The permutation *PC2* in the DES Key Schedule

lrot															
1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Figure 2.3: The left circular rotation *lrot* in the DES Key Schedule

In addition to these tables, we also require some subroutines or functions. We shall now describe these subroutines in detail. They are:

GETBIT This function is used to retrieve a particular bit from an array of bits specified by its position inside the array. The leftmost position in the

array of bits is regarded as position 0.

SETBIT This function is used to set a value either 1 or 0 at a given position in an array of bits.

LROTBIT This function is used to perform left circular rotation of bits on an array of bits of a given size and given number of left shifts.

PERMUTE This function is used to permute bits in an array of bits of a given size. Actually, the way that we permute the bits is by having a table as part of this function's input which indicates where the bits are to be placed. This function is also flexible in that it can also be used as an expansion function, *i.e.* it can also be used to expand, say, a 32-bit array of bits into a 48-bit array of bits.

First, we need to generate the required subkeys is to discard all the parity bits from the 64-bit key block so that we are left with on 56-bit actual key data. Once we have discarded the parity bits we first use the **PERMUTE** function to permute the 56 key bits according to the $PC1$ table. These permuted bits are then separated into two blocks (we refer to them as C_0 and D_0 blocks) of 28-bits. Using the **LROTBIT** function, we then perform left circular shifts on each of the C_0 and D_0 blocks independently to produce C_1 and D_1 blocks, respectively. As mentioned earlier, the number of left shifts needed to be performed here is given by the *lrot* table. Once we have completed these left shifts, we then use the **PERMUTE** function again on the concatenated block C_1D_1 according to the $PC2$ table. This will produce the first subkey K_1. It is important to note that the subkey K_1 is only 48-bits long whereas the concatenated block C_1D_1 is 56-bits long. This means that the **PERMUTE** function in this case, actually "chooses" 48-bits from the 56-bits data. In order to produce the other subkeys, we just need to use the **LROTBIT** function again on each of the blocks C_1 and D_1, independently according to the next number of left shifts given in the *lrot* table to produce C_2 and D_2 blocks respectively. Once again, we perform the permutation on the concatenated blocks C_2D_2 according to the $PC2$ table using the **PERMUTE** function thus producing the subkey K_2. The process repeats until we are able to produce 16 subkeys K_1 to K_{16}.

2.3 Encryption or Decryption

DES is an iterated Feistel cipher so it involves the repetition of a specific round function a certain number of times to achieve either encryption or decryption. Of course, this is not the only task that is being performed during encryption or decryption. Let us first outline what is required for the DES encryption or decryption process.

First of all, the required input for DES encryption or decryption is 64-bit block of plaintext data. We then perform the initial permutation on this block before going through a round function which is repeated for 16 times before finally performing the final permutation on the block to produce the resulting ciphertext data.

Note that here we do not actually differentiate between DES encryption and DES decryption because we know that the process in doing either encryption or decryption in DES is basically the same. The only difference being in the key schedule. If we have the order of the subkeys in the DES key schedule for DES encryption to be $K_1, K_2, \ldots, K_{16}$ then the order of the subkeys in the DES key schedule for DES decryption should be $K_{16}, K_{15}, \ldots, K_1$.

Now, similar to what we have in the previous Section, we also require certain tables for the process of encryption or decryption. These tables are as described below:

IP This is the initial permutation table which is used in performing the initial permutation on the 64-bit plaintext block.

FP This is the final permutation table which is used in performing the final permutation on 64-bit data block before we get the resulting 64-bit ciphertext block.

EX This is the expansion table that is used in DES's f function for expanding a 32-bit data block into a 48-bit data block.

S This is the famous DES's 8 S-boxes required in f function of DES.

P This is another permutation table which performs on 32-bit data and is part of DES's f function.

The tables above are useless without the following subroutines that actually performs the encryption/decryption process. We managed to break up the process involved into hierarchical tasks as outlined below:

SUBSTITUTE This is a subroutine that is specifically used to produce output from DES's S-boxes. Each S-box has 6 bits of input data and 4 bits of output data so since we have 8 S-boxes, we should have 48 bits of input data and 32 bits of output data. This is exactly what the subroutine does, *i.e.* it produces a 32-bit output data from a 48-bit input data according to the S-boxes.

F This is DES's f function. This function incorporates the **SUBSTITUTE** subroutine as well as the **PERMUTE** function described in the previous Section. The **PERMUTE** function is used to "expand" the 32-bit input for the **F** function into a 48-bit input before we XOR with the 48-bit subkey for the particular round. Once XORed, we then input the 48-bit result into the **SUBSTITUTE** function in order to obtain a 32-bit output which is finally permuted again using the **PERMUTE** function with the permutation given by the **P** table. For details of the **F** function, please refer to Figure. 1.2.

ROUND1 This is DES's round function which incorporates the Feistel structure as given in Figure 1.1.

DESEORD This function stands for DES Encryption OR Decryption. We decided to have a function which can either perform DES encryption or decryption depending upon the "flag" that is being set. If the "flag" is 1, then we perform DES encryption, whereas if the "flag" is 0 then we perform DES decryption. This subroutine functions as the DES algorithm where we first perform the initial permutation using the **PERMUTE** function according to the **IP** table. Then we perform the **ROUND1** function iteratively. This is where the difference between DES encryption and decryption comes in. One of the input of the **DESEORD** function is the key schedule (*i.e.* 16 subkeys $K_1, \ldots, K_{16}$). Each subkey is one of the required input to the **F** function in each round of DES. So, the difference between encryption and decryption lies in the order of the subkeys. For encryption, the order of the subkeys is $K_1, \ldots, K_{16}$ whereas the order of the subkeys for decryption is $K_{16}, \ldots, K_1$. Once we had fin-

ished performing the **ROUND1** function, we then finally perform the **PERMUTE** function again using the **FP** table which will result in the ciphertext/plaintext output depending upon whether we are performing DES encryption or decryption. Besides the "flag" input in **DESEORD** we also have a "round" input for which the number of rounds to be performed is specified by this input. So, say, if we want to perform 16-round DES encryption then we set "round" to 16.

Figure 2.4 shows the hierarchy of subroutines used in our implementation. We only show some schematics about what happens in the **F** function but it should be clear that we try to follow the specification from [80] in terms of naming convention and at the same time trying to optimise speed as well. We used the GNU C Compiler Version 4.8.4 for compiling our C source code. We had used the optimisation flag "-o3" when we compiled our source code so that it becomes more efficient by trading off disk space for speed. The full C source code for our DES implementation is given in Appendix A.1 & A.2.

Besides the subroutines specified previously, we also have an additional subroutine **DESEORD1**. This subroutine is basically similar to the **DESEORD** function; the only difference being the initial (**IP**) and final permutations (**FP**) are removed from the **DESEORD1** subroutine. The reason for this is that both the initial and final permutations have no cryptographic significance when we perform either differential or linear cryptanalysis.

2.4 Implementation Issues

The C implementation of DES was done using a desktop computer with a 12-core Intel® Core™ i7 CPU X990 3.47 GHz and 12 Giga Bytes of RAM and 2 Tera Bytes of Disk storage under the Ubuntu 14.04 LTS Linux Operating System. There had been some difficulties in implementing DES in software since the specification in [80] is obviously for hardware implementation. The numbering convention used in [80] is from left to right whereas in software implementation we tend to number from right to left. This can sometimes be confusing. Nevertheless, it is one of the first things that we deal with when we implement any such algorithm using software. This is one of the reasons why we need

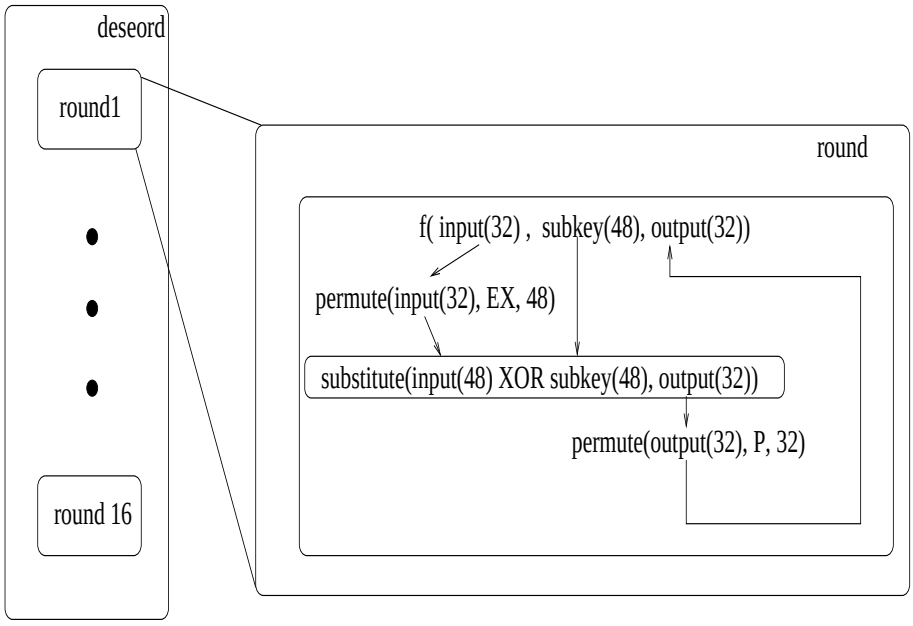

Figure 2.4: Hierarchical View of Subroutines for DES Implementation

careful planning and organization in our implementation so that we can avoid unnecessary problems that may arise.

We had tested our implementation on some DES test vectors and we found that our DES implementation was accurate using the NESSIE test vectors outlined in [79]. We tested our DES implementation and compared the performance with 3 other DES implementations mentioned in [86]. We had chosen only 3 other DES implementations because we had software compatibility issues with the other DES implementations mentioned in [86]. The 3 implementations that we used are Stuart Levy's (Dated April 1988), David A. Barrett's (Dated 4^{th} April 1991) and Phil Karn's (Dated 1987) DES implementations. We had found that the number of encryptions/decryptions that can be done using our implementation for 16-round DES in one second is 9636 encryptions/decryptions. The fastest DES implementation amongst these 4 DES implementations is Stuart Levy's implementation which can do 3792400 encryptions/decryptions in one second. We had used OpenMP [81] API for parallel programming to run these 4 DES implementations on 16777216 plaintexts using only 10 threads. The overall time for all 4 DES implementations to finish is 29 minutes and 28.9 seconds (see Figure A.3.1). Much of the time was due to our own im-

plementation being the slowest DES implementation amongst the 4 implementations.We found that knowledge of some bit manipulation tricks, such as using bit masking to extract certain bits, is useful in order to have a fast implementation of DES. We also found that using memory registers can improve performance of our implementation. However, our implementation was still not as fast the other DES implementations. In our opinion, our DES implementation was not as fast as others in doing encryptions/decryptions was due to much of the time spent in intercommunication between different functions within the DES implementation and lack of optimisation based on the specific hardware that we used.

The reason we made our own implementation of DES was because initially, we wanted to use our own implementation of DES as a test bed for implementing different cryptanalytic attacks such as Differential Cryptanalysis and Linear Cryptanalysis. However, since we found a much faster DES implementation (*i.e.* Stuart Levy's DES implementation), we used this implementation for the cryptanalytic attacks on DES that we discussed in this thesis. Although our own implementation may not be as fast as others, it is still good for any novice in cryptography who wants to learn about implementing an encryption algorithm in C since he/she can see clearly the synthesis of the different components that forms the algorithm.

Next, we will discuss Differential Cryptanalysis of 8-round DES in the following Chapter.

Chapter 3

DIFFERENTIAL CRYPTANALYSIS OF 8-ROUND DES

In this Chapter, we describe Differential Cryptanalysis with particular reference to 8-round DES. We had used Stuart Levy's DES implementation for Differential Cryptanalysis using known differential characteristics. We describe this in Sections 3.4 to 3.6. We have used this implementation to run experiments to test the time and data required for successful cryptanalysis of 8-round DES. The DES implementation that we used is faster in doing differential cryptanalysis than ones used by Biham and Shamir [4, 5]. Our results are summarised in Section 3.6.

Differential cryptanalysis is a method which analyses the effect of particular differences in plaintext pairs on the differences of the resultant ciphertext pairs. These differences can be used to assign probabilities to the possible keys and to locate the most probable keys. This method usually works on many pairs of plaintexts with the same particular difference using only the resultant ciphertext pairs. Differential cryptanalysis, a chosen plaintext attack, was invented in 1990 by Israeli researchers Eli Biham and Adi Shamir [4].

Basically, the differential cryptanalysis method searches for plaintext, ciphertext pairs whose difference is constant, and investigates the differential behaviour of the cryptosystem. For DES, the difference between two elements P_1 and P_2 is defined as $P_1 \oplus P_2$ (bitwise

Table 3.1: One of DES's S-boxes - S1

$S1$															
E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7
0	F	7	4	E	2	D	1	A	6	C	B	9	5	3	8
4	1	E	8	D	6	2	B	F	C	9	7	3	A	5	0
F	C	8	2	4	9	1	7	5	B	3	E	A	0	6	D

XOR operation). We now introduce the following notation:

- P denotes plaintext, T denotes ciphertext
- (P, P^*) is a pair of plaintexts which XOR to a specific value P', *i.e.* $P' = P \oplus P^*$
- (T, T^*) is a pair of ciphertexts which XOR to a specific value T', *i.e.* $T' = T \oplus T^*$
- Primed values are always differential: P', T', a', A', etc. For example, $a' = a \oplus a^*$

A cryptosystem should be a good pseudorandom number generator in order to foil key clustering attacks, that is using the distributions of plaintexts and ciphertexts to deduce information about the key. DES was designed so that all distributions are as uniform as possible. For example, changing 1 bit of the plaintext or the key causes the ciphertext to change in approximately 32 of its 64 bits in a seemingly unpredictable and random manner. Biham and Shamir [4, 5] observed that with a fixed key, the differential behaviour of DES does not exhibit pseudorandomness. If we fix the XOR of two plaintexts P and P^* at P' then T' (which is equal to $T \oplus T^*$) is not uniformly distributed.

3.1 S-box Differential Non-Uniformity

If the input to an S-box is a uniformly distributed random number, its output will be a uniformly distributed random number. Assuming that the 56-bit key is chosen according to a uniform probability distribution, the input to any S-box in any round will be uniformly distributed over all 2^6 (= 64) possible values. The output of any S-box in any round therefore is also uniformly distributed over its 2^4 (= 16) possible values (0 to F in hexadecimal) since each occurs 4 times in each S-box, once in every row (see Table 3.1).

Table 3.2: $S1$ Partial Differential Distribution Table

Input	Output y'															
x'	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
01	0	0	0	6	0	2	4	4	0	10	12	4	10	6	2	4
02	0	0	0	8	0	4	4	4	0	6	8	6	12	6	4	2
03	14	4	2	2	10	6	4	2	6	4	4	0	2	2	2	0
⋮																
0C	0	0	0	8	0	6	6	0	0	6	6	4	6	6	14	2
⋮																
34	0	8	16	6	2	0	0	12	6	0	0	0	0	8	0	6
⋮																
3E	4	8	2	2	2	4	4	14	4	2	0	2	0	8	4	4
3F	4	4	4	2	4	0	2	4	4	2	4	8	8	6	2	2

Consider the differential behaviour of an S-box, in which there are $64^2 = 4096$ possible input pairs (x, x^*). As the 6-bit quantities x, x^*, and $x' = x \oplus x^*$ each vary over their 64 possible values, the 4-bit quantities $y = S(x)$, $y^* = S(x^*)$, and $y' = y \oplus y^* = S(x) \oplus S(x^*)$ each vary over their 16 possible values. (Note that the S function here refers to S-box transformation). The distribution on the differential output y' can be computed for each of the eight S-boxes by counting the number of times each value y' occurs as the pair (x, x^*) varies over its 4096 possible values.

Table 3.2 shows a portion of the differential distribution table for one of DES's S-boxes *i.e.* $S1$. The 6-bit differential input x' takes 64 values (00 to 3F in hexadecimal) while the 4-bit differential output y' takes 16 values (0 to F in hexadecimal). Each row in Table 3.2 sums to 64 because each differential input x' occurs for 64 of the 4096 (x,x^*) pairs. The first row has zeros in all except for the first column because when $x' = x \oplus x^* = 0$, the inputs x and x^* are equal. Therefore, the outputs y and y^* are equal and $y' = y \oplus y^* = 0$. The later rows are more interesting; for example, when $x' = 01$, five of the sixteen possible y' values 0, 1, 2, 4, 8 occur with zero probability (*i.e.* never occurs) while A occurs with probability 16/64 and 9 and C each occur with probability 10/64. This is a highly non-uniform distribution. This differential non-uniformity is observed in all of the S-boxes $S1, S2, \ldots, S8$. This differential non-uniformity may be exploited in order to find the DES key. Let us now look at a particular example which illustrates how the differential

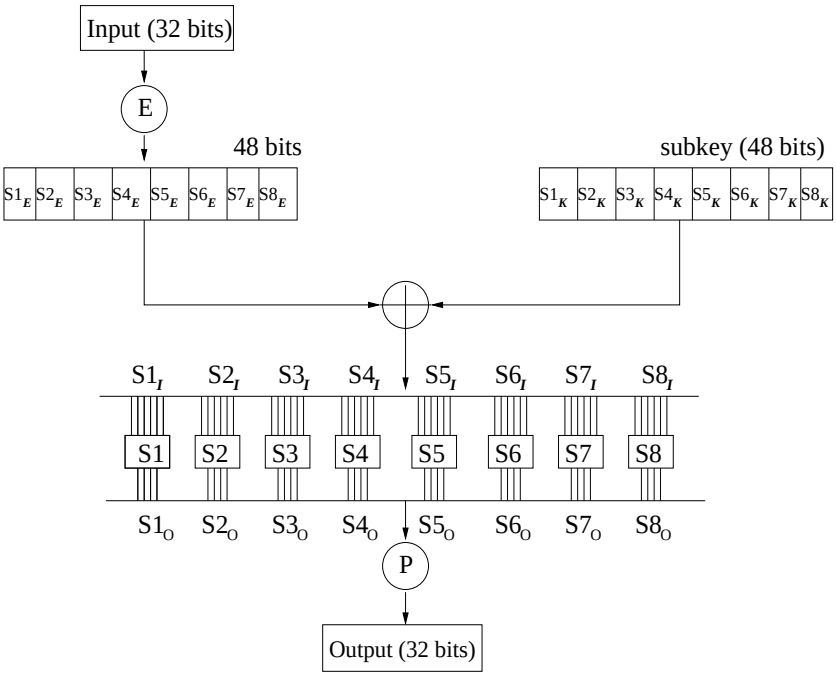

Figure 3.1: The DES f function

pairs are used in determining possible keys for a particular S-box.

3.2 Determination of the key

Figure 3.1 is a slightly modified version of Figure 1.2 with the necessary labels which are used in our discussion. Our example focuses on a particular S-box, *i.e.* $S1$, so Figure 3.2 shows us the parts we are interested in.

Let $S1_E$ and $S1^*_E$ denote two inputs to the S-box $S1$. Let $S1'_E = S1_E \oplus S1^*_E$ and let $S1'_O$ denote the output XOR from the S-box $S1$. Suppose that we know the two inputs to $S1$ to be 01 and 35 respectively and $S1'_O = D$. Thus, $S1'_E = 34$. Then, the input XOR $S1'_I$ is 34 regardless of the value of the key because

$$\begin{aligned} S1'_I &= S1_I \oplus S1^*_I \\ &= (S1_E \oplus S1_K) \oplus (S1^*_E \oplus S1_K) \\ &= S1_E \oplus S1^*_E \\ &= S1'_E \end{aligned}$$

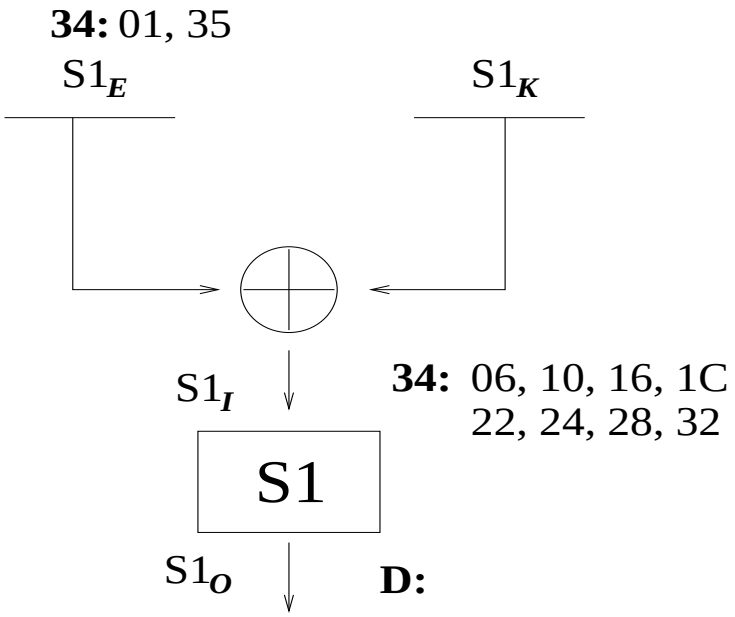

Figure 3.2: The S-box $S1$

Now only eight input pairs $S1_I$ and $S1_I^*$ with XOR $S1'_I = 34$ produce output XOR D. These are $(S1_I, S1_I^*) = (06, 32), \ldots, (32, 06)$. Also since

$$S1_I = S1_E \oplus S1_K$$

we have

$$S1_K = S1_I \oplus S1_E$$

and similarly

$$\begin{aligned}
06 \oplus 01 &= 07, & 06 \oplus 35 &= 33 \\
10 \oplus 01 &= 11, & 10 \oplus 35 &= 25 \\
16 \oplus 01 &= 17, & 16 \oplus 35 &= 23 \\
1C \oplus 01 &= 1D, & 1C \oplus 35 &= 29 \\
22 \oplus 01 &= 23, & 22 \oplus 35 &= 17 \\
24 \oplus 01 &= 25, & 24 \oplus 35 &= 11 \\
28 \oplus 01 &= 29, & 28 \oplus 35 &= 1D \\
32 \oplus 01 &= 33, & 32 \oplus 35 &= 07
\end{aligned}$$

Thus, the possible keys are: {07, 11, 17, 1D, 23, 25, 29,33}. Furthermore, suppose we know

two inputs to $S1$ as 21 and 15 which XORs to 34, and the output XOR as 3. Following similar calculations as above, we have the possible key values to be {00, 14, 17, 20, 23, 34}. So the correct key value must appear in both these sets:

$$\{07, 11, 1D, 23, 25, 29, 33\}$$
$$\{00, 14, 17, 20, 23, 34\}$$

Intersecting these two sets, we obtain 17, 23. Thus, the key value is either 17 or 23. In order to determine which one of these is the correct value, we would require more input/output XORs.

The above is just an example to demonstrate how we can get the required key using a particular S-box. We now introduce a concept which is practical for cryptanalysis of DES.

3.3 Characteristic

A *Characteristic* is the differential input which can be traced through several rounds. A *Characteristic* usually has some probability attached to it. Two observations that we need to consider when dealing with *Characteristics*:

1. The XOR of pairs is linear in the expansion E in DES's f function $i.e.$

$$E(X) \oplus E(X^*) = E(X \oplus X^*) = E(X')$$

2. The XOR of pairs is independent of the key $i.e.$

$$\begin{aligned} S_I &= S_E \oplus S_K \\ S_I^* &= S_E^* \oplus S_K \\ S_I \oplus S_I^* &= S_E \oplus S_K \oplus S_E^* \oplus S_K \\ S_I' &= S_E \oplus S_E^* \\ S_I' &= S_E' \end{aligned}$$

It is much easier to explain about *characteristics* by giving an example. For example, we shall consider what is called a *2-Round Characteristic*. Figure 3.3 shows this *Characteristic*. As can be seen from Figure 3.3, the differential input to function f in the first

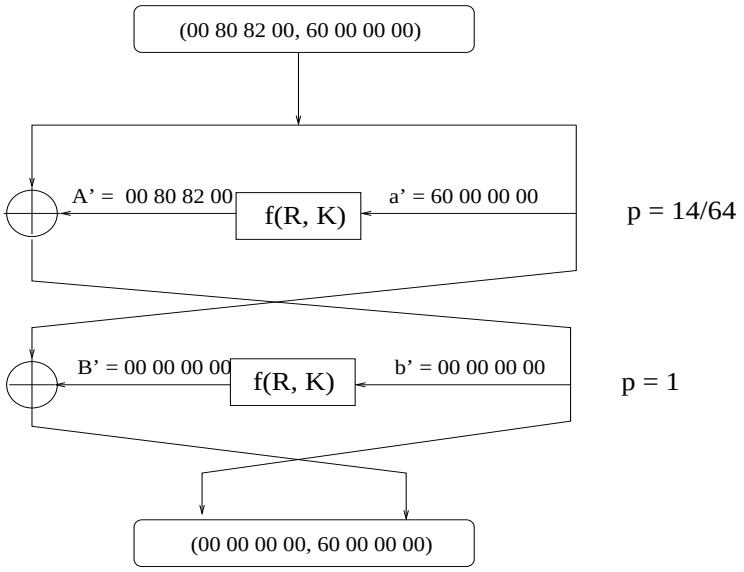

Figure 3.3: 2-Round Characteristic

round is $a' = 60\,00\,00\,00$. The expansion E operation puts these half bytes into the middle four bits of each S-box in order, *i.e.* $6 = 0110$ goes to $S1$ and $0 = 0000$ goes to $S2, \ldots, S8$. Since all the edge bits are zero, $S1$ is the only S-box receiving non-zero differential input. $S1$'s differential input is 0 0110 0 = 0C while the differential inputs of $S2, \ldots, S8$ are all zero. Looking at $S1$'s differential distribution table (see Table 3.2), we find that when $x' = 0C$, the highest probability differential output y' is $E = 1110$, which occurs with probability 14/64. All the other S-boxes have $x' = 0$ and $y' = 0$ with probability 1. The S-box outputs go through the permutation P before becoming the output for $f(R, K)$. As shown in Figure 3.3, the differential output of $f(R, K)$ is

$$A' = P(E0\,00\,00\,00) = 00\,80\,82\,00$$

$A' = 00\,80\,82\,00$ is then XORed with $L' = 00\,80\,82\,00$ to give $00\,00\,00\,00$. Thus, in the second round all S-boxes receive their differential inputs as zero, producing the differential outputs as zero. Finally, the output of $f(R, K)$ in the second round is zero, giving the differential output as depicted: $(00\,00\,00\,00, 60\,00\,00\,00)$

With the brief introduction about *2-Round Characteristic*, let us now move to a topic for which we can use the *2-Round Characteristic* in a successful attack.

3.4 Implementation of Differential Cryptanalysis on 2-Round DES

This implementation assumes that the initial (IP) and inverse (IP^{-1}) permutations are removed from the DES algorithm since they are of no cryptographic significance. Below is the algorithm used to attack 2-Round DES:

Step 1: Generate a plaintext pair (P, P^*) such that

$$P' = P \oplus P^* = 00\,80\,82\,00\,60\,00\,00\,00$$

This is done by generating a random P and XORing with $00\,80\,82\,00\,60\,00\,00\,00$ to generate P^*.

Step 2: Given the plaintext pair (P, P^*), generate the ciphertext pairs (T, T^*)

Step 3: Compute $T' = T \oplus T^*$ and see whether it is equal to $00\,00\,00\,00\,60\,00\,00\,00$. If it is not, the *characteristic* has not occurred and this pair is not used. Then, go to Step 1 and generate a new plaintext pair. If T' is equal to $00\,00\,00\,00\,60\,00\,00\,00$ then the *characteristic* has occurred, and we know the values of A' and B' (refer to Figure 3.3). Go to Step 4.

Step 4: Since $S2, \ldots, S8$ have their differential inputs equal to zero, no information can be gained about $S2_K, \ldots, S8_K$. In the differential distribution table of $S1$ (Table 3.2), we have $0C \rightarrow E$ with probability 14/64, only 14 of 64 possible $S1_K$ values allow

$$a' = 60\,00\,00\,00$$

to produce

$$A' = 00\,80\,82\,00$$

These 14 allowable values can be determined by XORing each possible $S1_K$ with the corresponding six bits of $S1_E$ and $S1^*_E$, computing $S1's$ differential output $S1'_O$ and checking if it is equal to E. Put these 14 values of $S1_K$ in a table

Step 5: Compute the intersection of these tables. Since the correct key value must occur

in each table, it will be in the intersection. If more than one $S1_K$ value results, this means we do not have enough plaintext, ciphertext differential pairs to uniquely determine $S1_K$. Go to Step 1 and generate additional data. The number of plaintext, ciphertext differential pairs needed is approximately equal to the inverse of the probability of the *characteristic* used; in this case $64/14 \approx 5$ pairs are needed.

Step 6: At this point we have recovered the 6 bits of the key comprising $S1_K$. Use similar *characteristics* to recover the 6 bits of key which are XORed with $S2$ through $S8$'s inputs in the first round.

Step 7: At this point we have 48 bits of the key which comprise S_K, or equivalently $S1_K$ through $S8_K$. Find the remaining 8 bits of K by exhaustive search over the 64 possible values.

If we remove Step 3 from the algorithm above and assume that the characteristic occurs for every pair of encipherment, the probability of this *characteristic* to occur is 14/64. Thus, we will be wrong for 50/64 of the pairs. The correct value of $S1_K$ need not occur in every table and we should look for the most frequent $S1_K$ value. The correct value occurs in 14/64 of the tables and the remaining 63 values occurs with approximately equal (or smaller) probability.

Apart from the above *characteristic*, Biham and Shamir [4] have also developed some specialised *characteristics* to be used in attacking specific reduced variants of DES. One such method is known as the 3R Attack.

3.5 3R Attack

The 3R Attack [4, 5] is a method in which we can use an n-round *characteristic* to break an $(n+3)$-round DES. Figure 3.4 shows the *1-Round Characteristic* used in the 3R Attack. This *1-Round Characteristic* is used to break 4-Round DES. Figure 3.5 shows how the *1-Round Characteristic* goes through the four rounds of DES for which we can trace the output in every round.

In Figure 3.5, we know the inputs x and x' for all S-boxes in the last round since d and d' are left halves of the ciphertext T and T'. To recover the 6-bit subkeys Si_K in the last round, we need to learn the differential outputs y' from some S-boxes. We know the values

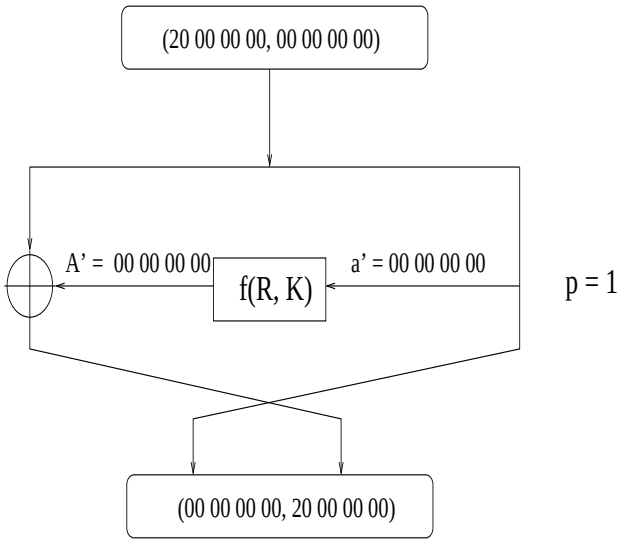

Figure 3.4: 1-Round Characteristic

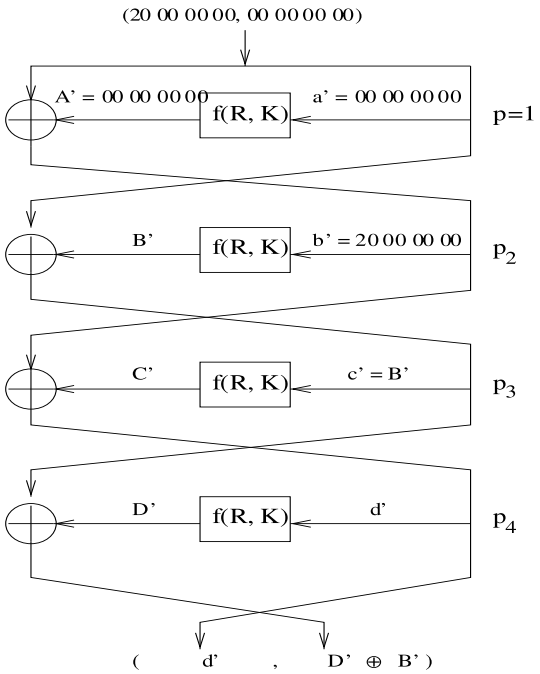

Figure 3.5: Differential Cryptanalysis Implementation on 4-Round DES

of d' and $D' \oplus B'$ since we know the ciphertext pair causing this *characteristic*. In order to get D', we need to obtain B'. Note that $b' = 20\,00\,00\,00$ causes the differential inputs to $S2$ through $S8$ in the second round to be all zeroes in their middle four bits. $2 = 0010$ is input to $S1$'s middle four bits and the seven zeroes are inputs to $S2, S3, \ldots, S8$. Thus B' have 28 zeroes in its representation; the places of the zeroes can be found by tracing back the permutation used at the output of $f(R, K)$. Therefore, we know 28 bits of D'. We can now obtain 42 bits of the key using our analysis technique; the remaining 14 bits can be obtained using exhaustive key search.

We now move to our main topic of discussion *i.e.* the differential cryptanalysis of 8-round DES. Biham and Shamir [4] claimed that we would need 150,000 differential pairs for a successful attack.

3.6 Implementation of Differential Cryptanalysis on 8-Round DES

Now, for the differential cryptanalysis of 8-Round DES, we need a *5-Round Characteristic* in order to have a successful attack [4, 5]. Figure 3.6 gives a schematic diagram of 8-Round DES when the *5-Round Characteristic* is being utilised. The probability of the *5-Round Characteristic* is

$$\frac{16}{64} \cdot \frac{10 \cdot 16}{64^2} \cdot \frac{16}{64} \cdot \frac{10 \cdot 16}{64^2} \approx 9.5 \times 10^{-5}$$

Thus, we expect approximately 10,000 (P,P^*) pairs are needed per occurrence of the *characteristic*. Since we cannot completely observe input and output XORs, we cannot guarantee that the characteristic had occurred. So, we assume that the *characteristic* occurs for every differential pair of plaintext (P,P^*). Statistically, we are correct in choosing the right pair only one in 10,000 thus we need several times this number of differential pairs for a successful attack. Here is an algorithm used to attack 8-Round DES using differential cryptanalysis (Note: this attack assumes that the initial IP and inverse IP^{-1} permutations are removed from the DES algorithm):

Step 1: Generate the pair (P, P^*) whose differential $P' = P \oplus P^*$ is equal to 40 5C 00 00 04 00 00 00

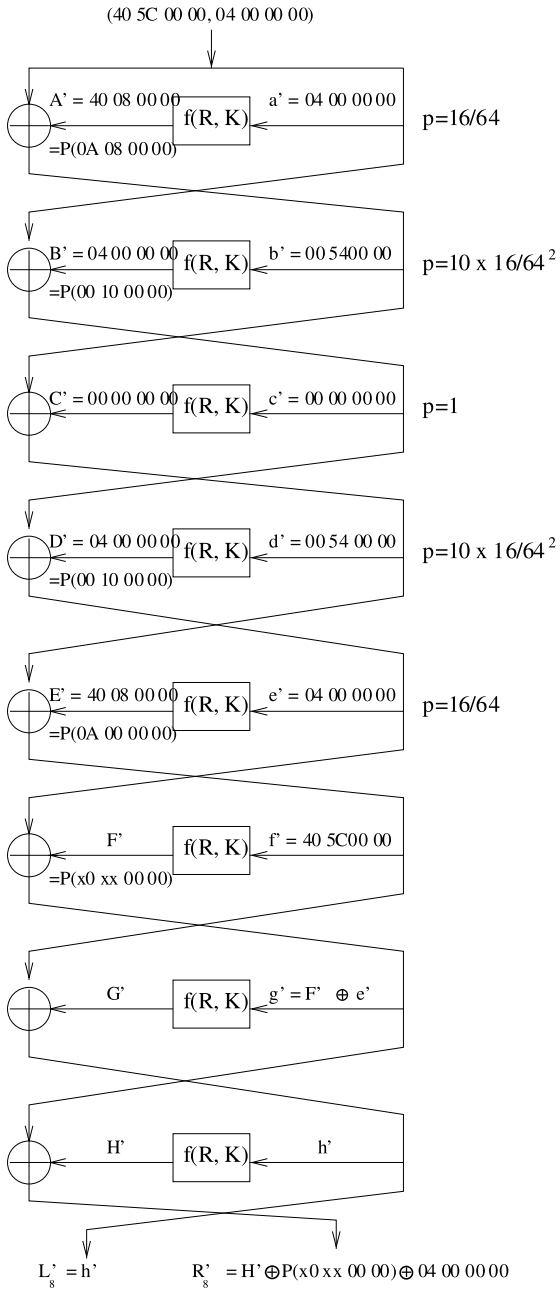

Figure 3.6: 8-Round DES using 5-Round Characteristic

Step 2: Obtain the ciphertext pair (T, T^*)

Step 3: Assume the *characteristic* has occurred and compute the differential outputs of $S2$, $S5$, $S6$, $S7$ and $S8$ in the eighth round using $P^{-1}(H')$ which is equal to $P^{-1}(R'_8 \oplus 04\,00\,00\,00) \oplus (x0\,xx\,00\,00)$

Step 4: Test each of the 64 possible 6-bit subkeys $K_{8,2}$ associated with $S2$ in the eighth round to see which case the observed (x,x^*) to produce y' computed for $S2$ in Step 3. Put those subkeys that produce y' in a table $\{K_{8,2}\}$. Repeat this step to produce tables of possible subkeys $\{K_{8,5}\}$, $\{K_{8,6}\}$, $\{K_{8,7}\}$ and $\{K_{8,8}\}$ for $S5$, $S6$, $S7$ and $S8$ respectively.

Step 5: If any of the five tables produced in Step 4 is empty, the *characteristic* could not have occurred. In this case, discard all 5 tables, return to Step 1 and try a new differential plaintext pair.

Step 6: If each of the 5 tables is non-empty, the characteristic may have occurred. In this case, generate all possible 30-bit portions of K_8 associated with $S2$, $S5$, $S6$, $S7$, and $S8$ by choosing one 6-bit value from each table. If n_2, n_5, n_6, n_7 and n_8 denote the number of 6-bit subkeys in the tables $\{K_{8,2}\}$, $\{K_{8,5}\}$, $\{K_{8,6}\}$, $\{K_{8,7}\}$ and $\{K_{8,8}\}$, then the number of 30-bit values is $N = n_2n_5n_6n_7n_8$. Let $\mathcal{K}$ denote such a 30-bit value.

Step 7: For each of the $\mathcal{K}$'s generated in Step 6, increment a counter corresponding to that value. Since a fixed number plaintext pairs are used in the cryptanalysis, the counter having the maximum value would should give the correct subkey value.

It is claimed by Biham and Shamir that 8-round DES can be broken in less than 2 minutes on a personal computer by analysing about 2^{14} ciphertexts [5]. We had actually implemented the steps above in performing Differential Cryptanalysis of 8-round DES. We summarised the experimental results of our findings at the end of this Chapter.

The steps outlined above are used in recovering 30 bits of key data. In order to recover all 56-bit DES key (other than using exhaustive key search alone), we first need to recover some bits using Differential Cryptanalysis or some other methods and then recover the remaining bits using exhaustive key search. So, if we recover only 30 bits of key data, then

we would need to find the other 26 bits using exhaustive key search. This is doable on our desktop PC but we can still recover additional bits in order to perform exhaustive key search which is faster to do because the search space for the key is smaller. Biham and Shamir[4] had provided some additional steps to recover additional key bits, *i.e.* 18 extra bits. This means that we can perform exhaustive key search to find just 8 extra bits.

According to [4], once we had obtained the 30 key bits (subkey bits from K_8) using the previous steps, we can calculate the values of 20 bits of H and H^* (refer Figure 3.6) for each pair and thus 20 bits of g and g^* (by $g' = H' \oplus R'$). There is a dependence of the g bits and the subkey bits of K_7 at the seventh round on the known and unknown subkey bits of K_8 at the eighth round. The expected value of G' is known by the formula $G' = f' \oplus h'$. We can now look for the 18 missing bits of K_8 by exhaustive key search of 2^{18} possibilities for every pair. Thus we know H, H^* and g, g^* and 40 bits of K_7. For each pair, we check that the expected value of G' holds. For the correct value of those 18 key bits the expected G' holds for almost all the right pairs. This is how we recover the extra 18 bits of K_8. Once we recovered these bits, we can perform exhaustive key search on the other 8 key bits.

We had decided to follow the method outlined by D.R. Stinson in [90] for our practical implementation of Differential Cryptanalysis of DES. The method employed in [90] is similar to the outlined steps above where we search possible key bits for the S boxes S_2, S_5, S_6, S_7 and S_8.

We decided to employ the method used in [4] to recover all 48 bits of K_8. Basically, the method in [4] is the outlined steps above to recover 30 bits from 5 S boxes and using these recovered bits, we then try to recover an extra 18 bits from the other 3 S boxes.

From our experimental result based on the steps outlined in [4, 5], we discovered that we can break 8-round DES in less than 1 second with reasonable percentage of success. Tables 3.3 shows the experimental result given different number of chosen plaintexts. The result is based on 100 trials for each different number of chosen plaintexts used and we give the percentage of successful recovery of 56 key bits. We had used MD5 [83] hash algorithm to generate the plaintexts used for our experiment.

As we can see from table 3.3, we can break 8-round DES in less than 1 second which is faster than the one done by Biham and Shamir in [4, 5]. However, Biham and Shamir's

estimate of using 150000 chosen plaintext pairs for 100% successful recovery of key is lower than our estimated 220000 chosen plaintext pair for 100% successful key recovery. In our opinion, this may be due to the difference in the generator that was used in producing plaintext pairs. Figures F.1.1 - F.1.10 in Appendix F of this thesis, give snapshots of the number of successful attacks and the time taken for Differential Cryptanalysis of 8-round DES with the specific amount of chosen plaintext pairs as given in Table 3.3.

Table 3.3: Percentage of Successful Key Recovery

Number of Chosen Plaintexts	40000	60000	80000	100000	120000	140000	160000	180000	200000	220000
Percentage of Success	13%	37%	62%	73%	83%	90%	95%	99%	99%	100%
Average time taken	0.24s	0.35s	0.46s	0.57s	0.69s	0.80s	0.91s	1.03s	1.14s	1.25s

In the next Chapter, we will discuss about Linear Cryptanalysis of 8-round DES.

Chapter 4

LINEAR CRYPTANALYSIS OF 8-ROUND DES

In this Chapter we describe Linear Cryptanalysis. We have successfully implemented linear cryptanalysis of 8-round DES. Linear cryptanalysis does not recover the 56 bits of key. It only provides a key ranking of possible keys. Increase in the number of known plaintexts used for linear cryptanalysis will "promote" the correct key bits to the top rank in the key ranking list. Once we have the suggested key bits, we then use exhaustive key search to obtain the rest of the key bits. We used parallel programming to implement the exhaustive key search to obtain the rest of the key bis. We discuss implementation issues and the results of our experiments in Section 4.5.

Linear cryptanalysis is a known plaintext attack and was introduced by Matsui [74, 75]. The principle behind linear cryptanalysis is to find linear approximations to the non-linear S boxes, then use these linear expressions to construct a linear path through the Feistel structure of DES, so that the result will be a linear expression for DES that relates the plaintext, ciphertext and key.

In linear cryptanalysis, we omit the initial (**IP**) and final (**FP**) permutations of the DES algorithm because they have no cryptographic significance whatsoever. We employ similar notations used in [74, 75] as follows:

- P : The 64-bit plaintext
- C : The corresponding 64-bit ciphertext

- P_L : The left 32-bit of P
- P_R : The right 32-bit of P
- C_L : The left 32-bit of C
- C_R : The right 32-bit of C
- X_i : The 32-bit intermediate value in the i-th round f function
- K_i : The 48-bit subkey in the i-th round
- $F(X_i, K_i)$: The i-th round f function
- $A[i]$: The i-th bit of A
- $A[i, j \ldots, k]$: $A[i] \oplus A[j] \oplus, \ldots, \oplus A[k]$

Note that, the right most bit of each symbol above is refered to as the zero-th (lowest) bit. The first thing that we need to do for linear cryptanalysis is to find a suitable linear approximation to the DES algorithm before applying the known plaintext attack algorithm. The following is an outline of how we are able to obtain linear approximations to the DES algorithm:

1. Find linear approximations to the S-boxes that relates a subset of the S-box inputs to a subset of its outputs. We find the probability of each approximation holding true.
2. Extend the approximations to DES's f function, and thus formulate linear expression for each approximation.
3. Construct a linear path, which is an approximation of the DES algorithm, by compounding linear expressions for DES's f function. We can calculate the probability for the DES approximation from the probabilities of the S box approximations.
4. Calculate the number of plaintexts required to obtain the value of the DES linear approximation expression. The number of plaintexts depends on the probability that the DES approximation holds, as well as our required degree of success.

We will discuss these steps in greater detail next.

4.1 Linear Approximation of S boxes

The S-boxes are the only nonlinear elements of DES. Therefore, good linear approximations for the S-boxes can extend to linear approximations for the DES algorithm as a whole. The measure of S-box linearity which Matsui uses is the number of times the XOR of certain input bits to an S-box is equal to the XOR of certain output bits. We can define this measure more formally as given in [74]

Definition 4.1.1. For given S-box S_a (a= 1,2,...,8), $1 \leq \alpha \leq 63$ and $1 \leq \beta \leq 15$, we define $NS_a(\alpha, \beta)$ as the number of times out of 64 input patterns of S_a, such that an XORed value of the input bits masked by α coincides with an XORed vlaue of the output bits masked by β; that is to say,

$$NS_a(\alpha,\beta) \stackrel{\text{def}}{=} \#\left\{x|\ 0 \leq x < 64, (\bigoplus_{s=0}^{5}(x[s] \bullet \alpha[s])) = (\bigoplus_{t=0}^{3}(S_a(x)[t] \bullet \beta[t]))\right\}, \qquad (4.1.1)$$

where the symbol # denotes the number of input patterns and the symbol $\bullet$ denotes a bitwise AND operation.

When $NS_a(\alpha, \beta)$ is not equal to 32, we may say that there is a correlation between the input and the output bits of S_a. The probability that the linear approximation expression associated with the pair (α, β) holds is $NS_n(\alpha, \beta)/64$. For example, $NS_6(16, 7) = 18$, therefore the probability that input bit 4 (value 16, "001000" in 6-bit binary) coincides with the output bits 0, 1 and 2 (value 7, "0111" in 4-bit binary) is $18/64 = 0.28125$.

Testing all S-boxes with all values of α and β reveals the best approximation (*i.e.* the one where $|NS_n(\alpha, \beta) - 32|$ is maximal) comes from $NS_5(16, 15) = 12$ with probability of occurrence of $12/64 = 0.1875$.

We can now derive a linear expression representing one round of the DES algorithm(*i.e.* a linear expression for DES's f function). To find the linear expression we note that the input bit(s) from the subkey must be in the same position as the input bit(s) of the data block after the expansion E in the f function, and that the output bit(s) are permuted through the P permutation of the f function.

For $NS_5(16, 15)$, we obtain the one-round linear approximation:

$$X[15] \oplus F(X, K)[7, 18, 24, 29] = K[22] \qquad (4.1.2)$$

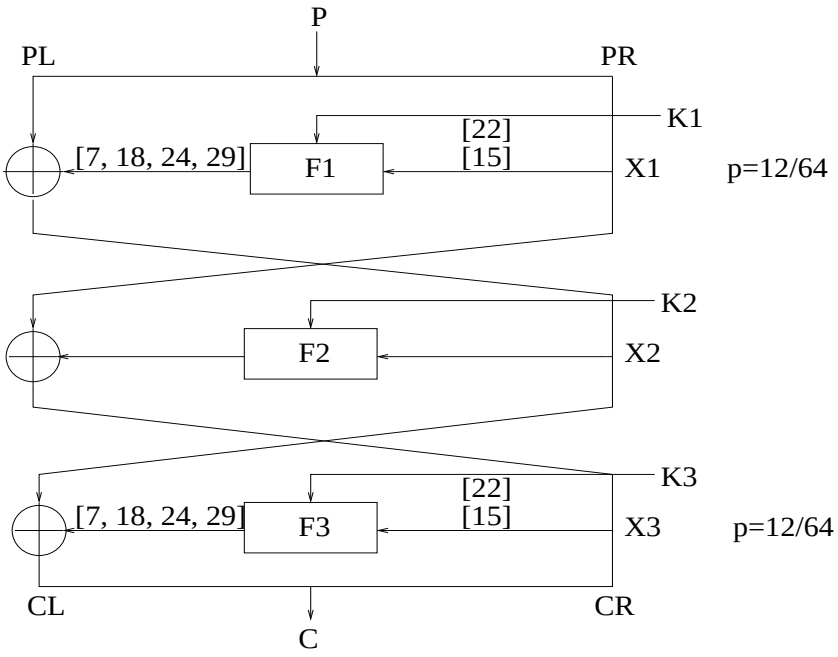

Figure 4.1: 3-round DES cipher

where X is the input block, K is the round subkey, and $F(X, K)$ is the output data block from DES's f function. This one round expression holds with the same probability as the linear approximation from which it was derived, *i.e.* equation (4.1.2) holds with probability 0.1875 for random X and fixed subkey K.

4.2 n-round Linear Approximations

In the previous Section, we were able to obtain a one-round linear approximation. In order to extend the linear approximation to more than one round, we need to compound such linear expressions. Let us look at a particular example. Figure 4.1 shows a 3-round DES with the appropriate approximations and the respective probabilities being displayed in the figure. We want to find a linear approximation to the 3-round DES cipher.

As we can see, in the first round, we have P_R, the right half of the plaintext P enters DES's f function so we have the following expression:

$$F(P_R, K_1)[7, 18, 24, 29] \equiv P_L[7, 18, 24, 29] \oplus X_2[7, 18, 24, 29], \tag{4.2.1}$$

where X_2 is the data block as it enters DES's f function in the second round. Substituting equation (4.2.1) into equation (4.1.2), we obtain the linear approximation to the first round:

$$P_R[15] \oplus P_L[7, 18, 24, 29] \oplus X_2[7, 18, 24, 29] = K_1[22]. \tag{4.2.2}$$

Similarly, we apply the linear approximation (4.1.2) to the final round:

$$C_R[15] \oplus C_L[7, 18, 24, 29] \oplus X_2[7, 18, 24, 29] = K_3[22], \tag{4.2.3}$$

where C_R and C_L are the right and left halves of the ciphertext C respectively. If we add equations (4.2.2) and (4.2.3), we will be able to cancel the unknown X_2 term giving a linear approximation to 3-round DES cipher:

$$P_R[15] \oplus P_L[7, 18, 24, 29] \oplus C_R[15] \oplus C_L[7, 18, 24, 29] = K_1[22] \oplus K_3[22]. \tag{4.2.4}$$

The probability that equation (4.2.4) holds for random plaintext P and its corresponding ciphertext C is $(12/64)^2 + (1 - 12/64)^2 = 0.6953$. This is because it will hold *if and only if* either both one-round approximations hold, or if neither hold. The probability can also be obtained using equation (4.2.5) given by Piling Up Lemma (Lemma 4.2.1) by Matsui [74]

Lemma 4.2.1 (Piling Up Lemma)**.** *Let X_i $(1 \leq i \leq n)$ be independent random variables whose values are 0 with probability p_i or 1 with probability $1 - p_i$. Then the probability that $X_1 \oplus X_2 \oplus \ldots \oplus X_n = 0$ is*

$$1/2 + 2^{n-1} \prod_{i=1}^{n} (p_i - 1/2). \tag{4.2.5}$$

Matsui also estimated the success rate of the algorithm and the number of plaintexts required to achieve a particular degree of success. The success rate is based on the maximum likelihood method and is derived by approximating binary distribution with normal distribution. Table 4.1 shows the relationship between the number of plaintext N and the

success rate.

Table 4.1: Success rate of linear cryptanalysis (*Algorithm* 4.2.1)

N	$\frac{1}{4}\|p-1/2\|^{-2}$	$\frac{1}{2}\|p-1/2\|^{-2}$	$\|p-1/2\|^{-2}$	$2\|p-1/2\|^{-2}$
Success rate	84.1%	92.1%	97.7%	99.8%

We need $|p-1/2|^{-2}$ plaintexts to achieve a success rate of 97.7%, *i.e.* $N = |0.6953 - 1/2|^{-2} \approx 27$ plaintexts. We use algorithm 4.2.1 to recover the value of the linear approximation (equation (4.2.4)) and hence, obtain a linear expression for a key bit.

Algorithm 4.2.1 (Linear Cryptanalysis of 1 key bit).

1. Initialize counter $T = 0$
2. For each plaintext P_i and the corresponding ciphertext C_i, where $0 \leq i < N$, do the following:
   - Let t be the evaluation of the left hand side of the DES linear approximation, for example, evaluate the left hand side of equation (4.2.4):
   $$t \leftarrow P_R[15] \oplus P_L[7, 18, 24, 29] \oplus C_R[15] \oplus C_L[7, 18, 24, 29].$$
   - If $t = 0$, increment counter T
3. If the probability of the DES linear approximation is greater than $1/2$ (for example, $p = 0.6953$):
   - If $T > N/2$ then guess that the value of the right hand side of the DES linear approximation is 0, for example, the right hand side of equation (4.2.4): $K_1[22] \oplus K_3[22] = 0$
   - If $T \leq N/2$ then guess that the value of the right hand side of the DES linear approximation is 1, for example, the right hand side of equation (4.2.4): $K_1[22] \oplus K_3[22] = 1$
4. If the probability of DES linear approximation is less than $1/2$:
   - If $T > N/2$ then guess that the value of the right hand side of the DES linear approximation is 1

Table 4.2: Percentage of Success for obtaining 1 key bit using *Algorithm* 4.2.1

N	6	13	26	52
Percentage of Success	82.8%	93.5%	97.1%	99.3%
Average time taken	0.012s	0.016s	0.022s	0.034s

- If $T \leq N/2$ then guess that the value of the right hand side of the DES linear approximation is 0

The success rate of the above algorithm increases when N or $|p - 1/2|$ increases. The value of the linear equation for the key bit would be used in a reduced exhaustive key search algorithm. Table 4.2 shows the experimental result of obtaining one key bit on 3-round DES using 1000 trials with 1000 random keys.

As can be seen from Table 4.2, the experimental result roughly equals the theoretical result of Table 4.1. Having recovered a linear expression for one key bit, we would have only 55 key bits to search. For linear expressions for a larger number of rounds (except for 5- and 11-round DES), in order to obtain the best ("best" here means having optimal probability), we would build the DES linear approximation using different S-box approximations (in our 3-round DES example, we used the same approximation for the first and final rounds, thus making the construction symmetrical). Matsui gives a table of the best linear approximation expressions for DES up to 20 rounds which was found by computer search [74].

We can use the symmetry of DES to obtain another linear expression for a key bit, with little additional overhead. In instances where the n-round DES linear approximation construction is not symmetrical (*i.e.* we do not use an S-box approximation in round $(n - i + 1)$ if it is used in the ith round) we can run algorithm 4.2.1 again, swapping the roles of the plaintext and ciphertext. The result would be the value of the "reverse" key linear equation, which is similar to the original expression, except that all subkeys K_i are replaced with K_{n-i+1}. For fast implementation, we do not actually run the algorithm twice; it would be more efficient to use two counters T and U, one for each expression, in the data analysis phase of the algorithm.

The next Section describes another approach in using linear approximations.

4.3 $(n-1)$-round Linear Approximations

Previously, we had a basic algorithm to recover one key bit using a linear approximation. An improvement on this basic algorithm, *i.e.* recovering more than one key bit, would be to use a linear expression that holds for n-rounds with the probability of an $(n-1)$-round approximation. The only way to construct such an expression is to apply the DES linear approximation expression to the first $n-1$ rounds only, and leave the final round unapproximated. The final (non-linear) round will be part of the resulting expression, but we will show how to deal with this with an example for 8-round DES.

From [74], the best linear expression for 8-round DES, using the best 7-round linear approximation (holding with probability $0.5 + 1.95 \times 2^{-10}$) is:

$$\begin{aligned} P_L[7,18,24] \oplus P_R[12,16] \oplus C_L[15] \oplus C_R[7,18,24,29] \oplus F(C_R, K_8)[15] \\ = K_1[19,23] \oplus K_3[22] \oplus K_4[44] \oplus K_5[22] \oplus K_7[22]. \end{aligned} \tag{4.3.1}$$

Note that we are only interested in one bit of the term involving the F function (DES's f function), *i.e.* bit 15. The value of this bit will be determined by one S-box, and hence will be affected by only six subkey bits. So although equation (4.3.1) contains 48-bit subkey K_8, we need to try only possibilities for the six subkey bits that influence $F(C_R, K_8)[15]$, namely, $K_8[47], \ldots K_8[42]$.

Thus we require counters U_i for each possible value for the 6 subkey bits, and proceed with the linear cryptanalysis algorithm. We expect the maximum likelihood method to reveal which of the $2^6 = 64$ possible results is the correct one. Not only will we obtain the linear expression for a key bit (the right hand side of the DES linear approximation), but we also find out the value of the 6 subkey bits. Thus we should recover a total of 7 key bits (including a linear expression for a key bit).

Note that, in equation (4.3.1), we need only 7 bits of information from the plaintext/ciphertext. These 7 *text bits* include a one bit result of an XOR of plaintext and ciphertext bits, and six bit input to the S-boxes. Since we shall need to evaluate part of DES's f function, (denoted by F) in order to calculate the left hand side of equation (4.3.1), it is efficient to separate the data collection and data analysis into two distinct phases; the first phase we call *data collection phase* which is to collect the text bits from each plain-

text/ciphertext pair and the second phase we call *key counting phase* which is to procees the collected data from the first phase. By "collecting" the text bits, it suffices to increment a counter, U_x, for each plaintext/ciphertext (where x is a 7-bit string representing the 7 text bits extracted from the plaintext/ciphertext pairs). For example, we may store the value of

$$P_L[7, 18, 24] \oplus P_R[12, 16] \oplus C_L[15] \oplus C_R[7, 18, 24, 29],$$

into the MSB (Most Significant Bit) of x, and the six bits

$$C_R[0], C_R[31], \ldots C_R[27]$$

into 6 LSBs (Least Significant Bits) of x, then increment U_x.

Algorithm 4.3.1.

1. **Data Counting Phase:**
   Initialize T_i, where $0 \leq i < 128$ to zero.

2. For each plaintext P_i and the corresponding ciphertext C_i, where $0 \leq i < N$, do the following:

   - Let t hold the text bits used in the left hand side of the DES linear approximation equation. For example from equation (4.3.1):

     $$\begin{aligned} t \quad \leftarrow \quad & (P_L[7, 18, 24] \oplus P_R[12, 16] \oplus C_L[15] \oplus C_R[7, 18, 24, 29], \\ & C_R[0], C_R[31], C_R[30], C_R[29], C_R[28], C_R[27]) \end{aligned}$$

   - Increment T_t.

3. **Key Counting Phase:**
   For each $s \in \mathbb{Z}_2^6$, do the following:

   - Initialize U_s to zero.
   - For each $t \in \mathbb{Z}_2^7$, let $U_s = U_s + T_t$ if the left hand side of the DES linear approximation, with $F(relevant\_six\_bits\_of\_t, s)$ for the final round, is zero.

For example, from equation (4.3.1): if

$$t[6] \oplus F((t[5], t[4], t[3], t[2], t[1], t[0]), (s[5], s[4], s[3], s[2], s[1], s[0]))[15] = 0,$$

then let $U_s = U_s + T_t$. Note that $F(X, K)[15] \equiv S_1(X \oplus K)[2]$.

4. Let U_{max} be the maximum U_i counter, and let U_{min} be the minimum U_i counter.

5. If the probability of the DES linear approximation is greater than $1/2$ (for example, $p = 0.5 + 1.95 \times 2^{-10}$):

   - If $|U_{max} - N/2| > |U_{min} - N/2|$ then guess that the value of the right hand side of the DES linear approximation is 0 and the 6 key bits affecting $F(X, K)$ in the last round are given by *max*, for example, the right hand side of equation (4.3.1): $K_1[19, 23] \oplus K_3[22] \oplus K_4[44] \oplus K_5[22] \oplus K_7[22] = 0$, and *max* represents the six bits entering S_1 in the last round.
   - If $|U_{max} - N/2| < |U_{min} - N/2|$ then guess that the value of the right hand side of the DES linear approximation is 1 and the 6 key bits affecting $F(X, K)$ in the last round are given by *min*, for example, the right hand side of equation (4.3.1): $K_1[19, 23] \oplus K_3[22] \oplus K_4[44] \oplus K_5[22] \oplus K_7[22] = 1$, and *min* represents the six bits entering S_1 in the last round.

6. If the probability of DES linear approximation is less than $1/2$:

   - If $|U_{max} - N/2| > |U_{min} - N/2|$ then guess that the value of the right hand side of the DES linear approximation is 1 and the 6 key bits affecting $F(X, K)$ in the last round are given by *max*.
   - If $|U_{max} - N/2| < |U_{min} - N/2|$ then guess that the value of the right hand side of the DES linear approximation is 0 and the 6 key bits affecting $F(X, K)$ in the last round are given by *min*.

The complexity of the data counting phase is $O(N)$ and the complexity of the key counting phase is $O(2^{13})$. The complexity of the key counting phase can be reduced to $O(2^{12})$ modifying Step 3. in *Algorithm* 4.3.1 with the following:

3a. For each $s \in \mathbb{Z}_2^6$, do the following:

- Initialize U_s to zero.
- For each $t_l \in \mathbb{Z}_2^6$ (t_l are the 6 LSBs of t and $t[6]$ is the MSB of t), let $U_s = U_s + T_{(t[6]*2^6)+t_l}$ if $F(six\_bits\_of\_t_l, s)$ for the final round, is $t[6]$.
  For example, from equation (4.3.1): if

$$F((t_l[5], t_l[4], t_l[3], t_l[2], t_l[1], t_l[0]), (s[5], s[4], s[3], s[2], s[1], s[0]))[15] = 0,$$

  i.e. if $t[6] = 0$, then let $U_s = U_s + T_{t_l}$. Otherwise, if

$$F((t_l[5], t_l[4], t_l[3], t_l[2], t_l[1], t_l[0]), (s[5], s[4], s[3], s[2], s[1], s[0]))[15] = 1,$$

  i.e. if $t[6] = 1$, then let $U_s = U_s + T_{2^6+t_l}$.

Matsui [74] estimated that for a success rate of 97%, we need around 2^{20} known plaintexts. This estimate was based on calculation for the success rate in recovering 1-bit key as outlined in [74]. Using *Algorithm* 4.3.1, we recover 7 key bits (including one linear expression for a key bit), but we can also recover another 7 key bits because of the symmetry of DES by applying the 7-round linear approximation to the second to final rounds, and include $F(P_R, K_1)[15]$ to recover $K_1^1 = (K_1[47], \ldots K_1[42])$. So running the algorithm twice will be able to recover 14 key bits. The remaining 42 key bits can be recovered by reduced exhaustive key search.

Another improvement of this approximation is given in the next Section.

4.4 $(n-2)$-round Linear Approximations

In [75], Matsui made an improvement on the $(n-1)$-round linear approximation by suggesting that the linear approximation cover the $(n-2)$ 'middle' rounds of the cipher, and to use counters for the other two rounds.

This attack will recover one linear expression for a key bit, six bits of the subkey applied in the first round, and six bits of the subkey applied in the final round: a total of 13 key bits. The algorithm can be applied in "reverse" (provided that the linear approximation is not symmetrical) to recover another 13 key bits, so that we have at total of 24 key bits plus two linear expressions for 2 key bits. The remaining $(56 - 26) = 30$ bits can be found

by reduced exhaustive key search.

The n-round DES linear approximation is base don the best approximation of $(n-2)$-round DES. Thus, we will need much less known plaintexts to achieve a high success rate than the n-round and $(n-1)$-round algorithms. The best 6-round linear approximation for 8-round DES is

$$\begin{aligned} & P_L[7,18,24] \oplus C_L[15] \oplus C_R[7,18,24,29] \\ & \oplus F(P_R,K_1)[7,18,24] \oplus F(C_R,K_8)[15] \\ & = K_3[22] \oplus K_4[44] \oplus K_5[22] \oplus K_7[22]. \end{aligned} \tag{4.4.1}$$

Equation (4.4.1) holds with probability $1/2 - 1.95 \times 2^{-9}$.

The following algorithm 4.4.1 gives the steps involved in performing linear cryptanalysis of DES using $(n-2)$-round linear approximation.

Algorithm 4.4.1.

1. **Data Counting Phase:**
   Initialize T_i, where $0 \leq i < 2^{13}$ to zero.

2. For each plaintext P_i and the corresponding ciphertext C_i, where $0 \leq i < N$, do the following:

   - Let t hold the text bits used in the left hand side of the DES linear approximation equation. For example from equation (4.4.1):

$$\begin{aligned} t \leftarrow\ & (P_L[7,18,24] \oplus C_L[15] \oplus C_R[7,18,24,29], \\ & P_R[16], P_R[15], P_R[14], P_R[13], P_R[12], P_R[11], \\ & C_R[0], C_R[31], C_R[30], C_R[29], C_R[28], C_R[27]) \end{aligned}$$

   - Increment T_t.

3. **Key Counting Phase:**
   For each $s \in \mathbb{Z}_2^{12}$, do the following:

   - Initialize U_s to zero.
   - For each $t_l \in \mathbb{Z}_2^{12}$ (t_l are the 6 LSBs of t and $t[12]$ is the MSB of t), let $U_s = U_s + T_{(t[12]*2^{12})+t_l}$ if

$F(six\_MSB\_bits\_of\_t_l, six\_MSB\_bits\_of\_s) \oplus$
$F(six\_LSB\_bits\_of\_t_l, six\_LSB\_bits\_of\_s)$ for the final round, is $t[12]$.
For example, from equation (4.4.1): if

$$F((t_l[11], t_l[10], t_l[9], t_l[8], t_l[7], t_l[6]), (s[11], s[10], s[9], s[8], s[7], s[6]))[7, 18, 24] \oplus F((t_l[5], t_l[4], t_l[3], t_l[2], t_l[1], t_l[0]), (s[5], s[4], s[3], s[2], s[1], s[0]))[15] = 0,$$

i.e. if $t[12] = 0$, then let $U_s = U_s + T_{t_l}$. Otherwise, if

$$F((t_l[11], t_l[10], t_l[9], t_l[8], t_l[7], t_l[6]), (s[11], s[10], s[9], s[8], s[7], s[6]))[7, 18, 24] \oplus F((t_l[5], t_l[4], t_l[3], t_l[2], t_l[1], t_l[0]), (s[5], s[4], s[3], s[2], s[1], s[0]))[15] = 1,$$

i.e. if $t[12] = 1$, then let $U_s = U_s + T_{2^{12}+t_l}$.

Note that $F(X, K)[15] \equiv S_1(X \oplus K)[2]$ and $F(X, K)[7, 18, 24] \equiv S_5(X \oplus K)[1, 2, 3]$

4. Let U_{max} be the maximum U_i counter, and let U_{min} be the minimum U_i counter.

5. If the probability of the DES linear approximation is greater than 1/2:

   - If $|U_{max} - N/2| > |U_{min} - N/2|$ then guess that the value of the right hand side of the DES linear approximation is 0 and the 12 key bits affecting $F(X, K)$ in the first and last rounds are given by *max*, for example, the right hand side of equation (4.4.1): $K_3[22] \oplus K_4[44] \oplus K_5[22] \oplus K_7[22] = 0$, and *max* represents the twelve bits entering S_1 and S_5 in the first and last rounds.
   - If $|U_{max} - N/2| < |U_{min} - N/2|$ then guess that the value of the right hand side of the DES linear approximation is 1 and the 12 key bits affecting $F(X, K)$ in the first and last rounds are given by *min*, for example, the right hand side of equation (4.4.1): $K_3[22] \oplus K_4[44] \oplus K_5[22] \oplus K_7[22] = 1$, and *min* represents the twelve bits entering S_1 and S_5 in the first and last rounds.

6. If the probability of DES linear approximation is less than 1/2:

   - If $|U_{max} - N/2| > |U_{min} - N/2|$ then guess that the value of the right hand side of the DES linear approximation is 1 and the 12 key bits affecting $F(X, K)$ in the first and last rounds are given by *max*.

- If $|U_{max} - N/2| < |U_{min} - N/2|$ then guess that the value of the right hand side of the DES linear approximation is 0 and the 12 key bits affecting $F(X, K)$ in the first and last rounds are given by *min*.

As we can see from the algorithm above, we only need some modification on the previous algorithm 4.3.1 to be able to perform linear cryptanalysis using $(n-2)$-round linear approximation. Instead of having a total of 128 text counters T_i, we have $2^{13} = 8192$ text counters and instead of having a total of 64 key counters U_j, we have $2^{12} = 4096$ key counters. The complexity of the above algorithm is $O(N) + O(2^{24})$

Let us now move on to issues on our implementation of linear cryptanalysis of 8-round DES.

4.5 Implementation Issues

We managed to successfully have a practical implementation of linear cryptanalysis of 3-round DES in finding 1 key bit. In the linear cryptanalysis of 8-round DES, we managed to have practical implementations for linear cryptanalysis using $(n-1)$-round linear approximation expression and also the $(n-2)$-round linear approximation expression. Previously, we mentioned the complexity of the linear cryptanalysis attack is $O(N) + O(2^{12})$ when we use $(n-1)$-round linear approximation expression and $O(N) + O(2^{24})$ when we use $(n-2)$-round linear approximation expression. The linear cryptanalysis attack on 8-round DES was fully implemented to recover all 56-bit key. If we use $(n-1)$-round linear approximation expression for our attack, we would only be able to recover 7 key bits including one bit for the linear expression involving key bits. Table 4.3 shows the percentage of success in recovering 7 key bits for 100 trials. It is also possible to recover an extra 7 bits, if we employ the method mentioned in Section 3 above, *i.e.* to apply the same $(n-1)$-round linear approximation expression to round 2 to round 8 of 8-round DES. But, it still leaves $56 - 14 = 42$ key bits to find using reduced exhaustive key search. It is possible to do exhaustive key search in finding the rest of the 42 bit keys but since the DES implementation can do 3792400 encryptions/decryptions in 1 second with parallel programming using 10 threads, we estimated that it would take $(2^{42}/3792400) = 1159700$ seconds or around 13 days to search for the correct key. However, if we use $(n-2)$-round linear approximation expression for linear cryptanalysis of 8-round DES, we would be able

Table 4.3: Percentage of Success for obtaining 7 key bit using *Algorithm* 4.3.1 (100 trials)

N	262144	524288	1048576	2097152
Percentage of Success	56%	75%	85%	100%
Average position of successful attack in key list	10	4	1	1
Average time taken	2.06s	4.14s	8.24s	16.51s

Table 4.4: Percentage of Success for obtaining 23 key bit using *Algorithm* 4.4.1 (100 trials)

N	262144	524288	1048576	2097152
Percentage of Success	49%	93%	100%	100%
Average position of successful attack in key list	4	1	1	1
Average time taken	2.66s	4.82s	9.13s	17.66s

to recover 13 key bits including one key bit for the linear expression involving key bits. Applying $(n-2)$-round linear approximation expression in "reverse" would recover an additional 13 key bits. So in total we have 26 recovered bits. This leaves us with $56-26=30$ bits to search for. If we use exhaustive key search, we would be searching 2^{30} key space for rest of the bits. An additional problem is that the 26 key bits that were recovered were not all distinct key bits because 3 of them are the same due to DES key scheduling process. So we would be able to recover only 23 key bits altogether and we would need to do exhaustive key search on the other 33 key bits. We managed to find the correct key with parallel programming using 10 threads. Recovering the 23 key bits using linear cryptanalysis only takes less than 20 seconds and it takes around 4 minutes to retrieve the rest of the key bits. Table 4.4 shows the percentage of success in recovering 23 key bits for 100 trials. In Table 4.4, the average time taken excludes the time taken for doing exhaustive key search for the rest of the key bits.

Chapter 5

DIFFERENTIAL-LINEAR CRYPTANALYSIS OF 8-ROUND DES

In this Chapter we discuss our implementation of differential-linear cryptanalysis. Although this attack recovers fewer key bits it has the advantage that it requires less data. We were able to run experiments to determine the success rate and show that with fewer than 1000 texts the key bits are found in a matter of seconds.

Differential-Linear Cryptanalysis is a chosen plaintext attack which combines both Differential and Linear cryptanalysis. It was introduced by Susan Langford et.al in [70]. Since its introduction, differential-linear cryptanalysis has been applied to other block ciphers such as FEAL [2], IDEA [25, 26, 18], Serpent [15, 38], Camellia [94], CTC2 [39, 72] and SHACAL-2 [89]. With this method, the amount of chosen plaintexts needed to do cryptanalysis was drastically reduced.

Before we move on further in discussing about the topic of differential-linear cryptanalysis, we should first outline the notation that is going to be used in this Chapter.

- The numbering of plaintext, ciphertext and the bits of intermediate results (L_n, R_n) are numbered from 0 to 63 reading from right to left. This numbering is similar to the one specified in the previous Chapter about linear cryptanalysis.
- Input to an S-box is taken as with Matsui's notation as $(x_5, x_4, x_3, x_2, x_1, x_0)$

- $A[i]$ represents the ith bit of A
- $A[i, j, \ldots, k] = A[i] \oplus A[j] \oplus \ldots \oplus A[k]$

For differential-linear cryptanalysis, similar to both differential and linear cryptanalysis, we shall ignore the initial (**IP**) and final (**FP**) permutations as they have no cryptographic significance in a chosen or known plaintext attack. In this Chapter, we refer (L_0, R_0), the 64-bits after **IP** as the plaintext and (L_n, R_n), the 64-bit before **FP** as the ciphertext. Lastly, any primed values from here onwards refer to differentials.

Now, let us move on the Differential-Linear Cryptanalysis on 8-round DES to give a general idea of how the attack works.

5.1 Differential-Linear Cryptanalysis of 8-round DES

Figure 5.1 shows Differential-Linear attack on 8-round DES. This attack combines a differential characteristic with a linear parity relation. By a technique given in the next Section, we can complement bits 29 and/or 30 of L_1 and keep the other 62 bits of (L_1, R_1) unchanged. The key observation in this attack is that this behaviour in (L_1, R_1) leaves many bits of (L_4, R_4) unchanged. In particular the input bits to Matsui's best 3-round parity relation (bits 7, 18, 24 and 29 of L_4 and bit 15 of R_4) *never* change, so that the parity of the output bits (bits 7, 18, 24 and 29 of L_7 and bit 15 of R_7) is unchanged with probability $r = p^2 + q^2 = 0.576$ where $p = 0.695$ is the probability of Matsui's parity relation holding once, and $q = 1 - p$. The probability is $p^2 + q^2$ because Matsui's parity relation must hold, or fail, twice: once for the reference (see next Section for details) and once for the plaintext which toggles only bits 29 and/or 30 in round 1.

Figure 5.2 shows why the differential characteristic holds going from (L_1, R_1) to (L_4, R_4). Because $R'_1 = 0$, the output of $F(R, K)$ in round 2 must also be 0 differentially. Thus, $R'_2 = L'_1$ and R'_2 has bits 29 and/or 30 toggled. These two bits affect only the input of S-box S_1, so only the output of S_1 can change in round 3. Since $R'_3 = L'_4$ and bits 7, 18, 24 and 29 are not outputs of S_1, these bits will be unchanged in L'_4. Further, because of expansion **E**, the four outputs of S_1 affect the inputs of six S-boxes in round 4. Two S-boxes will therefore be unchanged, namely S_1 and S_7. Bit 15 of R'_4 is the output of S_1 so it will remain unchanged.

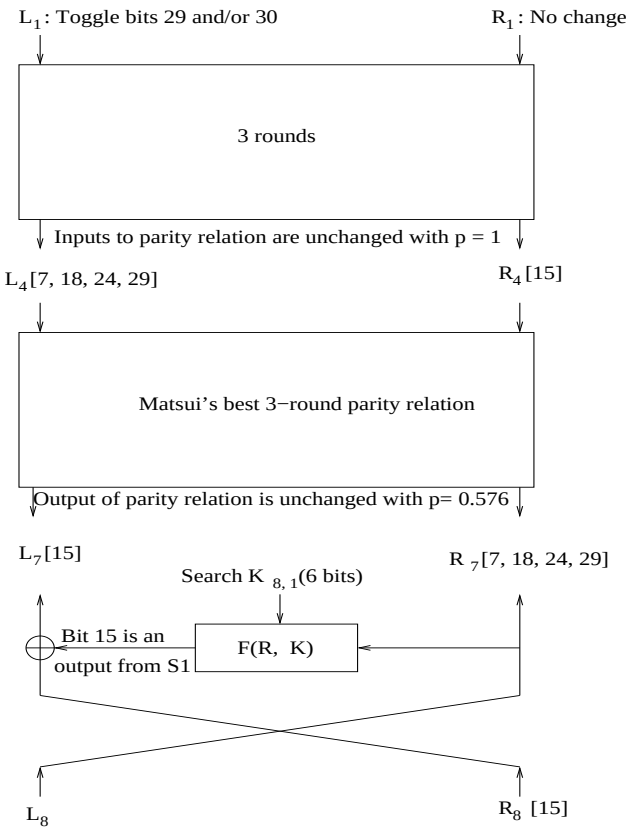

Figure 5.1: Differential-Linear attack on 8-round DES

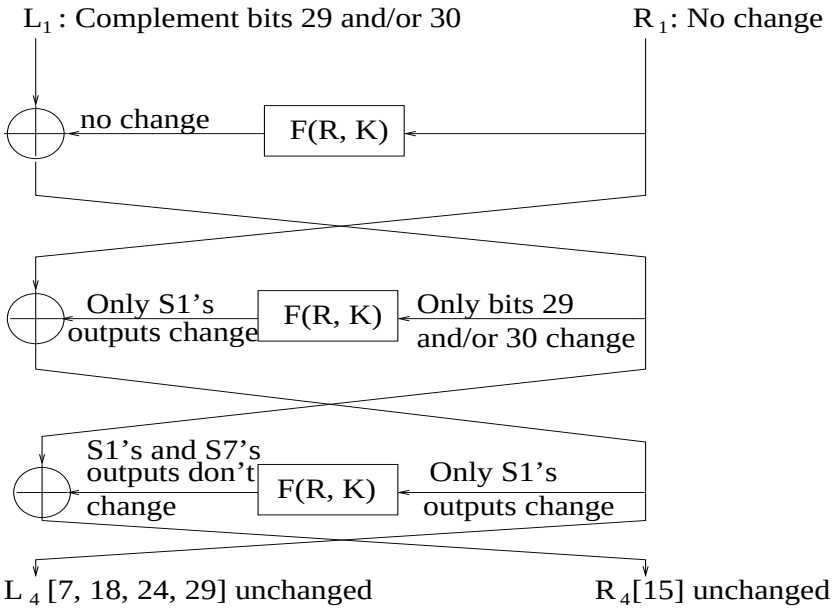

Figure 5.2: Differential Characteristic for Differential-Linear Attack

As Figure 5.1 shows, the parity invariance to be observed occurs in round 7 with probability 0.576. Using Matsui's method, we decipher the two ciphertexts (L_8, R_8) and (L_8^*, R_8^*) backwards through one round to get the output bits of the parity relation: bits 7, 18, 24 and 29 of R_7, and bit 15 of L_7. Bits 7, 18, 24 and 29 are known because $R_7 = L_8$, the left half of the ciphertext. Only bit 15 of L_7 must be computed. This computation only involves only S_1 in round 8, so we can test the 6-bit subkey $K_{8,1}$. When the correct value of $K_{8,1}$ is used, we expect to observe parity invariance 57.6% of the time; when an incorrect value is used, the data produced is more random and we expect the observed parity invariance closer to 50% of the time.

Based on Matsui's rule of thumb [74, 75], we would require approximately $8/(r - 0.5)^2$ observations, where r is the probability of observing a parity relation. So, we expect that the differential-linear attack would require about 1400 pairs of chosen plaintexts.

Let us now move on to describing how the actual differential-linear attack is done.

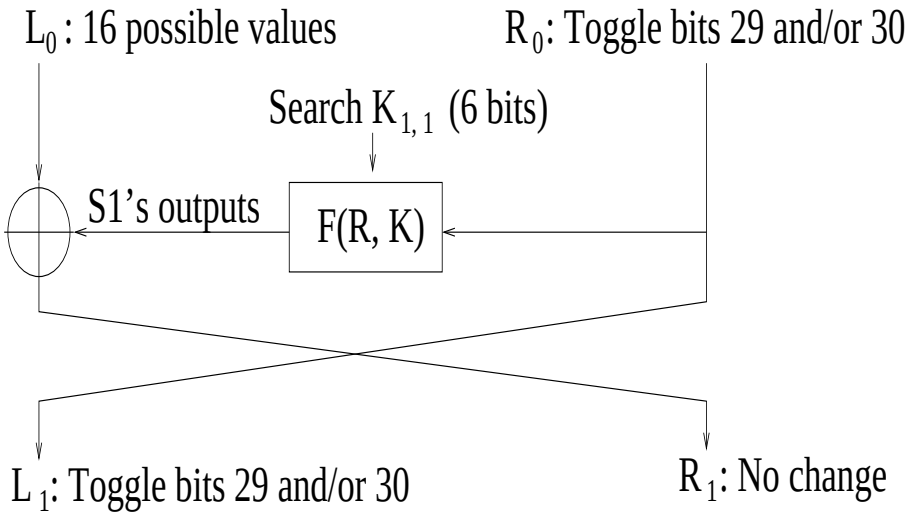

Figure 5.3: First round of 8-round Differential-Linear Attack

5.2 Differential-Linear Attack Setup

Differential-Linear Cryptanalysis requires that we have plaintext pairs which we can toggle only certain bits. The first thing we need to do is to choose any reference plaintext and let $P(0)$ through $P(63)$ be the 64 plaintexts obtained by varying bits 1, 9, 15 and 23 of L_0 and bits 29 and 30 of R_0. We shall refer $P(0)$ through $P(63)$ as a plaintext structure. The bits 1, 9, 15 and 23 correspond to the four outputs of S_1 , and bits 29 and 30 of R_0 become bits 29 and 30 of L_1, the bits to be toggled.

Bits 29 and 30 are the middle input bits to S_1. Since these bits are the only bits that can change in R_0 , only the outputs of S_1 can change in round 1, as shown in Figure 5.3.

Because we included all 16 possibilities for these bits in the plaintext structure, if we knew the 6-bit subkey $K_{1,1}$, for each of the 64 $P(i)$'s we could choose three other $P(j)$'s which had the desired toggling in round 1. One $P(j)$ would toggle bit 29, one would toggle bit 30, and one would toggle both bits 29 and 30. The 64 chosen plaintexts might seem to produce $64 \times 3 = 192$ differential pairs, but only half of them, or 96 pairs,are distinct because $(P(i), P(j))$ and $(P(j), P(i))$ are the same pair. All the 96 distinct pairs in a plaintext structure could be used to help determine $K_{8,1}$ if $K_{1,1}$ were known. Since we do

not know $K_{1,1}$, we search over all values of $K_{1,1}$ and $K_{8,1}$

Two of the bits of $K_{1,1}$ are also part of $K_{8,1}$, so there are only 10 distinct bits and 1024 possible subkeys to search over. Since each plaintext structure of 64 plaintexts produces 96 differential pairs and 1400 pairs are required by Matsui's rule of thumb, the differential-linear attack should work with approximately $64/96 \times 1400 = 900$ chosen plaintexts. Langford [70] specified that 512 chosen plaintexts would produce 80% success rate and 768 chosen plaintexts would produce 95% success rate in obtaining the 10 bits of key in $K_{1,1}$ and $K_{8,1}$. These two attacks use 8 and 12 plaintext structures of 64 chosen plaintexts respectively.

5.3 Obtaining Additional Key Bits

In the previous Section, we had shown how we can obtain 10 key bits using Differential-Linear attack. However, DES has a 56-bit key. So, this still leaves 46 more bits to find which is still considerably large if we use exhaustive key search to find them.

In order to recover additional key bits, we can use other lower probability 3-round parity relations in rounds 5 to 7, replacing Matsui's optimal 3-round parity relation. For example, bits 5, 11, 17 and 27 of L_4 and bit 1 of R_4 in round 4 also remain unchanged when bits 29 and/or 30 of L_1 are toggled. We can therefore use the relation

$$(L_4[5, 11, 17, 27] \oplus R_4[1]) \oplus (R_7[5, 11, 17, 27] \oplus L_7[3]) = 0. \tag{5.3.1}$$

Differentially, equation 5.3.1 holds with probability 0.527, instead of 0.576 for Matsui's optimal relation. Using this parity relation requires searching over $K_{1,1}$ and $K_{8,6}$ instead of $K_{1,1}$ and $K_{8,1}$, therefore recovering the six additional bits of $K_{8,6}$.

5.4 Dependence Issues

The Differential-Linear attack on 8-round DES appears at first, to be a major improvement to Linear Cryptanalysis but according to [71], these two attacks are are actually equivalent, *i.e.* the amount of text required for the Differential part in Differential-Linear attack is equivalent to the ones required in Linear Cryptanalysis.According to [71], if T_k represent

the count for a particular candidate subkey, $\sum_i P(i)$ represents the count obtained from a standard linear attack and N is the total number of known plaintexts used in linear part of the Differential-Linear Attack, we can derive the count T_k to be the following:

$$T_k = \frac{N^2}{4} - [\sum_i P(i) - \frac{N}{2}]^2$$

Further, because T_k is monotonic in $abs(\sum_i P(i) - \frac{N}{2})$, the subkey which generates the maximum value for $abs(\sum_i P(i) - \frac{N}{2})$ in the standard linear attack will also generate the minimum value of T_k in the differential style attack. Thus, the two attacks choose the same subkeys. For further information on this, we refer you to [71]. We had adjusted the count T_k in our implementation accordingly. The differential attack (in Differential-Linear Cryptanalysis) initially appears superior to the standard attack because of a neglected assumption that the plaintext pairs are independent for which they are actually not. In [71], another cryptanalysis was performed on 8-round DES also took into account, the dependence of plaintext pairs. Surprisingly, the results obtained were similar to the simplified cryptanalysis on 8-round DES which neglects dependence of pairs.

5.5 Implementation Issues

We were able to successfully implement the Differential-Linear attack on 8-round DES following the outlined steps given in Section 5.2. However, the amount of key material that were recovered from this attack is less than that obtained either by Differential or Linear Cryptanalysis. The advantage against both attacks (*i.e.* Differential and Linear Cryptanalysis) is that the amount of required plaintexts are drastically reduced. Using Differential-Linear attack, we were able to recover only 10 key bits instead of 12 key bits because two of the recovered key bits are the same. In the previous Section, we can recover an additional 6 key bits, which gives us 16 recovered bits in total. This still leaves us $56 - 16 = 40$ key bits to search for. As we had mentioned in the previous Chapter, it is

However, we decided to test the performance on the percentage of success in obtaining the 10 key bits. Table 5.1 shows the results of our experiment with 100 trials on each of the given number of chosen plaintexts.

We did look at recovering 16-bit key and found that the results were similar to table 5.1.

Table 5.1: Percentage of Success for 10-bit Key Recovery

Number of Chosen Plaintexts	64	128	256	512	768
Percentage of Success	77.7%	97.2%	100%	100%	100%
Average Position in List	3.00	1.00	1.00	1.00	1.00
Average Time Taken	0.0092s	0.0113s	0.0152s	0.0228s	0.0283s

Chapter 6

MULTIPLE LINEAR APPROXIMATIONS IN DIFFERENTIAL-LINEAR CRYPTANALYSIS OF 8-ROUND DES

In this Chapter we discuss the extension of differential-linear cryptanalysis to multiple linear approximations. We use an alternative method of generating plaintexts in our implementation.

Multiple Linear Approximations in Linear Cryptanalysis was introduced by Burton S. Kaliski Jr. and M.J.B. Robshaw [52] as an enhancement of Linear Cryptanalysis introduced by Matsui [74]. We know that in Linear Cryptanalysis we can use a single linear approximation to recover certain key bits of ciphers like DES, for instance. In addition to this, linear cryptanalysis also recovers parity bits of the key. Normally, the best linear approximation is used in linear cryptanalysis. Kaliski and Robshaw [52] first introduced the idea of using more than one linear approximation in linear cryptanalysis in 1994. The advantage of using multiple linear approximations rather than a single one is the increase in success rate without having to increase the number of known plaintexts.

In Differential-Linear cryptanalysis of 8-round DES, there are two parts to the cryptanalysis *i.e.* the differential cryptanalysis part and the linear cryptanalysis part. In this thesis, we are concentrating on improving the linear cryptanalysis part. Details of differential-linear cryptanalysis of 8-round DES can be found in [71].

Differential-Linear cryptanalysis of 8-round DES uses Matsui's optimal 3-round approximation in the linear part of the cryptanalysis. In this thesis, we are investigating the feasibility of using multiple 3-round linear approximations to decrease the number of required plaintexts.

Now, let us discuss about the different linear approximations that we can use in improving differential-linear cryptanalysis

6.1 Multiple Linear Approximations

Firstly, note that the i^{th} round DES subkey K_i is divided into 6-bit blocks so that $K_{i,j}$ refers to the j^{th} block of the subkey K_i (reading from left to right). S_i refers to the i^{th} S box of DES. Also, $K_i[j]$ refers to the j^{th} bit of the subkey K_i where the rightmost bit of K_i is $K_i[0]$.

The linear cryptanalysis part of the differential linear cryptanalysis relied on bits that are unchanged after going through the differential part of the cryptanalysis. The linear cryptanalysis part begins from round 5 to round 7. Figure 5.1 shows the setup used in differential-linear cryptanalysis of 8-round DES. After going through the 3-round differential characteristic, we are left with several bits which are unchanged. These bits are the ones that are used in the linear cryptanalysis part of differential-linear cryptanalysis. Figure 5.2 shows how a differential characteristic is used to produce unchanged bits to be used in linear cryptanalysis.

In Figure 5.2, we see that bits 7, 18, 24 and 29 of L_4 and bit 15 of R_4 are unchanged. These are not the only bits which are unchanged. Along with these bits, we also have bits 0, 2, 3, 4, 5, 6, 8, 10, 11, 12, 13, 14, 16, 17, 19, 20, 21, 22, 25, 26, 27, 28, 30 and 31 of L_4 and bits 0, 1, 9, 10, 20, 23 and 25 of R_4 unchanged, using the differential characteristic in Figure 5.2. We need to construct parity relations having different biases in order to

improve the existing Matsui's optimal parity relation

$$(L_4[7, 18, 24, 29] \oplus R_4[15]) \oplus (L_7[15] \oplus R_7[7, 18, 24, 29]) = 0$$

which has a bias of $\epsilon = 1.95 \times 10^{-1}$ (by Matsui's Piling Up Lemma). Let us look at parity relations involving these unchanged bits. The parity relations that are used involving these bits are of the form:

$$(L_4[V] \oplus R_4[W]) \oplus (L_7[X] \oplus R_7[Y]) = 0 \tag{6.1.1}$$

with bias ϵ (using Matsui's Piling Up Lemma) where V, W, X, Y and ϵ and the cumulative sum of squares of the bias ϵ are given in Table 6.1. We can see that the 3 linear approximations in Table 6.1 are amongst the best linear approximations that we can use in differential-linear cryptanalysis of 8-round DES. The main idea is to search for several correct key bits used in 8-round DES encryption. In order to do this, we need to know which DES's S boxes are involved. The values of X in Table 6.1 indicate which key bits we can search for. In Table 6.1, only the first and second linear approximations involve the same S box,*i.e.* S_1, because bit 15 and bit 1 (in column X in Table 6.1) are output bits from S_1. This means that we can search for the 8-round DES subkey, $K_{8,1}$ using these two approximations. Although by using these two approximations we can recover only the same key bits, the advantage is that we are able to reduce the number of required chosen plaintext pairs. The third linear approximation in Table 6.1 involves S_6 so that we can search for the subkey, $K_{8,6}$. Altogether we should be able to recover a total of 12 key bits from using these 3 linear approximations. However, in order to use these approximations, equation 6.1.1 needs to be satisfied for each of the linear approximations. Equation 6.1.1 depends on the XOR (exclusive or) of certain key bit values. Parity relations need to be satified for all the linear approximations in Table 6.1. The parity relation for the first linear approximation is $K_5[22] \oplus K_7[22] = 0$. For the second linear approximation, the parity relation is $K_5[2] \oplus K_7[2] = 0$. In the third linear approximation, we notice that the ϵ value in Table 6.1 is negative so the parity relation is $K_5[2] \oplus K_7[4] \oplus 1 = 0$. As $K_5[2]$ is involved in both the second and third linear approximations, there are only 5 key bits involved in these parity relations.

Table 6.1: Values of V, W, X, Y and ϵ for $K_{8,1}$ and $K_{8,6}$ key search

No.	V	W	X	Y	ϵ	Cumulative $\sum \epsilon^2$
1	7,18,24,29	15	15	7,18,24,29	1.95×10^{-1}	$3.80 \times 10{-}2$
2	5,11,17,27	1	1	5,11,17,27	9.58×10^{-2}	4.72×10^{-2}
3	5,11,17,27	1	3	5,11,17,27	-1.10×10^{-1}	5.93×10^{-2}

6.2 Implementation

In [70, 71], we see that a plaintext structure was used in order to generate the chosen plaintext pairs required for differential-linear cryptanalysis of 8-round DES. We decided not to use any plaintext structure in our experiment. Figure 6.1 shows how we generate the required chosen plaintext pairs in the first round of 8-round DES. For each plaintext, say P, we get another plaintext, say P', with bits 29 and/or 30 in the right half of P' toggled and the bits 1, 9, 15 and 23 of the left half of P' changed. Note that each half of P or P' is 32 bits in length starting from bit 0 to bit 31 with bit 0 being the rightmost bit. The right halves of P and P' goes into the F function of the first round of DES together with the first 48 bit subkey of the 8-round DES key, K, to produce a differential output, say Δ. We use this Δ to produce the left half of P' by XORing Δ with the left half of P. Thus, we finally XOR the left halves of P and P' with the two respective outputs of function F to produce a differential output of zero, which is required for the next stage in differential-linear cryptanalysis of 8-round DES as previously mentioned. In this first round, we are also able to search for the correct 6-bit subkey $K_{1,1}$. Altogether, we should be able to recover the correct key bits of $K_{1,1}$, $K_{8,1}$ and $K_{8,6}$ using this setup and using multiple linear approximations for differential-linear cryptanalysis of 8-round DES. However, since we have 2 key bits which are duplicated in $K_{1,1}$ and $K_{8,1}$, we are only able to recover a total of 16 key bits. When we use multiple linear approximations for differential-linear cryptanalysis of 8-round DES, the result obtained is a just key ranking of all possible keys. Increase in the number of chosen plaintext pairs used will "promote" the correct key bits to the top rank in the key ranking list.

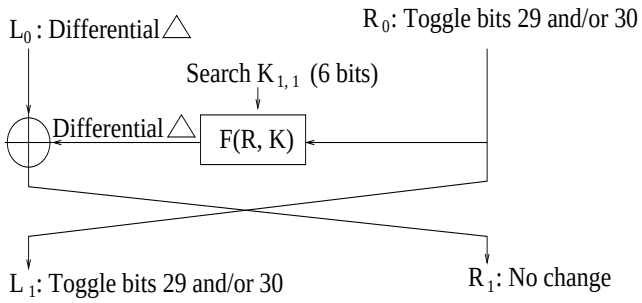

Figure 6.1: First round in Differential-Linear Attack of 8-round DES

6.3 Prediction

According to Matsui's rule of thumb, if we use only the single optimal linear approximation for the linear part of the differential-linear cryptanalysis of 8-round DES, we can estimate the number of plaintext pairs required with a very high percentage of success using the expression $8/(r - \frac{1}{2})^2$ where $r = \frac{1}{2} + 2\epsilon^2$ and ϵ is the bias for the optimal linear approximation. So, as the bias $\epsilon = 1.95 \times 10^{-1}$ for the first linear approximation in Table 6.1, we predict that the number of plaintext pairs required is about 1384 pairs. By multiplying our initial estimate of required plaintext pairs using the single linear approximation, *i.e.* 1384 pairs, with the square of the bias of using the first linear approximation and then dividing it with the cumulative sum of squares of the bias using the first two linear approximations in Table 6.1, *i.e.* $\frac{3.80\times10^{-2}}{4.72\times10^{-2}} \times 1384$, we estimate that we can reduce the plaintext pair requirement to about 1115 plaintext pairs without reducing the percentage of success. Similarly, if we use all the three linear approximations in Table 6.1, we predict that we would require about 887 plaintext pairs by using the same formula given above except we divide with the cumulative sum of squares of the bias using three linear approximations instead of dividing with the cumulative sum of squares of the bias using two linear approximations, *i.e.* $\frac{3.80\times10^{-2}}{5.93\times10^{-2}} \times 1384$.

6.4 Experimental Result

Table 6.2 gives the result of our implementation in using multiple linear approximations in differential-linear cryptanalysis of 8-round DES. The result is based on performing 100

Table 6.2: Recovery of 16 key bits using Multiple Linear Approximations in Differential-Linear Cryptanalysis of 8-round DES

No. of plaintext pairs used	1384	1115	887
Percentage of success	100%	100%	100%
Average position in key ranking list	1.00	1.00	1.00

trials using 100 random keys (with certain key bits set to specific values as previously described above). Table 6.2 indicates the position of the correct key bits out of 65536 possible key bit positions. As we can see from Table 6.2, the result that we obtained conformed to the prediction that we had previously made.

6.5 Issues Regarding Multiple Linear Approximations

In Section 6.1, we had used three linear approximations to perform our cryptanalysis. In fact, there are four linear approximations that we can actually use but we opted not to use the fourth one because it would be unweildy for a normal PC since we would be searching a keyspace of 2^{28} for the correct key bits. If we use only three linear approximations as we had done in this Chapter, we would be searching a keyspace of only 2^{16}.

In Section 6.2, we had used an implementation which does not require the use of any plaintext structures because with plaintext structures we are restricted to only multiples of 64 plaintext pairs whereas without it, we do not have any such restriction. On the other hand, using plaintext structures takes advantage of using a particular pair and generating the other remaining pairs from it. Whereas in our implementation, we just generate any plaintext pair and modify it accordingly and then move on to generating the next plaintext pair and so on until the required number of plaintext pairs has been reached. Regardless, the main idea in this Chapter is that we are able to use a different implementation which we can use to predict the plaintext pair requirements.

We have now finished our discussion about the cryptanalysis of 8-round DES. In the next Chapter, we will discuss about our proposal on distinguishing DES from a random permutation.

Chapter 7

OUR PROPOSAL ON DISTINGUISHING DES FROM A RANDOM PERMUTATION

In this Chapter, we discuss several statistical tests that have been applied to DES. The purpose of these statistical tests was to determine whether we can actually distinguish between the ciphertext output of a block cipher and a random permutation. Our tests rely on ciphertexts produced from various rounds of DES, from 1-round to the full 16-round version. We want to determine how many ciphertexts are required in order to make this distinction. We also want to determine, in the case of block ciphers, the number of rounds required in order for the ciphertexts to be indistinguishable from a random permutation. These statistical tests were based on the Strict Avalanche Criterion which will be explained in the next Section.

7.1 The Strict Avalanche Criterion (SAC) Test

The Strict Avalanche Criterion was originally presented in [45] in 1990, as a generalisation of the avalanche effect. It was devised for measuring the amount of non-linearity of substitution boxes, a vital component of many block ciphers.

The avalanche effect tries to reflect the idea of high-nonlinearity. If we make a small change in the input, it produces a huge change in the output with high probability. In

mathematical terms,

$$\forall x, y \,|\, H(x, y) = 1 \quad Average(H(F(x), F(y))) = \frac{n}{2} \tag{7.1.1}$$

where F is the function to be tested and $H(x, y)$ refers to the hamming distance/weight between x and y.

If F is to have the avalanche effect, the Hamming distance between outputs of a random input vector and one generated by randomly flipping one of the bits, should be $n/2$ on average. This means, a minimum change in the input (one bit only) produces, on average, a maximum output change (half of the bits).

This becomes our basis for using this statistical test is to realize the concept of independence of the output from the input. The ideal function F will resemble a perfect random function where inputs and outputs are statistically unrelated. Any such function would have perfect avalanche effect.

Moreover, we used the Strict Avalanche Criterion (SAC) as a property that implies the avalanche effect, and this can be described mathematically as,

$$\forall x, y \,|\, H(x, y) = 1 \quad H(F(x), F(y)) \approx B(n, \frac{1}{2}) \tag{7.1.2}$$

where B is the binomial distribution.

Given this notion that we produce a distribution of the output data that closely relates to another distribution (in this case, the Binomial distribution), we can use the χ-square goodness-of-fit test.

The following pseudocode describes the SAC statistical test that we used. If a function F passes this test, then it means that F has the SAC property.

```
1. While (No._of_input_vectors <  Total_No._of_input_vectors)
   {
   1.1 Randomly generate an input vector V;
   1.2 Randomly choose a position p in V;
   1.3 Flip the contents of the position p in V to get V';
   1.4 Calculate h the hamming distance between F(V) and F(V') and
       increment the number in the array observed_value at index h
       (observed_value[h]);
   };
2. Compute the Chi-squre statistic t over the results stored in
```

```
   observed_value[h]
3. Perform a hypothesis constrast to test if the observed distribution
   (the t value) is consistent with the expected probability distribution,
   i.e. Binomial(n,0.5), where n is the length of the output of F
```

In addition to this, we also want to test whether the F function can generate random numbers. If F is a random permutation, the recurrence relation:

$$I_0 = IV$$
$$I_{n+1} = F(I_n)$$

where $I_0 = IV$ is the intial vector, must produce a sequence $(I_1, I_2, I_3, \dots)$ of random numbers. This sequence of random numbers can be seen as a stream of random bits that could be used to generate V and p in the SAC test described above. This technique is called autofeeding. Any result statistically distinguishable from the expected will prove that the F function has no SAC property or is not a good random number generator. The SAC test and the SAC test with autofeeding was first introduced in [48]. In [48], the SAC test and the SAC test with autofeeding was performed on reduced rounds of the TEA block cipher. In this thesis, we performed these statistical tests on DES.

As an enhancement to the SAC tests described above, we had modified the SAC test to form a new test which will be explained in the following Section.

7.2 Modified Strict Avalanche Criterion (modSAC) Test

The test that we describe here is based on the SAC test explained earlier. While the SAC test uses the avalanche effect to measure the change in the hamming distance of the input string against the hamming distance of the output string, our modified SAC test uses the avalanche effect to see the consequence of the change in the hamming distance of the input strings to the specific n-bit substrings of the output strings, where $n = 2, 4, 8$. The reason we refer to the input and output as strings is because we normally represent them as strings of numbers, in particular, binary numbers, especially, when we consider the function F as a block cipher algorithm such as DES which uses 64 bits as input and produces 64 bits as output. Our modified test is a more localized test as it looks at substrings of limited lengths. Moreover it is a more refined test as we consider individual

strings of these lengths resulting from an XOR rather than their weight (corresponding to a Hamming distance).

The function F would produce two output strings $F(x)$ and $F(y)$ from the input strings x and y respectively. We then divide these output strings into n-bit substrings which we represent as $F(x[n])$ and $F(y[n])$. When we XOR $F(x[n])$ with $F(y[n])$, the result would be an n-bit substring (represented as an integer i between 0 and $2^n - 1$). In the ideal case, for each value i, the probability that i occurs is uniform and given as $1/2^n$. We describe mathematically as:

$$\forall x, y \,|\, H(x,y) = 1 \quad P((F(x[n]) \oplus F(y[n])) = i) = \frac{1}{2^n}, \quad \text{for} \quad i = 0_2, \ldots, (2^n - 1)_2 \quad (7.2.1)$$

where $x[n]$ and $y[n]$ represent the n-bit substrings of x and y respectively.

The following pseudocode describes the modSAC test that we used. If a function F passed this test, then it also means that F has the modSAC property.

```
1. While (No._of_input_vectors <  Total_No._of_input_vectors)
   {
   1.1 Randomly generate an input vector V;
   1.2 Randomly choose a position p in V;
   1.3 Flip the contents of the position p in V to get V';
   1.4 For (i = 0; i < 2^n; i++)
        {
        1.4.1 Search for  i in binary form (length n),
              in the output from XORing F(V[n]) with  F(V'[n])
              and  increment the number in the array
              observed_value at index i  (observed_value[i]), if found;
       };
   };
2. Compute the Chi-squre statistic t over the results stored in
   observed_value[i]
3. Perform a hypothesis constrast to test if the observed distribution
   (the t value) is consistent with the expected discrete uniform
   probability distribution where each of the i values has probability 1/n.
```

As in the previous Section, we also test whether the F function can generate random numbers. The autofeeding technique that was described previously would also be used in our modSAC test.

7.3 Rationale of Experimental Tests

Hernandez *et al.* [48, 29] used the SAC Test and SAC Test with Autofeeding on reduced round TEA encryption algorithm. Strict Avalanche Criterion is a good measure of non-linearity in encryption algorithms. However, it was mentioned that the SAC test has not been proven independent from other classical randomness tests [49]. Although, this is also true for other classical randomness tests as well. Regardless, we decided to use the same SAC Test and SAC Test with Autofeeding together with our modified SAC Tests, *i.e.* 2-bit modSAC, 4-bit modSAC and 8-bit modSAC Tests with/without Autofeeding, on two different encryption algorithms (*i.e.* DES and RC5). DES and RC5 are popular yet old and outdated as compared with other new encryption algorithms. We still wanted to do our experimental tests on these two algorithms because of their similar use of Substitution-Permutation Network and Feistel-like structure. As DES uses only 64-bit (56 data bits + 8 parity bits) block as input and 64-bit block as output, we chose RC5-32 to compare with DES because RC5-32 uses 64-bit block as input and 64-bit block as output. Our aim was also to produce complete Distinguishing Attack Profiles for DES and RC5-32 based on our experimental tests. As far as we know, the SAC Tests with/without Autofeeding had only been performed on reduced round TEA by Hernandez, et. al. [48]. We performed these tests on both DES and RC5-32 and also compared the results from these tests with our modified experimental tests.

We had used a maximum of 2^{24} plaintext pairs for each experimental test. This is the limit that we had set due to time constraints because as we double the number of plaintext pairs, the computation time also doubles or triples. With 8 experimental tests (each 10 trials) for each round of the encryption algorithm, this was a very challenging task for us. This was the reason we had included our results for all rounds of each encryption algorithm. Having the Distinguishing Attack Profiles for DES and RC5 would enable us to gauge the strength of these encryption algorithms and make comparisons as to empirically judge whether one encryption algorithm is stronger than the other based on just the output ciphertexts produced by each algorithm. The tests with Autofeeding would enable us to have a high degree of confidence as to how many rounds would be sufficient to make the output ciphertexts indistinguishable from a random permutation.

In the next Section, we summarise the results of the SAC tests we have applied and in

Table 7.1: Distinguishing Attack Profile of DES

DES Rounds	Without Autofeeding				With Autofeeding				Figure No. in this thesis
	SAC	2-bit mod-SAC	4-bit mod-SAC	8-bit mod-SAC	SAC	2-bit mod-SAC	4-bit mod-SAC	8-bit mod-SAC	
1	Fail (2^7)	Fail (2^7)	Fail (2^7)	Fail (2^8)	Fail (2^7)	Fail (2^7)	Fail (2^7)	Fail (2^8)	Figs.B.1-8
2	Fail (2^7)	Fail (2^7)	Fail (2^7)	Fail (2^8)	Fail (2^7)	Fail (2^7)	Fail (2^7)	Fail (2^8)	Figs.B.9-16
3	Fail (2^8)	Fail (2^7)	Fail (2^7)	Fail (2^8)	Fail (2^8)	Fail (2^7)	Fail (2^7)	Fail (2^8)	Figs.B.17-24
4	Fail (2^{10})	Fail (2^7)	Fail (2^7)	Fail (2^8)	Fail (2^{10})	Fail (2^7)	Fail (2^7)	Fail (2^8)	Figs.B.25-32
5	Fail (2^{16})	Fail (2^{13})	Fail (2^{14})	Fail (2^{15})	Fail (2^{16})	Fail (2^{13})	Fail (2^{14})	Fail (2^{15})	Figs.B.33-40
6 to 16	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Figs.B.41-128

the following Section give details of these results for the SAC with/without Autofeeding and modSAC with/without Autofeeding tests performed on different numbers of rounds of DES.

7.4 Summary of Experimental Results on DES

In this Section, we summarise the results that we obtained from the various tests that we had performed for n-round DES, $n = 1 \ldots 16$.

Note that in Table 7.1, the number inside the braces indicates the minimum amount of ciphertext pairs where it begins to fail the specific test. From Table 7.1, we can see that n-round DES failed the experimental tests (either with or without Autofeeding) for $n < 6$. This means that the experimental tests were only able to distinguish between uniform random distribution and the observed distribution for reduced rounds of DES. We compared the results from using the original SAC Test with our 3 different modSAC Tests, *i.e.* 2-bit modSAC, 4-bit modSAC and 8-bit modSAC Tests.

For 1-round DES, we see that the results (with or without Autofeeding) of using the SAC Test is comparable to results of using 2-bit modSAC and 4-bit modSAC Tests but 8-bit modSAC Test requires twice the amount of ciphertext pairs to get the same results as

the SAC Test. For 2-round DES, we obtained similar results as with our tests on 1-round DES. For 3-round DES, 2-bit modSAC and 4-bit modSAC Tests performed better than the SAC and 8-bit modSAC Tests (either with or without Autofeeding). For 4-round DES, 2-bit modSAC and 4-bit modSAC Tests were able to distinguish the observed distribution from uniform random distribution by using only an eighth of the amount of ciphertext pairs used by the SAC Test. Even 8-bit modSAC Test was able to do the same by using only a quarter of the amount of ciphertext used by the SAC Test. For 5-round DES, the SAC Test required at least 2^{16} ciphertext pairs (for either with or without Autofeeding) to distinguish from uniform random distribution. 2-bit modSAC Test was able to do this by just an eighth of the amount of ciphertext pairs used by the SAC Test, while 4-bit modSAC Test required a quarter of the amount of ciphertext used by the SAC Test and 8-bit modSAC Test required half the amount of ciphertext pairs used by SAC Test.

As we can see, 2-bit modSAC Test's performance in making the distinction between our observed distribution and uniform random distribution is similar and in some cases, better than that of the SAC Test. 4-bit modSAC Test's performance in distinguishing from uniform random distribution is similar to 2-bit modSAC Test's performance but slightly poor performance for 4 & 5 rounds of DES. As compared with the SAC Test, 4-bit modSAC Test gave better performance. Even 8-bit modSAC Test's performance was better than the SAC Test in 4 or 5 rounds of DES. For 1 and 2 rounds of DES, the performance of the SAC Test was better than 8-bit modSAC Test. This maybe caused by the fact that the number of ciphertext pairs required by 8-bit modSAC Test were not sufficient to make the distinction between the observed distribution and uniform random distribution.

The tests with Autofeeding try to find out whether a particular encryption algorithm can be used to generate pseudorandom ciphertext output as we feed the ciphertext back into the encryption algorithm to get another ciphertext output. This process continues until we have generated the desired amount of ciphertext outputs. The tests were designed to find out how the input plaintext influences or affects the output ciphertext in terms of measuring its avalanche effect. We see that the tests with Autofeeding gave us the same results as that of the tests without Autofeeding. This tells us that it takes a bit of time for DES to produce the desired ciphertext outputs because we only started to see an increase in the number of ciphertext pairs when the number of rounds for DES is 3 or more. DES with

rounds greater than 5 passed all the tests with and without Autofeeding. For tests without Autofeeding, this means that n-round DES for $n = 6 \ldots 16$, were able to produce ciphertext outputs indistinguishable from random outputs. For tests with Autofeeding, this means that n-round DES for $n = 6 \ldots 16$ can generate ciphertext outputs indinguishable from random outputs.

7.5 Detail of Experimental Results on on DES

In this Section, we describe the experimental setup that was used to perform our various tests on various rounds of DES. Basically, there are four tests that are being performed, namely,

i SAC Test

ii SAC Test with Autofeeding

iii modSAC Test for n-bit substrings, where $n = 2, 4, 8$

iv modSAC Test with Autofeeding for n-bit substrings, where $n = 2, 4, 8$

Input data sets for these tests are plaintexts and keys. The output data sets are ciphertexts. The plaintexts and keys are generated random data. The number of plaintexts generated range from 2^7 to 2^{24}. 10 trials would be performed for each test, *i.e.* each test would be used on 10 different input data sets and we used these data sets for each number of rounds of the cryptographic algorithm. For the case of DES, we started off testing 1 round and gradually increased the number of rounds until finally we tested on the full 16-rounds. The results are displayed graphically according to the number of ciphertext pairs being used (see Appendix B). We subdivided the graphs into four parts according to the number of ciphertexts used. Part (a) ranges from 2^7 to 2^{11} ciphertext pairs. Part (b) is from 2^{12} to 2^{16} ciphertext pairs. Part (c) ranges from 2^{17} to 2^{20} ciphertext pairs and part (d) is from 2^{21} up to 2^{24} ciphertext pairs. We note that for a plaintext P and a key K, we will produce a ciphertext C but we will also produce a ciphertext C' from plaintext P' and and the same key K. The hamming distance between P and P' is 1 and P' is produced by randomly choosing and toggling one bit position of P. In the case of tests with Autofeeding, we generated a random key, K and a random plaintext, P. We

then generate another plaintext P' from P so that the hamming distance between P and P' is 1 and P' is produced by randomly choosing and toggling one bit position of P. P and K would be fed to DES to produce ciphertext C. Similarly, P' and K would be fed to DES to produce ciphertext C'. For Autofeeding, these ciphertexts would become the next plaintext inputs to be fed into DES to produce another set of new ciphertexts. This cycle repeats until the required number of ciphertexts is reached.

7.5.1 Tests on DES

As we had mentioned previously, we performed four tests. We will start by displaying the results from the SAC Test and the modSAC Test for n-bit substrings, where $n = 2, 4, 8$. This is then followed by the results for the SAC Test with Autofeeding and modSAC Test with Autofeeding for n-bit substrings, where $n = 2, 4, 8$.

7.5.1.1 SAC Test

Table 7.2: Experimental Results of SAC Tests on Various Rounds of DES

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	63	0.005	95.649	106.414	128($i.e.$ 2^7)	Fail	Fig.B.1
2	63	0.005	95.649	104.782	128($i.e.$ 2^7)	Fail	Fig.B.9
3	63	0.005	95.649	118.065	256($i.e.$ 2^8)	Fail	Fig.B.17
4	63	0.005	95.649	205.087	1024($i.e.$ 2^{10})	Fail	Fig.B.25
5	63	0.005	95.649	127.569	65536($i.e.$ 2^{16})	Fail	Fig.B.33
6	63	0.005	95.649	57.2137	16777216($i.e.$ 2^{24})	Pass	Fig.B.41
7	63	0.005	95.649	38.6085	16777216($i.e.$ 2^{24})	Pass	Fig.B.49
8	63	0.005	95.649	42.0026	16777216($i.e.$ 2^{24})	Pass	Fig.B.57
9	63	0.005	95.649	38.836	16777216($i.e.$ 2^{24})	Pass	Fig.B.65
10	63	0.005	95.649	39.3879	16777216($i.e.$ 2^{24})	Pass	Fig.B.73
11	63	0.005	95.649	36.4323	16777216($i.e.$ 2^{24})	Pass	Fig.B.81
12	63	0.005	95.649	37.5238	16777216($i.e.$ 2^{24})	Pass	Fig.B.89
13	63	0.005	95.649	33.4853	16777216($i.e.$ 2^{24})	Pass	Fig.B.97
14	63	0.005	95.649	36.4978	16777216($i.e.$ 2^{24})	Pass	Fig.B.105
15	63	0.005	95.649	40.2178	16777216($i.e.$ 2^{24})	Pass	Fig.B.113
16	63	0.005	95.649	37.337	16777216($i.e.$ 2^{24})	Pass	Fig.B.121

As we can see from the table above, DES failed the SAC test for rounds 1 to 5 only. The p-value that we used is 0.005 which is a standard statistical significance level value for χ^2 test. The number of ciphertext pairs needed to distinguish from a random permutation is 128 pairs for 1-Round DES and increases to 65536 for 5-round DES. 6-Round DES and higher round DES, pass the SAC test. 95.649 is the χ^2 test threshold value when the degrees of freedom is $63(64-1)$ and the p-value as stated previously. For 6-Round DES and higher round DES, we did not find the experimental χ^2 test value exceeding or approaching towards this threshold value for 16777216 ciphertext pairs.

7.5.1.2 modSAC Test for 2-bit substring

Table 7.3: Experimental Results of modSAC Tests for 2-bit substring on Various Rounds of DES

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	3	0.005	12.838	9528.44	128($i.e.$ 2^7)	Fail	Fig.B.2
2	3	0.005	12.838	4685.03	128($i.e.$ 2^7)	Fail	Fig.B.10
3	3	0.005	12.838	1111.74	128($i.e.$ 2^7)	Fail	Fig.B.18
4	3	0.005	12.838	62.4072	128($i.e.$ 2^7)	Fail	Fig.B.26
5	3	0.005	12.838	13.084	8192($i.e.$ 2^{13})	Fail	Fig.B.34
6	3	0.005	12.838	8.39007	16777216($i.e.$ 2^{24})	Pass	Fig.B.42
7	3	0.005	12.838	2.98366	16777216($i.e.$ 2^{24})	Pass	Fig.B.50
8	3	0.005	12.838	3.6796	16777216($i.e.$ 2^{24})	Pass	Fig.B.58
9	3	0.005	12.838	4.45962	16777216($i.e.$ 2^{24})	Pass	Fig.B.66
10	3	0.005	12.838	3.21198	16777216($i.e.$ 2^{24})	Pass	Fig.B.74
11	3	0.005	12.838	3.93367	16777216($i.e.$ 2^{24})	Pass	Fig.B.82
12	3	0.005	12.838	2.5603	16777216($i.e.$ 2^{24})	Pass	Fig.B.90
13	3	0.005	12.838	2.87226	16777216($i.e.$ 2^{24})	Pass	Fig.B.98
14	3	0.005	12.838	3.32912	16777216($i.e.$ 2^{24})	Pass	Fig.B.106
15	3	0.005	12.838	3.7441	16777216($i.e.$ 2^{24})	Pass	Fig.B.114
16	3	0.005	12.838	2.77292	16777216($i.e.$ 2^{24})	Pass	Fig.B.122

In the above table, only 1- to 5-Round DES fail the modSAC test for 2-bit substring. Similarly, we use a p-value of 0.005 for the significance level for the χ^2 test. The number of ciphertext pairs needed to distinguish from a random permutation using this test is the

same for 1-Round DES up to and including 4-round DES. Then it suddenly increase to 8192 for 5-round DES. 6-Round DES and higher round DES, pass the modSAC test for 2-bit substring. 12.838 is the χ^2 test threshold value when the degrees of freedom is $3(4-1)$ and the p-value as stated previously. For 6-Round DES and higher round DES, we did not find the experimental χ^2 test value exceeding or approaching towards this threshold value for 16777216 ciphertext pairs.

7.5.1.3 modSAC Test for 4-bit substring

Table 7.4: Experimental Results of modSAC Tests for 4-bit substring on Various Rounds of DES

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	15	0.005	32.801	20714.6	128($i.e.$ 2^7)	Fail	Fig.B.3
2	15	0.005	32.801	7860.59	128($i.e.$ 2^7)	Fail	Fig.B.11
3	15	0.005	32.801	1387.03	128($i.e.$ 2^7)	Fail	Fig.B.19
4	15	0.005	32.801	75.8984	128($i.e.$ 2^7)	Fail	Fig.B.27
5	15	0.005	32.801	33.287	16384($i.e.$ 2^{14})	Fail	Fig.B.35
6	15	0.005	32.801	20.9423	16777216($i.e.$ 2^{24})	Pass	Fig.B.43
7	15	0.005	32.801	15.9754	16777216($i.e.$ 2^{24})	Pass	Fig.B.51
8	15	0.005	32.801	17.1252	16777216($i.e.$ 2^{24})	Pass	Fig.B.59
9	15	0.005	32.801	16.7073	16777216($i.e.$ 2^{24})	Pass	Fig.B.67
10	15	0.005	32.801	13.3849	16777216($i.e.$ 2^{24})	Pass	Fig.B.75
11	15	0.005	32.801	13.7027	16777216($i.e.$ 2^{24})	Pass	Fig.B.83
12	15	0.005	32.801	14.9797	16777216($i.e.$ 2^{24})	Pass	Fig.B.91
13	15	0.005	32.801	16.7239	16777216($i.e.$ 2^{24})	Pass	Fig.B.99
14	15	0.005	32.801	15.8471	16777216($i.e.$ 2^{24})	Pass	Fig.B.107
15	15	0.005	32.801	16.5728	16777216($i.e.$ 2^{24})	Pass	Fig.B.115
16	15	0.005	32.801	13.7929	16777216($i.e.$ 2^{24})	Pass	Fig.B.123

The results that we obtain in the above table are very similar to the results that we get for the modSAC test for 2-bit substring. We can see from the above table that only 1- to 5-round DES fail the modSAC test for 4-bit substring. The number of ciphertext pairs needed to distinguish from a random permutation using this test is the same for 1-Round DES up to and including 4-round DES. Then it suddenly increase to 16384 for 5-round

DES. 6-Round DES and higher round DES, pass the modSAC test for 4-bit substring. 32.801 is the χ^2 test threshold value when the degrees of freedom is $15(16-1)$ and the p-value as stated previously. For 6-Round DES and higher round DES, we did not find the experimental χ^2 test value exceeding or approaching towards this threshold value for 16777216 ciphertext pairs. Thus these higher round DES pass our modSAC test for 4-bit substring.

7.5.1.4 modSAC Test for 8-bit substring

Table 7.5: Experimental Results of modSAC Tests for 8-bit substring on Various Rounds of DES

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	255	0.005	316.919	262303	256($i.e.$ 2^8)	Fail	Fig.B.4
2	255	0.005	316.919	61001.2	256($i.e.$ 2^8)	Fail	Fig.B.12
3	255	0.005	316.919	5563.32	256($i.e.$ 2^8)	Fail	Fig.B.20
4	255	0.005	316.919	469.775	256($i.e.$ 2^8)	Fail	Fig.B.28
5	255	0.005	316.919	322.231	32768($i.e.$ 2^{15})	Fail	Fig.B.36
6	255	0.005	316.919	268.857	16777216($i.e.$ 2^{24})	Pass	Fig.B.44
7	255	0.005	316.919	259.604	16777216($i.e.$ 2^{24})	Pass	Fig.B.52
8	255	0.005	316.919	253.888	16777216($i.e.$ 2^{24})	Pass	Fig.B.60
9	255	0.005	316.919	250.482	16777216($i.e.$ 2^{24})	Pass	Fig.B.68
10	255	0.005	316.919	242.394	16777216($i.e.$ 2^{24})	Pass	Fig.B.76
11	255	0.005	316.919	255.995	16777216($i.e.$ 2^{24})	Pass	Fig.B.84
12	255	0.005	316.919	258.378	16777216($i.e.$ 2^{24})	Pass	Fig.B.92
13	255	0.005	316.919	251.967	16777216($i.e.$ 2^{24})	Pass	Fig.B.100
14	255	0.005	316.919	258.105	16777216($i.e.$ 2^{24})	Pass	Fig.B.108
15	255	0.005	316.919	255.948	16777216($i.e.$ 2^{24})	Pass	Fig.B.116
16	255	0.005	316.919	257.17	16777216($i.e.$ 2^{24})	Pass	Fig.B.124

From the table above, we see that 1-Round DES fail the modSAC test for 8-bit substring. The number of ciphertext pairs sufficient for distinguishing 1-Round DES from a random permutation is 256 pairs. 256 pairs of ciphertexts is also sufficient for distinguishing 2-Round DES, 3-Round DES and 4-Round DES, from a random permutation. For 5-Round DES, the number of ciphertext pairs needed to make this distinction is at least

32768. 316.919 is the χ^2 test threshold value when the degrees of freedom is $255(256-1)$ with a p-value of 0.005. For 6-Round DES and higher round DES, we did not find the experimental χ^2 test value exceeding or approaching towards this threshold value for 16777216 ciphertext pairs. Thus these higher round DES pass our modSAC test for 8-bit substring.

7.5.1.5 SAC Test with Autofeeding

Table 7.6: Experimental Results of SAC Tests on Various Rounds of DES (Autofeeding)

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	63	0.005	95.649	106.414	128($i.e.$ 2^7)	Fail	Fig.B.5
2	63	0.005	95.649	103.945	128($i.e.$ 2^7)	Fail	Fig.B.13
3	63	0.005	95.649	117.894	256($i.e.$ 2^8)	Fail	Fig.B.21
4	63	0.005	95.649	173.923	1024($i.e.$ 2^{10})	Fail	Fig.B.29
5	63	0.005	95.649	124.843	65536($i.e.$ 2^{16})	Fail	Fig.B.37
6	63	0.005	95.649	60.1454	16777216($i.e.$ 2^{24})	Pass	Fig.B.45
7	63	0.005	95.649	36.197	16777216($i.e.$ 2^{24})	Pass	Fig.B.53
8	63	0.005	95.649	38.8928	16777216($i.e.$ 2^{24})	Pass	Fig.B.61
9	63	0.005	95.649	42.1217	16777216($i.e.$ 2^{24})	Pass	Fig.B.69
10	63	0.005	95.649	41.3929	16777216($i.e.$ 2^{24})	Pass	Fig.B.77
11	63	0.005	95.649	38.821	16777216($i.e.$ 2^{24})	Pass	Fig.B.85
12	63	0.005	95.649	40.3234	16777216($i.e.$ 2^{24})	Pass	Fig.B.93
13	63	0.005	95.649	40.9324	16777216($i.e.$ 2^{24})	Pass	Fig.B.101
14	63	0.005	95.649	39.6288	16777216($i.e.$ 2^{24})	Pass	Fig.B.109
15	63	0.005	95.649	40.2927	16777216($i.e.$ 2^{24})	Pass	Fig.B.117
16	63	0.005	95.649	40.9511	16777216($i.e.$ 2^{24})	Pass	Fig.B.125

When we perform the SAC Test with Autofeeding, we do not see any difference in the results as compared to the SAC test results in Table 7.2 in terms of the number of required ciphertext pairs. However, the experimental χ^2 values in the above table are different from those obtained in Table 7.2 with the exception of 1-Round DES. Only 1- to 5-round DES fail the SAC test with Autofeeding. 6-round DES and higher round DES pass the SAC test with Autofeeding. This means that these higher round DES are able to generate ciphertext outputs indistinguishable from random outputs.

7.5.1.6 modSAC Test with Autofeeding for 2-bit substring

Table 7.7: Experimental Results of modSAC Tests for 2-bit substring on Various Rounds of DES (Autofeeding)

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	3	0.005	12.838	9533.39	128($i.e.$ 2^7)	Fail	Fig.B.6
2	3	0.005	12.838	4708.12	128($i.e.$ 2^7)	Fail	Fig.B.14
3	3	0.005	12.838	1110.32	128($i.e.$ 2^7)	Fail	Fig.B.22
4	3	0.005	12.838	72.8266	128($i.e.$ 2^7)	Fail	Fig.B.30
5	3	0.005	12.838	15.3479	8192($i.e.$ 2^{13})	Fail	Fig.B.38
6	3	0.005	12.838	8.46687	16777216($i.e.$ 2^{24})	Pass	Fig.B.46
7	3	0.005	12.838	2.13801	16777216($i.e.$ 2^{24})	Pass	Fig.B.54
8	3	0.005	12.838	4.41917	16777216($i.e.$ 2^{24})	Pass	Fig.B.62
9	3	0.005	12.838	2.1929	16777216($i.e.$ 2^{24})	Pass	Fig.B.70
10	3	0.005	12.838	2.03053	16777216($i.e.$ 2^{24})	Pass	Fig.B.78
11	3	0.005	12.838	3.01876	16777216($i.e.$ 2^{24})	Pass	Fig.B.86
12	3	0.005	12.838	2.95697	16777216($i.e.$ 2^{24})	Pass	Fig.B.94
13	3	0.005	12.838	3.30985	16777216($i.e.$ 2^{24})	Pass	Fig.B.102
14	3	0.005	12.838	2.34561	16777216($i.e.$ 2^{24})	Pass	Fig.B.110
15	3	0.005	12.838	2.28238	16777216($i.e.$ 2^{24})	Pass	Fig.B.118
16	3	0.005	12.838	2.25921	16777216($i.e.$ 2^{24})	Pass	Fig.B.126

As we can see from the table above, the number of ciphertext pairs required to distinguish generated outputs from 1-Round DES from random outputs is 128 pairs. The same number of ciphertext pairs is also sufficient for distinguishing generated outputs from 2-Round DES, 3-Round DES and 4-Round DES from random outputs. For 5-Round DES, the number is slightly higher, that is, 8192 pairs. This means that 1- to 5-Round DES fail the modSAC test for 2-bit substring. However, 6-Round DES and higher round DES pass this test. This means that these higher round DES are able to generate ciphertext outputs indistinguishable from random outputs.

7.5.1.7 modSAC Test with Autofeeding for 4-bit substring

Table 7.8: Experimental Results of modSAC Tests for 4-bit substring on Various Rounds of DES (Autofeeding)

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	15	0.005	32.801	20803.8	128($i.e.$ 2^7)	Fail	Fig.B.7
2	15	0.005	32.801	7992	128($i.e.$ 2^7)	Fail	Fig.B.15
3	15	0.005	32.801	1415.82	128($i.e.$ 2^7)	Fail	Fig.B.23
4	15	0.005	32.801	91.0266	128($i.e.$ 2^7)	Fail	Fig.B.31
5	15	0.005	32.801	40.8635	16384($i.e.$ 2^{14})	Fail	Fig.B.39
6	15	0.005	32.801	21.5484	16777216($i.e.$ 2^{24})	Pass	Fig.B.47
7	15	0.005	32.801	14.9609	16777216($i.e.$ 2^{24})	Pass	Fig.B.55
8	15	0.005	32.801	17.1044	16777216($i.e.$ 2^{24})	Pass	Fig.B.63
9	15	0.005	32.801	15.636	16777216($i.e.$ 2^{24})	Pass	Fig.B.71
10	15	0.005	32.801	14.1291	16777216($i.e.$ 2^{24})	Pass	Fig.B.79
11	15	0.005	32.801	14.9727	16777216($i.e.$ 2^{24})	Pass	Fig.B.87
12	15	0.005	32.801	15.5705	16777216($i.e.$ 2^{24})	Pass	Fig.B.95
13	15	0.005	32.801	13.5526	16777216($i.e.$ 2^{24})	Pass	Fig.B.103
14	15	0.005	32.801	15.7968	16777216($i.e.$ 2^{24})	Pass	Fig.B.111
15	15	0.005	32.801	13.5228	16777216($i.e.$ 2^{24})	Pass	Fig.B.119
16	15	0.005	32.801	14.5736	16777216($i.e.$ 2^{24})	Pass	Fig.B.127

As given in Table 7.8 above, for 1- to 4-Round DES, we have number of ciphertext pairs required to distinguish generated outputs of these reduced round DES from random outputs is 128 pairs. 5-Round DES requires 16384 ciphertext pairs to make the same distinction. Thus, 1- to 5-Round DES fail the modSAC test for 4-bit substring with Autofeeding. 6-Round DES and higher round DES pass this test.

7.5.1.8 modSAC Test with Autofeeding for 8-bit substring

Table 7.9: Experimental Results of modSAC Tests for 8-bit substring on Various Rounds of DES (Autofeeding)

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	255	0.005	316.919	264916	256($i.e.$ 2^8)	Fail	Fig.B.8
2	255	0.005	316.919	61335.7	256($i.e.$ 2^8)	Fail	Fig.B.16
3	255	0.005	316.919	5590.48	256($i.e.$ 2^8)	Fail	Fig.B.24
4	255	0.005	316.919	435.25	256($i.e.$ 2^8)	Fail	Fig.B.32
5	255	0.005	316.919	319.34	32768($i.e.$ 2^{15})	Fail	Fig.B.40
6	255	0.005	316.919	255.665	16777216($i.e.$ 2^{24})	Pass	Fig.B.48
7	255	0.005	316.919	262.091	16777216($i.e.$ 2^{24})	Pass	Fig.B.56
8	255	0.005	316.919	257.935	16777216($i.e.$ 2^{24})	Pass	Fig.B.64
9	255	0.005	316.919	262.077	16777216($i.e.$ 2^{24})	Pass	Fig.B.72
10	255	0.005	316.919	250.389	16777216($i.e.$ 2^{24})	Pass	Fig.B.80
11	255	0.005	316.919	268.339	16777216($i.e.$ 2^{24})	Pass	Fig.B.88
12	255	0.005	316.919	250.452	16777216($i.e.$ 2^{24})	Pass	Fig.B.96
13	255	0.005	316.919	254.044	16777216($i.e.$ 2^{24})	Pass	Fig.B.104
14	255	0.005	316.919	242.371	16777216($i.e.$ 2^{24})	Pass	Fig.B.112
15	255	0.005	316.919	248.262	16777216($i.e.$ 2^{24})	Pass	Fig.B.120
16	255	0.005	316.919	251.998	16777216($i.e.$ 2^{24})	Pass	Fig.B.128

The results that we obtained in the above table is similar to those results given in the previous tables. However, the number of required ciphertext pairs is slightly higher than those in the previous tables. 1- to 4-Round DES require 256 pairs of ciphertext to distiguish generated outputs of these reduced round DES from random outputs. 5-Round DES requires 32768 ciphertext pairs to make the same distinction. Thus, 1- to 5-Round DES fail the modSAC test for 8-bit substring with Autofeeding. 6-Round DES and higher round DES pass this test.

Chapter 8

LINEAR CRYPTANALYSIS OF RC5

In order to do linear cryptanalysis of RC5, we need to look at how we can find linear approximations for a half-round for RC5.

8.1 Linear approximations for a half-round

According to Kaliski and Yin in [55], finding linear approximations for a half-round of RC5 involves the following equation:

$$R_i = ((L_{i-1} \oplus R_{i-1}) \lll R_{i-1}) + S_i \tag{8.1.1}$$

We need to decompose the equation (8.1.1) into three equations, each of which involves only a single primitive operation. We can then consider possible linear approximations for each of these equations. Equation (8.1.1) can be rewritten as the following three equations :

$$X = L_{i-1} \oplus R_{i-1} \tag{8.1.2}$$

$$Y = X \lll R_{i-1} \tag{8.1.3}$$

$$R_i = Y + S_i \tag{8.1.4}$$

For equation (8.1.2), we see that there are numerous linear approximations holding with bias 1/2 (*i.e.* having probability 1 or 0). In particular, all approximations involving the same bits of X, L_{i-1} and R_{i-1}. All other approximations have bias 0.

The linear approximations for equation (8.1.3) can be divided into two types depending on whether bits of R_{i-1} are involved. First, we consider approximations in which no bits of R_{i-1} are involved. Any such approximation involving just one bit of X and Y holds with probability $1/2 + 1/2w$, since for one rotation amount, the bits are guaranteed to be equal and for the other $w-1$ amounts, the bits are equal with probability 1/2. Next, we consider approximations where some bits of R_{i-1} are involved. Some of these approximations have non-zero bias. For example,

$$Y[0] = X[0] \oplus R_{i-1}[0] \tag{8.1.5}$$

holds with probability $1/2 + 1/2w$, since when the rotation amount is zero, $R_{i-1}[0] = 0$ and both (8.1.3) and (8.1.5) yield $Y[0] = X[0]$, and when the amount is otherwise, the equation (8.1.5) holds with probability 1/2. Note that a linear approximation will have bias zero if any bit $R_{i-1}[s]$ where $s \geq log_2 w$ is involved.

For equation (8.1.4), the best linear approximation is the following:

$$R_i[0] = Y[0] + S_i[0] \tag{8.1.6}$$

which holds with probability 1. All other approximations do not have bias 1/2 and their biases are dependent on the key.

Since in the first half-round of RC5 we have only the + operation, then we have both approximations

$$L_1[0] = L_0[0] \oplus S_0[0] \quad \text{and} \quad R_1[0] = R_0[0] \oplus S_1[0]$$

hold with probability 1. We will denote these as **C** and **D**, respectively.

Based on our discussion above, we can now construct many possible linear approximations for a half-round of RC5 by joining the approximations for the three operations involved. For example, by joining $X[0] = L_{i-1}[0] \oplus R_{i-1}[0]$, approximation (8.1.5) and

approximation (8.1.6), we obtain the following approximation for a half-round:

$$R_i[0] = L_{i-1}[0] \oplus S_i[0]$$

which holds with probability $1/2 + 1/2w$. We will denote it as **E**.

Since $L_i = R_{i-1}$ in a half-round of RC5, there are many approximations which involve the same bits of L_i and R_{i-1} and hold with probability 1. The trivial approximation

$$L_i[0] = R_{i-1}[0]$$

which we will denote as **-**, can be alternated with approximation **E**.

Once we have determined the possible linear approximations for a half-round of RC5 then we can do linear cryptanalysis of RC5.

8.2 Linear Cryptanalysis of RC5

The basic idea of linear cryptanalysis is to find expressions consisting of combinations of bits of the plaintext, the ciphertext and key bits which holds with probability $p \neq 1/2$. With the assumption that each round in the cipher is independent, Kaliski and Yin [55] determine the most likely linear approximation by concatenating the half-round linear approximations **C**,**D**,**-** and **E**. The resulting (r-1) round approximation $L_{2r}[0] \oplus R_0[0] = S_1[0] \oplus S_3[0] \oplus \ldots \oplus S_{2r-1}[0]$ can then be used to attack the r-round cipher. The bias of $L_{2r} \oplus R_0$ can be found using the Piling Up Lemma [74]. In [55], it is stated that the bias ϵ_{r-1} is equal to $1/(2w^{r-1})$.

In recovering one bit of S_{2r+1} *i.e.* the final subkey used in RC5 encryption, Kaliski and Yin stated that the number of plaintexts, N, required is estimated to be the inverse square of the bias ϵ_{r-1}. Since we have w bits in the final subkey S_{2r+1}, then the number of known plaintexts required is given by the following:

$$N = w \cdot \epsilon_{r-1}^2 \tag{8.2.1}$$

Since we can repeatedly use the same plaintexts, Kaliski and Yin stated that we can recover the other subkeys without requiring any additional plaintexts. For RC5 r-round

cipher with nominal parameters *i.e.* $w = 32$ and $b = 2r + 2$, linear cryptanalysis would require 2^{37} plaintexts for $r = 4$, 2^{47} plaintexts for $r = 5$ and 2^{57} plaintexts for $r = 6$.

Kaliski and Yin [55] had also outlined the algorithm to perform linear cryptanalysis of RC5. For convenience, we present the algorithm again.

The algorithm is as follows:

1. Compute $S_{2r+1}[0]$. We observe that for plaintext/ciphertext pairs such that L_{2r+1} $mod\, w = 1$, one of the following two approximations is perfect *i.e.* either having bias $1/2$ or $-1/2$:

$$\begin{aligned} L_{2r}[0] &= L_{2r+1}[0] \oplus R_{2r+1}[1] && \text{if } S_{2r+1}[0] = 0 \\ L_{2r}[0] &= L_{2r+1}[0] \oplus R_{2r+1}[1] \oplus R_{2r+1}[0] && \text{if } S_{2r+1}[0] = 1 \end{aligned}$$

   The first approximation has zero bias if $S_{2r+1}[0] = 1$ and the second approximation has zero bias if $S_{2r+1}[0] = 0$. To compute $S_{2r+1}[0]$, we obtain N known plaintext/ciphertext pairs such that $L_{2r+1} \bmod w = 1$ and consider the two quantities $R_0[0] \oplus (L_{2r+1}[0] \oplus R_{2r+1}[1])$ and $R_0[0] \oplus (L_{2r+1}[0] \oplus R_{2r+1}[1] \oplus R_{2r+1}[0])$. Let U_0 be the number of plaintexts such that the first quantity is zero and U_1 be the number of plaintexts such that the second quantity is zero. If $|U_0 - N/2| \geq |U_1 - N/2|$, we predict $S_{2r+1}[0] = 0$; otherwise, we predict $S_{2r+1}[0] = 1$.

2. Compute T, *i.e.* the value of $S_1[0] \oplus S_3 \oplus \cdots \S_{2r-1}[0]$, given $S_{2r+1}[0]$. We observe that for plaintext/ciphertext pairs such that $L_{2r+1} \bmod w = 0$, the approximation

$$L_{2r}[0] = L_{2r+1}[0] \oplus R_{2r+1}[0] \oplus S_{2r+1}[0]$$

   holds with probability 1 and the right-hand side is known. To compute T, we obtain N known plaintext/ciphertext pairs such that $L_{2r+1} \bmod w = 0$. Let U be the number of plaintexts such that

$$R_0[0] \oplus (L_{2r+1}[0] \oplus R_{2r+1}[0] \oplus S_{2r+1}[0])$$

   is zero. If $U \geq N/2$, we predict $T = 0$; otherwise, we predict $T = 1$.

3. For $s = 1, \ldots, w - 1$, compute $S_{2r+1}[s]$ given T and $S_{2r+1}[s - 1 \ldots 0]$. For a given

plaintext/ciphertext pair, let $y = R_{2r+1} - S_{2r+1}$ and let $carry(s)$ denote the carry out from $y[s-1\ldots0] + S_{2r+1}[s-1\ldots0]$. We observe that for plaintext/ciphertext pairs such that $L_{2r+1} \bmod w = s$, the approximation

$$L_{2r}[0] = L_{2r+1}[0] \oplus R_{2r+1}[s] \oplus S_{2r+1}[s] \oplus carry(s)$$

holds with probability 1. To compute $S_{2r+1}[s]$, we obtain N known plaintext/ciphertext pairs such that $L_{2r+1} \bmod w = s$. Let U be the number such that

$$(R_0[0] \oplus T) \oplus (L_{2r+1}[0] \oplus R_{2r+1}[s] \oplus carry(s))$$

is zero. If $U \geq N/2$, we predict $S_{2r+1}[s] = 0$; otherwise, we predict $S_{2r+1}[s] = 1$.

8.3 Further Results on Linear Cryptanalysis of RC5

Selçuk [91] had pointed out that if we had used the linear cryptanalysis method by Kaliski and Yin [55] we described earlier, we would not be able to get the expected success rate (*i.e.* 95-99%) and surprisingly the success rate did not improve when the amount of data used was increased. In [91], Selçuk suggested that there were some hidden assumptions which caused the high success rate which led him to develop different linear cryptanalytic attacks on RC5 to recover the round key S_{2r+1}. The linear approximation $R_0[0] \oplus L_{2r}[0] = S_1[0] \oplus S_3[0] \oplus \cdots \oplus S_{2r-1}[0]$ (used by Kaliski and Yin) was modified so that the actual value of $L_{2r}[0]$ was used. The actual value of $L_{2r}[0]$ refers to $(R_{2r+1} - S_{2r+1})[\rho] \oplus L_{2r+1}$ where ρ denotes $L_{2r+1} \bmod w$. So the approximation becomes $R_0[0] \oplus (R_{2r+1} - S_{2r+1})[\rho] \oplus L_{2r+1} = S_1[0] \oplus S_3[0] \oplus \cdots \oplus S_{2r-1}[0]$. Note that, when we substitute a wrong value s for S_{2r+1} in this approximation, the bias is not zero. It was also explained that if we were to change one bit in the round key S_{2r+1}, the bias would also change in the opposite direction. This makes it harder to recover the round key since we would require large amounts of plaintext/ciphertext pairs to obtain a high success rate. We will not explain the method by Selçuk any further because we had found another linear cryptanalysis method (introduced by Borst, Preneel and Vandewalle in [27]) which is better at recovering the round key of RC5. This is the method that we will be discussing next.

As with the linear cryptanalysis method used by Kaliski and Yin, we need to firstly

rewrite RC5 algorithm as a series of equations which involve only primitive operations as follows:

$$L_1 = L_0 + S_0 \tag{8.3.1}$$

$$R_1 = R_0 + S_1 \tag{8.3.2}$$

$$U_i = L_i \oplus R_i \tag{8.3.3}$$

$$V_i = U_i \lll R_i \tag{8.3.4}$$

$$R_{i+1} = V_i + S_{i+1} \tag{8.3.5}$$

$$L_{i+1} = R_i \tag{8.3.6}$$

for $i = 1, \ldots, 2r$. We then find the linear approximations for XOR, data-dependent rotation and addition for the above equations. In order to do that, we look at one bit of each term of the equation. If $A = B \oplus C$, then we have w approximations

$$A[i] = B[i] \oplus C[i], \quad \delta = 2^{-1} \tag{8.3.7}$$

for $i \in \{0, \ldots, w-1\}$. δ is called the deviation and $|\delta| = \epsilon$ refers to the bias of the linear approximation. If $D = E \lll F$, then

$$D[i] = E[j] \oplus F[k] \oplus (i-j)[k], \quad \delta = 2^{-log_2 w - 1} \tag{8.3.8}$$

for $i, j \in \{0, \ldots, w-1\}$ and $k \in \{0, \ldots, log_2 w - 1\}$. Using equations (8.3.7) and (8.3.8) to pass through the XOR and rotation in RC5 we get the following equations:

$$U_i[j] = L_i[j] \oplus R_i[j], \quad \delta = 2^{-1} \tag{8.3.9}$$

$$V_i[k] = U_i[j] \oplus R_i[j] \oplus (k-j)[j], \quad \delta = 2^{-log_2 w - 1} \tag{8.3.10}$$

Chaining equations (8.3.9) and (8.3.10), we get the following:

$$V_i[k] = L_i[j] \oplus (k-j)[j], \quad \delta = 2^{-log_2 w - 1} \tag{8.3.11}$$

Finally, if we have $G = H + S$ where S is fixed, then

$$G[0] = H[0] \oplus S[0], \quad \delta = 2^{-1} \tag{8.3.12}$$

$$G[i] = H[i] \oplus S[i], \quad \delta = 2^{-1} - 2^{-i}S^i \tag{8.3.13}$$

for $i \in \{0, \ldots, w-1\}$. Here $S^x = S \bmod 2^{x+1}$ *i.e.* the x least significant bits (LSB) of S. So, depending on the round key S, the bias of (8.3.13) can vary between 0 and 1/2. On average, the bias is 1/4.

Using approximations (8.3.7), (8.3.8) and (8.3.12) we can derive the following iterative approximation for one round of RC5:

$$L_i[0] \oplus S_{i+1}[0] = L_{i+2}[0], \quad (i \geq 1), \quad \delta = 2^{-log_2 w - 1} \tag{8.3.14}$$

This approximation can be chained to l rounds as follows:

$$L_i[0] \oplus \bigoplus_{j=0}^{l-1} S_{i+1+2j} = L_{i+2l}[0], \quad (i \geq 1) \tag{8.3.15}$$

According to the Piling Up Lemma [74, 75], this approximation would have deviation $\delta = 2^{l-1}2^{(-log_2 w-1)l} = 2^{-l\, log_2 w - 1}$ if the chained approximations are independent. According to Borst, Preneel and Vandewalle [27], the Piling Up Lemma gives a good estimate for the average bias for RC5.

The linear cryptanalysis used in [27] is different from the one used previously since it uses linear hulls. A linear hull is the set of chains of linear equations over (a part of) the cipher that produce the same linear equation. The following describes a linear hull for an approximation of two rounds (from i to $i+3$) of RC5 given in (8.3.20). Using (8.3.7), (8.3.8), (8.3.12) and (8.3.13), the following collections of $log_2 w$ approximations for two rounds can be derived. For $j \in \{0, \ldots, w-1\}$,

$$L_i[j] \oplus (k-j)[j] \oplus S_{i+1}[k] = L_{i+2}[k], \quad \delta = \delta_{j,k} \tag{8.3.16}$$

for $k = 0, \ldots, log_2 w - 1$. Similarly, for the next two rounds also $log_2 w$ approximations can

be derived. For $l \in \{0, \ldots, w-1\}$,

$$L_{i+2}[k] \oplus (l-k)[k] \oplus S_{i+3}[l] = L_{i+4}[l], \quad \delta = \delta_{k,l} \tag{8.3.17}$$

for $k = 0, \ldots, log_2 w - 1$. Therefore, we can chain the $log_2 w$ pairs of (8.3.16) and (8.3.17) to obtain the two round approximation

$$L_i[j] \oplus S_{i+1}[k] \oplus S_{i+3}[l] \oplus (k-j)[j] \oplus (l-k)[k] = L_{i+4}[l], \quad \delta = \delta_{j,k,l} \tag{8.3.18}$$

Using the Piling Up Lemma, we get (for the bias)

$$\delta_{j,k,l} = \begin{cases} 2^{-2log_2 w-1} & k = 0, l = 0 \\ 2^{-2log_2 w-1}(1 - 2^{-k+1} S_{i+1}^{k-1}) & k \neq 0, l = 0 \\ 2^{-2log_2 w-1}(1 - 2^{-l+1} S_{i+3}^{l-1}) & k = 0, l \neq 0 \\ 2^{-2log_2 w-1}(1 - 2^{-k+1} S_{i+1}^{k-1})(1 - 2^{-l+1} S_{i+3}^{l-1}) & k \neq 0, l \neq 0 \end{cases} \tag{8.3.19}$$

It is clear that from (8.3.19) that the deviation is key dependent *i.e.* dependent on the $log_2 w - 1$ LSB's of S_{i+1} and S_{i+3} but due to the linear hull effect, this key dependency is negligible. Note that for each of the triples j, k, l the term $C_{j,k,l} = S_{i+1}[k] \oplus S_{i+3}[l] \oplus (k-j)[j] \oplus (l-k)[k]$ is constant, either 0 or 1. This constant actually determines the sign of the deviation of the following approximation, which can be derived from (8.3.18) by leaving out $C_{j,k,l}$.

$$L_i[j] = L_{i+4}[l], \quad \delta = \tilde{\delta}_{j,l} \tag{8.3.20}$$

For the deviation, the following holds:

$$\tilde{\delta}_{j,l} = \sum_{k \in V_{j,l}} \delta_{j,k,l} - \sum_{k \notin V_{j,l}} \delta_{j,k,l}, \tag{8.3.21}$$

where $V_{j,l} = \{k \in 0, \ldots, log_2 w - 1 | C_{j,k,l} = 0\}$

We can extend approximation (8.3.20) to hold for r subsequent rounds. In this way we get the following approximation for r rounds.

$$L_i[j] = L_{i+2r}[l], \quad \delta = \tilde{\delta}_{j,l}, \tag{8.3.22}$$

where $i, j, l \in \{0, \ldots, w-1\}$.

We have established equations in order for us to use the linear attack used in [27]. In the next Section, we will explain this linear attack and will give some experimental results that we get from performing the attack on reduced round RC5.

8.4 Experimental Results on Linear Cryptanalysis of RC5

The linear attack that we have implemented uses the approximation (8.3.22). To attack r rounds of RC5 we use the fact that each linear path that is part of the hull given by (8.3.22) is a chain of r 1-round approximations. We split the approximation into two parts. The first contains key addition and the first round. This gives us the following $log_2 w$ approximations. Each is the first part of a set of linear approximations that is contained in the linear hull of (8.3.22).

$$L_0[0] = L_3[k] \tag{8.4.1}$$

for $k = 0, \ldots, log_2 w - 1$. The remainder of the whole approximation can be specified by the following $log_2 w$ approximations, each beginning with a different bit of L_3:

$$L_3[k] = L_{2r+1}[0] \tag{8.4.2}$$

for $k = 0, \ldots, log_2 w - 1$. When for a certain plaintext encryption the intermediate value $R_1 \bmod w = k$ where $k \in \{0, \ldots, log_2 w - 1\}$, then (8.4.1) behaves in the deviation direction. Hence, if (8.4.2) also behaves in the deviation direction then the whole approximation behaves in that direction. On the other hand, if $R_1 \bmod w \notin \{0, \ldots, log_2 w - 1\}$ then the probability that the whole approximation behaves in the deviation direction is much lower. In the linear attack, we want to check every text to determine whether one of the approximations that correspond to (8.4.2) was followed. Since we have no information about any intermediate values, we do not have a criterion that always holds. Instead a function is derived which is expected to give higher values when one of the approximations was followed. Hence this function will have higher values for encryptions where $R_1 \bmod w \in \{0, 1, \ldots, log_2 w - 1\}$ than for other R_1 values. As $R_1 \bmod w = (R_0 - S_1) \bmod w$ and R_0 is known we can guess $S_1 \bmod w$ from this. This function is called the non-uniformity

function.

The non-uniformity function ν computes non-uniformity values for a given set of corresponding plaintext/ciphertext pairs. This set is divided into w subsets, each set contains plaintexts with the same $R_0 \bmod w$-value. For each set a non-uniformity value can be computed. If we were to know the value of $R_{2r-1} \bmod w$ it would be possible to compute $log_2 w$ bits of S_{2r+1} or two possible values for those bits from the ciphertexts using the following:

$$S_{2r+1} = R_{2r+1} - (R_{2r-1} \oplus L_{2r+1}) \lll L_{2r+1}, \tag{8.4.3}$$

since the values of R_{2r+1} and L_{2r+1} are known from the ciphertexts. We will use $S\langle n \rangle$ to denote the $log_2 w$-bit string, given by the bits $(n+log_2 w-1) \bmod w, \ldots, (n+1) \bmod w, n$ of S. The value of $L_{2r+1} \bmod w$ determines for which $log_2 w$ bits of S_{2r+1} can be computed, *i.e.* $S\langle L_{2r+1} \bmod w \rangle$. If $L_{2r+1} \bmod w = 0$ then the $log_2 w$ LSB's can be computed. When $L_{2r+1} \bmod w \neq 0$ we can compute two values for $S\langle L_{2r+1} \bmod w \rangle$. Since we do not know the value of $R_{2r-1} \bmod w$ or even which value would be the most probable, we make a similar computation for a value which we denote as S' instead of trying to compute $log_2 w$ bits of S_{2r+1}. This value is computed by taking $R_{2r-1} \bmod w = 0$ in (8.4.3) so we have

$$S' = R_{2r+1} - (L_{2r+1} \lll L_{2r+1}). \tag{8.4.4}$$

Due to the non-uniform distribution of $R_{2r-1} \bmod w$ itis expected that the distribution of $S'\langle \cdot \rangle$-values will be more non-uniform for encryptions where the approximation was followed than for others. So, for the attack we use a counter array $A(i,j,k)$ for $i,j,k = 0, \ldots, w-1$, where each i corresponds to a possible value of $R_0 \bmod w$, each j corresponds to a possible value of $L_{2r+1} \bmod w$ and each k corresponds to a possible value of $S'\langle L_{2r+1} \bmod w \rangle$. For each text we check if the approximation holds. If it holds, we change the counter array as follows. If $L_{2r+1} \bmod w = 0$, we increase $A(R_0 \bmod w, L_{2r+1} \bmod w, v)$ by 2, where v is the suggested $S'\langle L_{2r+1} \bmod w \rangle$-value. If $L_{2r+1} \bmod w \neq 0$, we increase $A(R_0 \bmod w, L_{2r+1} \bmod w, v_0)$ and $A(R_0 \bmod w, L_{2r+1} \bmod w, v_1)$ by 1, where v_0 and v_1 are the suggested values. If the approximation does not hold, we decrease the specified array entries accordingly. Each $(R_0 \bmod w, L_{2r+1} \bmod w)$-combination gives a distribution of $S'\langle \cdot \rangle$-values. It was discovered in [27] that values of $R_1 \bmod w \in \{0, \ldots, log_2 w - 1\}$ gives the most non-uniform distributions. In order to measure the non-uniformity we check for

all w bits of S' based on the $S'\langle\cdot\rangle$-values, we count the number of times 0 is suggested and also the number of times 1 is suggested and take the difference of these two amounts. For each $R_0 \bmod w$ we take the sum over all possible $L_{2r+1} \bmod w$ of the absolute values of these differences. The non-uniformity function $\nu : \{0, \ldots, w-1\} \to \mathbb{N}$ is defined as follows:

$$\nu(r_0) = \sum_{l_{2r+1}=0}^{w-1} \sum_{x=0}^{log_2 w - 1} | \sum_{v:v[x]=0} A(r_0, l_{2r+1} + x, v) - \sum_{v:v[x]=1} A(r_0, l_{2r+1} + x, v)| \qquad (8.4.5)$$

where all indices of A are taken modulo w. The w values derived in this way are called the non-uniformity values. It is expected that the sum of $log_2 w$ non-uniformity values will be maximal for the values corresponding to texts with $R_1 \bmod w \in \{0, \ldots, log_2 w - 1\}$. Finally, we guess the value of $S_1 \bmod w$ accordingly. Below is a summary of the algorithm for performing the linear attack on RC5.

1. Acquire n known plaintext/ciphertext pairs $(P_0, C_0), \ldots, (P_{n-1}, C_{n-1})$
2. Initialize a counter array $A(i, j, k) \leftarrow 0$ for $i, j, k = 0, \ldots, w-1$
3. Fore each plaintext/ciphertext pair do:
   - If $L_{2r+1} \bmod w = 0$ then
     (a) Compute $S'\langle 0\rangle$-guess v
     (b) If $L_0[0] = L_{2r+1}[0]$ then $A(R_0, L_{2r+1}, v) \leftarrow A(R_0, L_{2r+1}, v) + 2$.
     If $L_0[0] = L_{2r+1}[0] \oplus 1$ then $A(R_0, L_{2r+1}, v) \leftarrow A(R_0, L_{2r+1}, v) - 2$.
   - If $L_{2r+1} \bmod w \neq 0$ then
     (a) Compute $S'\langle L_{2r+1} \bmod w\rangle$-guesses v_0 and v_1.
     (b) If $L_0[0] = L_{2r+1}[0]$ then $A(R_0, L_{2r+1}, v_0) \leftarrow A(R_0, L_{2r+1}, v_0) + 1$ and $A(R_0, L_{2r+1}, v_1) \leftarrow A(R_0, L_{2r+1}, v_1) + 1$
     If $L_0[0] \neq L_{2r+1}[0]$ then $A(R_0, L_{2r+1}, v_0) \leftarrow A(R_0, L_{2r+1}, v_0) - 1$ and $A(R_0, L_{2r+1}, v_1) \leftarrow A(R_0, L_{2r+1}, v_1) - 1$
4. Compute w non-uniformity values $\nu(i)$ according to (8.4.5), where i corresponds to a value of $R_0 \bmod w$.
5. Find the value $x \in \{0, \ldots, w-1\}$ for which $\sum_{i=0}^{log_2 w - 1} |\nu((x+i) \bmod w)|$ is maximal.

6. Guess $S_1 \bmod w = w - x$.

We have implemented our attack on RC5-32. The results are given in Table 8.1. We have done tests only up to 4 rounds. We only did up to 4 rounds due to the increase in the amount of texts required is growing exponentially with increase in number of rounds of RC5 for reasonable success rate as estimated in Table 8.2. We have used 100 different keys for our results. Note that we performed these attacks for RC5-32 only.

Table 8.1: Experimental results of the linear attack on RC5-32

Rounds	Known Plaintexts	Success Rate
2	2^{14}	45%
	2^{15}	87%
	2^{16}	90%
3	2^{20}	47%
	2^{21}	68%
	2^{22}	79%
4	2^{26}	42%
	2^{27}	76%

The results confirm the results found in [27] for the same number of known plaintexts used. According to [27], we could estimate the number of plaintexts required to perform the linear attack and use it also for RC5-64. Table 8.2 shows the expected number of plaintexts required for the attack as given in [27].

Table 8.2: Expected number of plaintexts required for a known plaintext attack on $r(\geq 2)$ rounds of RC5-32 or RC5-64

Success Probability	45%	90%
RC5-32	2^{2+6r}	2^{4+6r}
RC5-64	2^{1+8r}	2^{3+8r}

Chapter 9

RELATED KEY CRYPTANALYSIS

In this Chapter, we are going to discuss a different type of cryptanalysis which is called Related Key Cryptanalysis. Unlike the different cryptanalyses described in the previous Chapters, related key cryptanalysis assumes that the attacker learns the encryption of certain plaintexts under the original unknown key K, but also under some derived key $K' = f(K)$. In a chosen related key attack, the attacker specifies how the key is to be changed; known related key attacks are those where the key difference is known, but cannot be chosen by the attacker. The attacker knows or chooses the relationship between keys, not the actual key values. Before we describe further about this cryptanalysis, we would like to mention about linearly weak keys of RC5. It may seem that this topic has nothing to do with related key cryptanalysis but it is important for our discussion on the cryptanalysis that we are going to do.

9.1 Linearly Weak Keys of RC5

Linearly weak keys of RC5 were first discovered by Heys in [51]. Heys described the existence of these keys through modification of the linear cryptanalysis method that was used by Kaliski and Yin in [55]. This method was described in the previous Chapter. Heys outlined that the algorithm for the linear attack by Kaliski and Yin can be modified by considering that one of w^2 possible pairs of values of the $log_2 w$ least significant bits

of L_0 and R_0, *i.e.* L_0^{rot} and R_0^{rot}, will result in $R_1^{rot} = R_0^{rot} + S_1^{rot} = 0$ and $R_2^{rot} = L_0^{rot} + S_0^{rot} + S_2^{rot} = 0$. When this is true, $R_3^{rot} = S_3^{rot}$ and a linear attack for an r-round cipher can now be based on an (r - 2)-round approximation of $L_{2r}[0] = R_3[0] = S_3[0]$. The resulting bias of the (r - 2)-round approximation is significantly larger than the bias of an (r - 1)-round approximation which does not use fixed values for L_0^{rot} and R_0^{rot}.

Heys found that when we fix the inputs to L_0^{rot} and R_0^{rot} such that $R_1^{rot} = 0$ and $R_2^{rot} = 0$, this implies that $R_3^{rot} = S_3^{rot}$. If we then assume that $S_3^{rot} = 0$ (this happens with probability $1/w$ assuming that the key schedule generates reasonably pseudo-random subkeys) then $R_3^{rot} = 0$, $L_3^{rot} = 0$ and $R_4^{rot} = S_4^{rot}$. If we extend this argument so that $S_i^{rot} = 0$ for $3 \leq i \leq 2(r-1)$, then $L_{2r}^{rot} = R_{2r-1}^{rot} = S_{2r-1}^{rot}$ for all input plaintexts with the correct values of L_0^{rot} and R_0^{rot}. In such cases, the bias of the approximation $L_{2r}[0] = 0$ is 1/2. This class of extremely weak keys are labelled as WK_{r-1} since it trivializes the first $r-1$ rounds of the cipher and requires $2(r-1)-2$ subkeys to have the $log_2 w$ least significant bits equal to zero. Assuming that the subkey values are random and independent, the probability of this weak key occurring is given by the product of the probabilities that the partial subkey at each round is 0. Therefore, the probability $P(WK_{r-1}) = 1/(w^{2r-4})$. So, if we have a 12-round RC5, the probability that the selected key is in WK_{r-1} class is 2^{-100}.

We can generalize the argument by defining weak keys denoted as WK_m for $1 \leq m \leq r-1$, where $S_i^{rot} = 0$ for all i, where $3 \leq i \leq 2m$. The probability that a key in the WK_m class is selected is given by

$$P(WK_m) = \frac{1}{w^{2m-2}} \tag{9.1.1}$$

In [51], Heys had assumed that we would be able to reduce the number of plaintext/ciphertext pairs required in linear cryptanalysis of RC5 based on the results obtained by the algorithm made by Kaliski and Yin. This means that the result obtained in [51] is no longer applicable to the linear attack method used by Borst, Preneel and Vandewalle which we had described in the previous Chapter. However, we are going to use a method of that utilizes these linearly weak keys to obtain the subkeys of RC5.

9.2 Related Key Cryptanalysis

The idea of the related key attack is that the attacker knows (or chooses) a relation between several keys (up to 256 in some recent attacks) and is given access to encryption functions with such related keys. The goal of the attacker is to find the keys themselves. The first attacks of this type were developed independently by Biham [6] and Knudsen [59], and the notion of a *related key attack* was defined by Biham [6].

The relation between the keys can be an arbitrary bijective function R (or even a family of such functions) chosen (or known) in advance by the adversary [3]. In the simplest form of this attack, this relation is just an XOR with a constant: $K_2 = K_1 \oplus C$, where the constant C is chosen by the adversary. This type of relation allows the adversary to trace the propagation of XOR differences induced by the key difference C through the key schedule of the cipher. However, more complex forms of this attack allow other (possibly nonlinear) relations between the keys. For example, [6] uses rotational relations $K_2 = ROL(K_1)$ and the attack on AES-256 [19] uses XOR relations between the subkeys $K_2 = F^{-1}(F(K_1 \oplus C) = R_C(K_1)$.

In this Section, we are looking at Related Key Cryptanalysis as applied to RC5. Let us first consider an r-round RC5 cipher. We have $S_0, \ldots, S_{2r+1}$ to denote the $2r+2$ subkeys for the r-round cipher. We also denote $S_0^{rot}, \ldots, S_{2r+1}^{rot}$ as the $log_2 w$ least significant bits (LSB) of $S_0, \ldots, S_{2r+1}$ respectively. Assume that we had used a linearly weak key in our r-round cipher such that $S_i^{rot} = 0$ for $i \leq i \leq 2r-1$. Next, let us consider a (r-1)-round RC5 cipher. We then have a set of $2r$ subkeys denoted as $S_0^*, \ldots, S_{2r}^*$. Similarly, for this set of subkeys we have $S_0^{*rot}, \ldots, S_{2r}^{*rot}$ as the $log_2 w$ least significant bits (LSB) of $S_0^*, \ldots, S_{2r}^*$ respectively. Since we are assuming that we know the relationship between two set of subkeys then let us have the following relationship for the subkeys.

$$\begin{aligned} S_0^{*rot} &= S_0^{rot} + S_2^{rot}, \\ S_1^{*rot} &= S_1^{rot} + S_3^{rot}, \\ S_i^{*rot} &= S_{i+2}^{rot} \quad (2 \leq i < 2r) \end{aligned} \tag{9.2.1}$$

As S_3^{rot} is zero, we can simplify (9.2.1) further into

$$\begin{aligned} S_0^{*rot} &= S_0^{rot} + S_2^{rot}, \\ S_1^{*rot} &= S_1^{rot}, \\ S_i^{*rot} &= S_{i+2}^{rot} \quad (2 \leq i < 2r) \end{aligned} \tag{9.2.2}$$

In the previous Chapter we described linear cryptanalysis and gave some experimental results on performing this attack on RC5. So, we can estimate the number of plaintext/ciphertext pairs required to perform linear cryptanalysis on RC5. Say, for an r(≥ 3)-round RC5-32 we require N plaintext/ciphertext pairs for a 90% success rate to perform the linear cryptanalysis described in [27], then for an (r-1)-round RC5-32 we would reduce the number of plaintext/ciphertext pairs by a factor of 64 so $N/64$ would be needed to obtain the same success rate.

In terms of practicality, it would be possible to obtain parameters like the number of rounds (r) used in RC5 block cipher since only the subkeys are unknown. If we choose the subkey relationships as given by (9.2.2) then we would be able to perform linear cryptanalysis on an (r-1)-round RC5 instead of an r-round RC5. This would be done by changing a parameter of RC5, namely, the number of rounds. Once that is done, we would then obtain plaintext/ciphertext pairs based on this reduced RC5 in order to perform linear cryptanalysis.

However, this attack would only be possible if the key used in the r-round RC5 block cipher is a weak key. Table 9.1 gives an estimate on the number of plaintext/ciphertext pairs required if we had used the method we had described here. This estimate is based on the number of plaintext/ciphertext pairs required when we use linear cryptanalysis method by Borst, Preneel and Vandewalle in [27].

Table 9.1: Expected number of plaintexts required for related key attack on $r(\geq 3)$ rounds of RC5-32 or RC5-64

Success Probability	45%	90%
RC5-32	2^{6r-4}	2^{6r-2}
RC5-64	2^{8r-7}	2^{8r-5}

Chapter 10

OUR PROPOSAL ON DISTINGUISHING RC5 FROM A RANDOM PERMUTATION

In this Chapter, we discuss several statistical tests that has been applied to RC5 similar to the tests performed on DES in Chapter 7. We wanted to compare the results that we obtained for DES with that of RC5 in this Chapter. We wanted to find out whether DES and RC5 have the same or similar Distinguishing Attack Profile. By adjusting the parameters of RC5 to match that of DES in terms of block size, outputs from both algorithms are supposed to be indistiguishable from each other as well as from a random permutation. By looking at the Distinguishing Attack Profile, we wanted to be able to recognise which algorithm we are using. Our tests rely on ciphertexts produced from various rounds of RC5, from 1-round to the full 12-round version. We want to determine how many ciphertexts are required to distinguish between the ciphertext output of a block cipher and a random permutation. We also want to determine the number of rounds required in order for the ciphertexts to be indistinguishable from a random permutation. These tests are done because we want to make a comparison to the results that we got for DES in Chapter 7. We want to see how different or similar those results are as compared to another block cipher algorithm. We chose RC5 for this purpose due to the flexibility of setting the block size. Therefore, we chose RC5 with word size 32 bits which would be able to encrypt a plaintext block of size 64 bits since DES also encrypts a plaintext block

Table 10.1: Distinguishing Attack Profile for RC5

RC5 Rounds	Without Autofeeding				With Autofeeding				Figure No. in this thesis
	SAC	2-bit mod-SAC	4-bit mod-SAC	8-bit mod-SAC	SAC	2-bit mod-SAC	4-bit mod-SAC	8-bit mod-SAC	
1	Fail (2^8)	Fail (2^7)	Fail (2^7)	Fail (2^8)	Fail (2^{20})	Fail (2^{16})	Fail (2^{15})	Fail (2^{15})	Figs.C.1-8
2	Fail (2^9)	Fail (2^7)	Fail (2^7)	Fail (2^8)	Fail (2^{24})	Fail (2^{23})	Fail (2^{22})	Fail (2^{22})	Figs.C.9-16
3	Fail (2^{11})	Fail (2^7)	Fail (2^7)	Fail (2^8)	Pass	Pass	Pass	Pass	Figs.C.17-24
4	Fail (2^{15})	Fail (2^{14})	Fail (2^{14})	Fail (2^{14})	Pass	Pass	Pass	Pass	Figs.C.25-32
5	Fail (2^{20})	Fail (2^{20})	Fail (2^{20})	Fail (2^{21})	Pass	Pass	Pass	Pass	Figs.C.33-40
6 to 12	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Figs.C.41-96

of 64 bits.

We start off by giving a summary of the experimental results performed on RC5 and then give details on the experimental test results on RC5.

10.1 Summary of Experimental Results on RC5

In this Section, we summarise the results that we obtained from the various tests that we had performed for n-round RC5, $n = 1 \ldots 12$.

Note that in the Table 10.1, the number inside the braces indicates the minimum amount of ciphertext pairs where it began to fail the specific test. For tests without Autofeeding, from Table 10.1, we can see that n-round RC5 failed the experimental tests when $n < 6$. For tests with Autofeeding, Table 10.1 shows that n-round RC5 failed the experimental tests when $n < 3$.

We look at experimental tests without Autofeeding first. For 1-round RC5, we see that the minimum amount of ciphertext pairs needed to sufficiently distinguish from a random permutation is 2^8 for the SAC and 8-bit modSAC Tests while we needed half that amount for 2-bit modSAC and 4-bit modSAC Tests. For 2-round RC5, the minimum amount of ciphertext pairs needed to sufficiently distinguish from a random permutation is 2^9 for the

SAC Test, while the amount of ciphertext pairs required remains the same as with 1-round RC5 for 2-bit modSAC, 4-bit modSAC and 8-bit modSAC Tests. For 3-round RC5, the minimum amount of ciphertext pairs required had increased 4 times from 2^9 for the SAC Test while for the other tests, the amount of ciphertext pairs required remained the same as what we have for 1-round RC5 and 2-round RC5. For 4-round RC5, the amount of ciphertext pairs required again increased 16 times that of what we have for 3-round RC5 while for 2-bit modSAC, 4-bit modSAC and 8-bit modSAC tests, we only required half this amount. For 5-round RC5, the number of ciphertext pairs needed is the same for the SAC, 2-bit modSAC and 4-bit modSAC Tests. For 8-bit modSAC Test, it needed twice this amount.

For n-round RC5 where $1 < n < 5$, the performance in distiguishing our observed distribution from uniform random distribution of 2-bit modSAC, 4-bit modSAC and 8-bit modSAC Tests were better than the SAC Test. For 1-round RC5, the performance of SAC Test is comparable with that of 8-bit modSAC Test's performance. However, for 5-round RC5, the performance of the SAC Test is comparable with that of 2-bit modSAC and 4-bit modSAC Tests and better than that of 8-bit modSAC Test.

Next, we look at experimental tests with Autofeeding. For 1-round RC5, we see that the amount of ciphertext pairs needed to sufficiently distinguish from a random permutation is 2^{20} for the SAC Test while for 2-bit modSAC Test, the amount of ciphertext pairs needed is one sixteenth this amount whereas for 4-bit modSAC and 8-bit modSAC Tests, the amount of ciphertext pairs required is one thirty-second of 2^{20}. For 2-round RC5, the amount of ciphertext pairs needed is 2^{24} for the SAC Test while 2-bit modSAC Test required half this amount whereas for 4-bit modSAC and 8-bit modSAC Tests, the amount of ciphertext pairs required is a quarter of 2^{24}. In terms of performance in sufficiently distinguishing from a random permutation, for both 1-round RC5 and 2-round RC5, 2-bit modSAC, 4-bit modSAC and 8-bit modSAC Tests were better than that of the SAC Test.

10.2 Detail of Experimental Results on RC5

In this Section, we describe the experimental setup that was used to perform our various tests on various number of rounds of RC5. Basically, there are four tests that are being performed, namely,

i SAC Test

ii SAC Test with Autofeeding

iii modSAC Test for n-bit substrings, where $n = 2, 4, 8$

iv modSAC Test with Autofeeding for n-bit substrings, where $n = 2, 4, 8$

The SAC tests and modSAC tests performed on RC5 are similar tests performed on DES in the previous Chapter. Input data sets for these tests are plaintexts and keys. The output data sets are ciphertexts. The plaintexts and keys are generated random data. The number of plaintexts generated range from 2^7 to 2^{24}. 10 trials would be performed for each test, *i.e.* each test would be used on 10 different input data sets and we used these data sets for each number of rounds of the cryptographic algorithm. For the case of RC5, we started off testing 1 round and gradually increased the number of rounds until finally we tested on the full 12-rounds. The results are displayed graphically according to the number of ciphertext pairs being used (see Appendix C). We subdivided the graphs into four parts according to the number of ciphertexts used. Part (a) ranges from 2^7 to 2^{11} ciphertext pairs. Part (b) is from 2^{12} to 2^{16} ciphertext pairs. Part (c) ranges from 2^{17} to 2^{20} ciphertext pairs and part (d) is from 2^{21} up to 2^{24} ciphertext pairs. We note that for a plaintext P and a key K, we will produce a ciphertext C but we will also produce a ciphertext C' from plaintext P' and and the same key K. The hamming distance between P and P' is 1 and P' is produced by randomly choosing and toggling one bit position of P. In the case of tests with Autofeeding, we generated a random key, K and a random plaintext, P. We then generate another plaintext P' from P so that the hamming distance between P and P' is 1 and P' is produced by randomly choosing and toggling one bit position of P. P and K would be fed to RC5 to produce ciphertext C. Similarly, P' and K would be fed to RC5 to produce ciphertext C'. For Autofeeding, these ciphertexts would become the next plaintext inputs to be fed into RC5 to produce another set of new ciphertexts. This cycle repeats until the required number of ciphertexts is reached. The tests were performed on RC5 with parameters $w = 32$,$r = 1 \ldots 12$ and $b = 16$. This means that the word size, w is 32 bits and and the number of bytes b used for the secret key is 16.

10.2.1 Tests on RC5

As we mentioned previously, we performed four tests. We will start by displaying the results from the SAC Test and the modSAC Test for n-bit substrings, where $n = 2, 4, 8$. This is then followed the results for the SAC Test with Autofeeding and modSAC Test with Autofeeding for n-bit substrings, where $n = 2, 4, 8$.

10.2.1.1 SAC Test

Table 10.2: Experimental Results of SAC Tests on Various Rounds of RC5

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	63	0.005	95.649	198.727	256($i.e.$ 2^8)	Fail	Fig.C.1
2	63	0.005	95.649	139.311	512($i.e.$ 2^9)	Fail	Fig.C.9
3	63	0.005	95.649	120.516	2048($i.e.$ 2^{11})	Fail	Fig.C.17
4	63	0.005	95.649	127.294	32768($i.e.$ 2^{15})	Fail	Fig.C.25
5	63	0.005	95.649	118.886	1048576($i.e.$ 2^{20})	Fail	Fig.C.33
6	63	0.005	95.649	58.1954	16777216($i.e.$ 2^{24})	Pass	Fig.C.41
7	63	0.005	95.649	34.3148	16777216($i.e.$ 2^{24})	Pass	Fig.C.49
8	63	0.005	95.649	40.34	16777216($i.e.$ 2^{24})	Pass	Fig.C.57
9	63	0.005	95.649	41.7892	16777216($i.e.$ 2^{24})	Pass	Fig.C.65
10	63	0.005	95.649	35.1084	16777216($i.e.$ 2^{24})	Pass	Fig.C.73
11	63	0.005	95.649	43.924	16777216($i.e.$ 2^{24})	Pass	Fig.C.81
12	63	0.005	95.649	34.805	16777216($i.e.$ 2^{24})	Pass	Fig.C.89

As we can see from the table above that 1- to 5-Round RC5 fail the SAC test. The p-value that we used is 0.005 which is a standard statistical significance level value for χ^2 test. The number of ciphertext pairs needed to distinguish from a random permutation is 256 pairs for 1-Round RC5. The number of pairs increases to 512 pairs for 2-Round RC5, then 2048 pairs for 3-Round RC5, 32768 pairs for 4-Round RC5 and finally 1048576 pairs for 5-Round RC5. 95.649 is the χ^2 test threshold value when the degrees of freedom is $63(64 - 1)$ and the p-value as stated previously. For 6-Round RC5 or higher, we did not find the experimental χ^2 test value exceeding or approaching towards this threshold value for 16777216 ciphertext pairs. So, 6-Round RC5 or higher pass the SAC test.

10.2.1.2 modSAC Test for 2-bit substring

Table 10.3: Experimental Results of modSAC Tests for 2-bit substring on Various Rounds of RC5

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	3	0.005	12.838	5676.07	128($i.e.$ 2^7)	Fail	Fig.C.2
2	3	0.005	12.838	858.937	128($i.e.$ 2^7)	Fail	Fig.C.10
3	3	0.005	12.838	30.4125	128($i.e.$ 2^7)	Fail	Fig.C.18
4	3	0.005	12.838	34.7027	16384($i.e.$ 2^{14})	Fail	Fig.C.26
5	3	0.005	12.838	13.0495	1048576($i.e.$ 2^{20})	Fail	Fig.C.34
6	3	0.005	12.838	3.86949	16777216($i.e.$ 2^{24})	Pass	Fig.C.42
7	3	0.005	12.838	3.70583	16777216($i.e.$ 2^{24})	Pass	Fig.C.50
8	3	0.005	12.838	3.20419	16777216($i.e.$ 2^{24})	Pass	Fig.C.58
9	3	0.005	12.838	2.86368	16777216($i.e.$ 2^{24})	Pass	Fig.C.66
10	3	0.005	12.838	3.47462	16777216($i.e.$ 2^{24})	Pass	Fig.C.74
11	3	0.005	12.838	3.82949	16777216($i.e.$ 2^{24})	Pass	Fig.C.82
12	3	0.005	12.838	4.98504	16777216($i.e.$ 2^{24})	Pass	Fig.C.90

In the above table, only 1- to 5-Round RC5 fail the modSAC test for 2-bit substring. Similarly, we use a p-value of 0.005 for the significance level for the χ^2 test. The number of ciphertext pairs needed to distinguish from a random permutation using this test is the same for 1-Round RC5 up to and including 3-round RC5. Then it suddenly increase to 16384 for 4-round RC5 and further increase to 1048576 for 5-Round RC5. 6-Round RC5 or higher, pass the modSAC test for 2-bit substring. 12.838 is the χ^2 test threshold value when the degrees of freedom is $3(4-1)$ and the p-value as stated previously. For 6-Round RC5 or higher, we did not find the experimental χ^2 test value exceeding or approaching towards this threshold value for 16777216 ciphertext pairs

10.2.1.3 modSAC Test for 4-bit substring

Table 10.4: Experimental Results of modSAC Tests for 4-bit substring on Various Rounds of RC5

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	15	0.005	32.801	12792.2	128($i.e.$ 2^7)	Fail	Fig.C.3
2	15	0.005	32.801	1804.78	128($i.e.$ 2^7)	Fail	Fig.C.11
3	15	0.005	32.801	70.9734	128($i.e.$ 2^7)	Fail	Fig.C.19
4	15	0.005	32.801	66.0697	16384($i.e.$ 2^{14})	Fail	Fig.C.27
5	15	0.005	32.801	34.1705	1048576($i.e.$ 2^{20})	Fail	Fig.C.35
6	15	0.005	32.801	16.5666	16777216($i.e.$ 2^{24})	Pass	Fig.C.43
7	15	0.005	32.801	16.5852	16777216($i.e.$ 2^{24})	Pass	Fig.C.51
8	15	0.005	32.801	13.1851	16777216($i.e.$ 2^{24})	Pass	Fig.C.59
9	15	0.005	32.801	17.4684	16777216($i.e.$ 2^{24})	Pass	Fig.C.67
10	15	0.005	32.801	17.8568	16777216($i.e.$ 2^{24})	Pass	Fig.C.75
11	15	0.005	32.801	16.778	16777216($i.e.$ 2^{24})	Pass	Fig.C.83
12	15	0.005	32.801	17.7279	16777216($i.e.$ 2^{24})	Pass	Fig.C.91

The results that we obtain in the above table are very similar to the results that we get for the modSAC test for 2-bit substring. We can see from the above table that only 1- to 5-round RC5 fail the modSAC test for 4-bit substring. The number of ciphertext pairs needed to distinguish from a random permutation using this test is the same for 1-Round RC5 up to and including 3-round RC5. Then it suddenly increase to 16384 for 4-round RC5 and further increase to 1048576 for 5-Round RC5. 6-Round RC5 or higher, pass the modSAC test for 4-bit substring. 32.801 is the χ^2 test threshold value when the degrees of freedom is $15(16-1)$ and the p-value as stated previously. For 6-Round RC5 or higher, we did not find the experimental χ^2 test value exceeding or approaching towards this threshold value for 16777216 ciphertext pairs. Thus these higher round RC5 pass our modSAC test for 4-bit substring.

10.2.1.4 modSAC Test for 8-bit substring

Table 10.5: Experimental Results of modSAC Tests for 8-bit substring on Various Rounds of RC5

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	255	0.005	316.919	155011	256($i.e.$ 2^8)	Fail	Fig.C.4
2	255	0.005	316.919	15503	256($i.e.$ 2^8)	Fail	Fig.C.12
3	255	0.005	316.919	550.175	256($i.e.$ 2^8)	Fail	Fig.C.20
4	255	0.005	316.919	418.723	16384($i.e.$ 2^{14})	Fail	Fig.C.28
5	255	0.005	316.919	350.417	2097152($i.e.$ 2^{21})	Fail	Fig.C.36
6	255	0.005	316.919	254.535	16777216($i.e.$ 2^{24})	Pass	Fig.C.44
7	255	0.005	316.919	253.653	16777216($i.e.$ 2^{24})	Pass	Fig.C.52
8	255	0.005	316.919	246.877	16777216($i.e.$ 2^{24})	Pass	Fig.C.60
9	255	0.005	316.919	266.301	16777216($i.e.$ 2^{24})	Pass	Fig.C.68
10	255	0.005	316.919	247.419	16777216($i.e.$ 2^{24})	Pass	Fig.C.76
11	255	0.005	316.919	254.829	16777216($i.e.$ 2^{24})	Pass	Fig.C.84
12	255	0.005	316.919	253.275	16777216($i.e.$ 2^{24})	Pass	Fig.C.92

From the table above, we see that 1-Round RC5 fail the modSAC test for 8-bit substring. The number of ciphertext pairs sufficient for distinguishing 1-Round RC5 from a random permutation is 256 pairs. 256 pairs of ciphertexts is also sufficient for distinguishing 2-Round RC5 and 3-Round RC5, from a random permutation. For 4-Round RC5, the number of ciphertext pairs needed to make this distinction is 16384 and for 5-Round RC5, at least 2097152 pairs is required. 316.919 is the χ^2 test threshold value when the degrees of freedom is $255(256-1)$ with a p-value of 0.005. For 6-Round RC5 or higher, we did not find the experimental χ^2 test value exceeding or approaching towards this threshold value for 16777216 ciphertext pairs. Thus these higher round RC5 pass our modSAC test for 8-bit substring.

10.2.1.5 SAC Test with Autofeeding

Table 10.6: Experimental Results of SAC Tests on Various Rounds of RC5 (Autofeeding)

No. of Rounds	d.o.f	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	63	0.005	95.649	119.201	1048576($i.e.$ 2^{20})	Fail	Fig.C.5
2	63	0.005	95.649	50.1576	16777216($i.e.$ 2^{24})	Pass	Fig.C.13
3	63	0.005	95.649	39.3954	16777216($i.e.$ 2^{24})	Pass	Fig.C.21
4	63	0.005	95.649	38.4581	16777216($i.e.$ 2^{24})	Pass	Fig.C.29
5	63	0.005	95.649	35.8877	16777216($i.e.$ 2^{24})	Pass	Fig.C.37
6	63	0.005	95.649	38.3948	16777216($i.e.$ 2^{24})	Pass	Fig.C.45
7	63	0.005	95.649	38.0321	16777216($i.e.$ 2^{24})	Pass	Fig.C.53
8	63	0.005	95.649	36.8354	16777216($i.e.$ 2^{24})	Pass	Fig.C.61
9	63	0.005	95.649	34.3999	16777216($i.e.$ 2^{24})	Pass	Fig.C.69
10	63	0.005	95.649	40.2835	16777216($i.e.$ 2^{24})	Pass	Fig.C.77
11	63	0.005	95.649	35.7596	16777216($i.e.$ 2^{24})	Pass	Fig.C.85
12	63	0.005	95.649	39.0038	16777216($i.e.$ 2^{24})	Pass	Fig.C.93

When we perform the SAC Test with Autofeeding, we see a difference in the results as compared to the SAC test results in Table 10.2 in terms of the number of rounds where RC5 fail the test and also the number of required ciphertext pairs. Only 1-Round RC5 fails the SAC Test with Autofeeding. The number of ciphertext pairs required to distinguish the ciphertext outputs of 1-Round RC5 from random outputs is 1048576. 2-Round RC5 or higher pass the test. This means that these higher round RC5 are able to generate ciphertext outputs indistinguishable from random outputs.

10.2.1.6 modSAC Test with Autofeeding for 2-bit substring

Table 10.7: Experimental Results of modSAC Tests for 2-bit substring on Various Rounds of RC5 (Autofeeding)

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	3	0.005	12.838	14.0934	65536($i.e.$ 2^{16})	Fail	Fig.C.6
2	3	0.005	12.838	17.4723	8388608($i.e.$ 2^{23})	Fail	Fig.C.14
3	3	0.005	12.838	4.11861	16777216($i.e.$ 2^{24})	Pass	Fig.C.22
4	3	0.005	12.838	2.039	16777216($i.e.$ 2^{24})	Pass	Fig.C.30
5	3	0.005	12.838	3.84587	16777216($i.e.$ 2^{24})	Pass	Fig.C.38
6	3	0.005	12.838	1.79948	16777216($i.e.$ 2^{24})	Pass	Fig.C.46
7	3	0.005	12.838	2.38491	16777216($i.e.$ 2^{24})	Pass	Fig.C.54
8	3	0.005	12.838	3.2513	16777216($i.e.$ 2^{24})	Pass	Fig.C.62
9	3	0.005	12.838	2.12019	16777216($i.e.$ 2^{24})	Pass	Fig.C.70
10	3	0.005	12.838	4.74951	16777216($i.e.$ 2^{24})	Pass	Fig.C.78
11	3	0.005	12.838	3.02097	16777216($i.e.$ 2^{24})	Pass	Fig.C.86
12	3	0.005	12.838	1.78175	16777216($i.e.$ 2^{24})	Pass	Fig.C.94

As we can see from the table above, our modSAC Test with Autofeeding for 2-bit substring is slightly better than the SAC Test with Autofeeding in Table 10.6 in terms of distinguishing the generated ciphertext outputs from random outputs. The SAC Test with Autofeeding is only able to distinguish the generated outputs of 1-Round RC5 from random outputs whereas our modSAC Test with Autofeeding for 2-bit substring is able to do this distinction for 1-Round RC5 and 2-Round RC5. The number of ciphertext pairs required for 1-Round RC5 to make this distinction is 65536 and for 2-Round RC5 is 8388608. 3-Round RC5 or higher pass the test. This means that these higher round RC5 are able to generate ciphertext outputs indistiguishable from random outputs.

10.2.1.7 modSAC Test with Autofeeding for 4-bit substring

Table 10.8: Experimental Results of modSAC Tests for 4-bit substring on Various Rounds of RC5 (Autofeeding)

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	15	0.005	32.801	35.2929	32768($i.e.$ 2^{15})	Fail	Fig.C.7
2	15	0.005	32.801	50.3696	4194304($i.e.$ 2^{22})	Fail	Fig.C.15
3	15	0.005	32.801	18.1037	16777216($i.e.$ 2^{24})	Pass	Fig.C.23
4	15	0.005	32.801	13.3448	16777216($i.e.$ 2^{24})	Pass	Fig.C.31
5	15	0.005	32.801	16.8036	16777216($i.e.$ 2^{24})	Pass	Fig.C.39
6	15	0.005	32.801	10.9897	16777216($i.e.$ 2^{24})	Pass	Fig.C.47
7	15	0.005	32.801	13.859	16777216($i.e.$ 2^{24})	Pass	Fig.C.55
8	15	0.005	32.801	15.0436	16777216($i.e.$ 2^{24})	Pass	Fig.C.63
9	15	0.005	32.801	14.6511	16777216($i.e.$ 2^{24})	Pass	Fig.C.71
10	15	0.005	32.801	16.449	16777216($i.e.$ 2^{24})	Pass	Fig.C.79
11	15	0.005	32.801	16.7118	16777216($i.e.$ 2^{24})	Pass	Fig.C.87
12	15	0.005	32.801	15.9253	16777216($i.e.$ 2^{24})	Pass	Fig.C.95

From the table above, our modSAC Test with Autofeeding for 4-bit substring produce slightly better result than our modSAC Test with Autofeeding for 2-bit substring in Table 10.7 in terms of the number of ciphertext pairs needed to distinguish generated ciphertext outputs from random outputs. 1-Round RC5 requires only 32768 pairs instead of 65536 pairs and 2-Round RC5 requires 4194304 pairs instead of 8388608 pairs. However, the number of rounds which fail the modSAC Test with Autofeeding for 4-bit substring is the same as for the modSAC Test with Autofeeding for 2-bit substring. Only 1-Round RC5 and 2-Round RC5 fail this test. 3-Round RC5 or higher, pass the test.

10.2.1.8 modSAC Test with Autofeeding for 8-bit substring

Table 10.9: Experimental Results of modSAC Tests for 8-bit substring on Various Rounds of RC5 (Autofeeding)

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	255	0.005	316.919	323.907	32768($i.e.$ 2^{15})	Fail	Fig.C.8
2	255	0.005	316.919	329.563	4194304($i.e.$ 2^{22})	Fail	Fig.C.16
3	255	0.005	316.919	247.718	16777216($i.e.$ 2^{24})	Pass	Fig.C.24
4	255	0.005	316.919	241.326	16777216($i.e.$ 2^{24})	Pass	Fig.C.32
5	255	0.005	316.919	256.238	16777216($i.e.$ 2^{24})	Pass	Fig.C.40
6	255	0.005	316.919	254.54	16777216($i.e.$ 2^{24})	Pass	Fig.C.48
7	255	0.005	316.919	260.008	16777216($i.e.$ 2^{24})	Pass	Fig.C.56
8	255	0.005	316.919	258.011	16777216($i.e.$ 2^{24})	Pass	Fig.C.64
9	255	0.005	316.919	253.176	16777216($i.e.$ 2^{24})	Pass	Fig.C.72
10	255	0.005	316.919	264.841	16777216($i.e.$ 2^{24})	Pass	Fig.C.80
11	255	0.005	316.919	246.527	16777216($i.e.$ 2^{24})	Pass	Fig.C.88
12	255	0.005	316.919	259.5	16777216($i.e.$ 2^{24})	Pass	Fig.C.96

The results that we have in the table above for the modSAC Test with Autofeeding for 8-bit substring is similar to the result obtained from our modSAC Test with Autofeeding for 4-bit substring in Table 10.8 in terms of the number of ciphertext pairs needed to distinguish ciphertext outputs from random outputs. 1-Round RC5 requires 32768 ciphertext pairs to make this distinction and 2-Round RC5 requires 4194304 ciphertext pairs. This means only 1-Round RC5 and 2-Round RC5 fail the modSAC Test with Autofeeding for 8-bit substring. 3-Round RC5 or higher, pass this test. This means that these higher round RC5 are able to generate ciphertext outputs indistiguishable from random outputs.

Chapter 11

CONCLUSION

There have been major developments in cryptanalysis of block ciphers. We were given the chance to investigate one of the oldest and most enduring block ciphers, *i.e.* DES. We were able to do a software implementation of DES, though not as optimised as some other implementations. We were successful in implementing Differential Cryptanalysis of 8-round DES with the retrieval of all 56-bit key. We were also able to do Linear Cryptanalysis and Differential-Linear Cryptanalysis of 8-round DES. We were also successful in implementing our modified version when using multiple linear approximations in Differential-linear cryptanalysis of 8-round DES and gave some experimental results of our attack. We would like to mention that our thesis does not give the complete picture of all kinds of cryptanalysis that can be done on DES, in particular, 8-round DES. There are some other methods [42, 30] that may be used against 8-round DES but we feel that the methods that we had used in this thesis fairly covers the different kinds of cryptanalysis that can be done on 8-round DES.

The latter half of this thesis covers our contribution in the statistical cryptanalysis field. In particular we develop new tests based on the Strict Avalanche Criterion which are used in Distinguishing Attack. We have used a well-known test, the SAC Test, and modified it to produce our own tests called 2-bit modSAC, 4-bit modSAC and 8-bit modSAC Tests. These tests (without Autofeeding) were designed to find out at what stage, in terms of number of rounds and number of ciphertext pairs required, the ciphertext outputs produced were indistinguishable from a random permutation.

In addition to that, we also performed the SAC Test as well as our modified tests with

Autofeeding to make a comparison with our new tests. Again these tests were devised to find out at what stage, in terms of number of rounds and number of ciphertext pairs required, we can generate ciphertext outputs indistinguishable from a random permutation.

Thus, we produced a Distinguishing Attack Profile for DES as given in Table 7.1 in Chapter 7. In Chapter 10, we had performed the same tests as in Chapter 7 but on the RC5 algorithm. We also produced a Distinguishing Attack Profile for RC5 as given in Table 10.1. We found that for tests with Autofeeding, only 1-round RC5 and 2-round RC5 failed the tests (see Table 10.1). Whereas, for tests without Autofeeding, n-round RC5 where $n < 6$, failed the tests. This is unlike the results that we obtained for n-round DES using tests with Autofeeding and without Autofeeding. Both tests with and without Autofeeding failed only for n-round DES where $n < 6$.

For the case of the tests with Autofeeding, this shows that RC5 is faster than DES in generating ciphertexts indistinguishable from a random permutation. For the case of the tests without Autofeeding, the amount of ciphertext pairs required where RC5 began to fail the tests grows more rapidly than what we have for DES. However, both RC5 and DES began to pass all tests without Autofeeding from 6 rounds onwards. From both Table 7.1 and Table 10.1, we also compared the performance, in terms of distiguishing from a random permutation, of SAC Test with our modified Tests, either with Autofeeding or without Autofeeding.

Generally, for the case of Tests without Autofeeding, the performance of our modified tests, in particular, 2-bit modSAC Test and 4-bit modSAC Test are better than that of the SAC Test. For the case of Tests with Autofeeding, the performance of our modified tests are still better than that of the SAC Test as well.

We had fairly covered, to some extent, a range of cryptanalysis attacks/methods involving block ciphers, in particular DES. As a comparative study, we subjected RC5 to the same distinguishers that we had used for testing DES so that we can empirically judge the effectiveness of the distinguishers in attacking another block cipher algorithm similar in configuration to DES. Our techniques have successfully differentiated between DES and RC5.

We had also implemented our proposed technique on two well known block ciphers, *i.e.* AES [32, 33, 82] and KASUMI [1]. In the following Section we give a summary of our

experimental results on both block ciphers.

11.1 Distinguishing Block Ciphers from a Random Permutation

11.1.1 AES

We had performed our proposed technique on AES-128. Table 11.1 gives a summary of the experimental results that we obtained for 1-round AES-128 till the full 10-round AES-128.

Table 11.1: Distinguishing Attack Profile of AES

AES Rounds	Without Autofeeding				With Autofeeding				Figure No. in this thesis
	SAC	2-bit mod-SAC	4-bit mod-SAC	8-bit mod-SAC	SAC	2-bit mod-SAC	4-bit mod-SAC	8-bit mod-SAC	
1	Fail (2^8)	Fail (2^7)	Fail (2^7)	Fail (2^7)	Fail (2^8)	Fail (2^7)	Fail (2^7)	Fail (2^7)	Figs.D.1-8
2	Fail (2^8)	Fail (2^7)	Fail (2^7)	Fail (2^7)	Fail (2^8)	Fail (2^7)	Fail (2^7)	Fail (2^7)	Figs.D.9-16
3	Fail (2^{16})	Fail (2^{12})	Fail (2^{12})	Fail (2^{10})	Fail (2^{16})	Fail (2^{12})	Fail (2^{12})	Fail (2^{10})	Figs.D.17-24
4 to 10	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Figs.D.25-80

From Table 11.1, we can see that n-round AES-128 failed the experimental tests when $n < 4$ for tests with and without Autofeeding. If we look at the experimental test results for tests without Autofeeding, we see that the minimum amount of ciphertext pairs needed to sufficiently distinguish from a random permutation is 2^8 for the SAC Test and half that amount for 2-bit modSAC, 4-bit modSAC and 8-bit modSAC tests. These are for 1-round and 2-round AES-128 only. For 3-round AES-128, we would require 2^{16} ciphertext pairs for the SAC Test, 2^{12} for both 2-bit modSAC and 4-bit modSAC Tests, and 2^{10} for 8-bit modSAC Test to distinguish from a random permutation. Surprisingly, these are also the same results we obtained for the experimental tests with Autofeeding, for n-round AES-128 where $n < 4$. We expected the results to be higher than the results we obtained from the experimental tests without Autofeeding. For 4- to 10-round AES-128, the block cipher passed the tests. Figure D.1 to Figure D.80 in Appendix D, give the graphical results for

all experimental tests done for the various rounds of AES-128. Table D.1 to Table D.8 give the details of experimental tests according to the different test categories *i.e.* SAC Test without Autofeeding, 2-bit modSAC Test without Autofeeding and so on.

11.1.2 KASUMI

We had also performed our proposed technique on KASUMI. Table 11.2 gives a summary of the experimental results that we obtained for 1-round KASUMI till the full 8-round KASUMI.

Table 11.2: Distinguishing Attack Profile of KASUMI

KASUMI Rounds	Without Autofeeding				With Autofeeding				Figure No. in this thesis
	SAC	2-bit mod-SAC	4-bit mod-SAC	8-bit mod-SAC	SAC	2-bit mod-SAC	4-bit mod-SAC	8-bit mod-SAC	
1	Fail (2^9)	Fail (2^7)	Fail (2^7)	Fail (2^8)	Fail (2^9)	Fail (2^7)	Fail (2^7)	Fail (2^8)	Figs.E.1-8
2	Fail (2^9)	Fail (2^7)	Fail (2^7)	Fail (2^8)	Fail (2^9)	Fail (2^7)	Fail (2^7)	Fail (2^8)	Figs.E.9-16
3 to 8	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Figs.E.17-64

From Table 11.2, we see that only n-round KASUMI where $n < 3$ failed the experimental tests. For both tests without Autofeeding and with Autofeeding, we have similar results. In fact, the results are the same for both 1-round KASUMI and 2-round KASUMI. We see that the minimum amount of ciphertext pairs need to sufficiently distinguish from a random permutation is 2^9 for the SAC Test either with Autofeeding or without Autofeeding. We needed 2^7 ciphertext pairs for 2-bit modSAC Test either with Autofeeding or without Autofeeding. This is also the amount required by 4-bit modSAC Test similarly, with Autofeeding or without Autofeeding. We required twice as much for 8-bit modSAC Test either with Autofeeding or without Autofeeding. For n-round KASUMI where $n > 2$, the cryptographic algorithm passed the experimental tests. Figure E.1 to Figure E.64 in Appendix E, give the graphical results for all experimental tests done for the various rounds of KASUMI. Table E.1 to Table E.8 give the details of experimental tests according to the different test categories.

11.2 Future Work

The modSAC tests that we proposed are fast and easy to implement in software. They are not meant to replace the already established SAC test. On the contrary, the modSAC tests can complement the existing SAC test. One thing that is obvious from the experimental results that we got when testing block ciphers is that, the tests are only able to detect reduced round versions of the block ciphers. By using the modSAC tests, we managed to reduce the minimum required ciphertext pairs by a factor of 2 or 4 or 8 etc. We believe that we can extend these modSAC tests further by incorporating other techniques so that it can be used to distinguish the full version of the block cipher algorithm. In Section 1 above, we demonstrated that the modSAC tests can be applied to other block ciphers, not just DES and RC5. We believe that the modSAC tests can also be used for other primitives such as stream ciphers, hash functions and pseudorandom number generators. We believe that further development in this area of research can enhance our knowledge and understanding of block cipher cryptanalysis.

Bibliography

[1] 3rd Generation Partnership Project, Technical Specification Group Services and System Aspects,"Specification of the 3GPP Confidentiality and Integrity Algorithms: Document 2: KASUMI Specification" *V.3.1.1.* 2001.

[2] K. Aoki and K. Ohta, "Differential-Linear Cryptanalysis of FEAL-8", *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences* vol.E79-A, No.1 pp. 20-27, 1996.

[3] M. Belare and T. Khono,"A theoretical treatment of related key attacks, *RKA-PRPs, RKA-PRFs, and Applications, Advances in cryptology EUROCRYPT 2003, Lecture Notes in Computer Science, Springer-Verlag*, vol 2656, pp 491-506, 2003.

[4] E. Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems", *Advances in Cryptology - CRYPTO'90, Lecture Notes in Computer Science, Springer-Verlag*, vol. 537, pp. 2-21, 1990.

[5] E. Biham and A. Shamir, "Differential Cryptanalysis of the Data Encryption Standard", *Springer-Verlag*, 1993.

[6] E. Biham, "New Types of Cryptanalytic Attacks Using Related Keys", *Advances in Cryptology - Eurocrypt'93, Lecture Notes in Computer Science, Springer-Verlag*, vol. 765, pp. 398-409, 1994.

[7] E. Biham, "On Matsui's Linear Cryptanalysis", *Advances in Cryptology - Eurocrypt'94, Lecture Notes in Computer Science, Springer-Verlag*, vol. 950, pp. 341-355, 1995.

[8] E. Biham, "An Improvement of Davies' Attack on DES", *Journal of Cryptology*, vol. 10, no. 3, pp. 195-205, 1997.

[9] E. Biham, A. Biryukov and A. Shamir,"Cryptanalysis of Skipjack Reduced to 31 Rounds using Impossible Differentials", *Advances in Cryptology: EUROCRYPT99, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1592, pp. 12-23, 1999.

[10] E. Biham, O. Dunkelman and N. Keller,"The rectangle attack - rectangling the Serpent", *EUROCRYPT 2001, Lecture Notes in Computer Science, Springer-Verlag*, vol. 2045, pp. 340-357, 2001.

[11] E. Biham, O. Dunkelman and N. Keller,"Linear cryptanalysis of reduced round Serpent", *FSE 2001, Lecture Notes in Computer Science, Springer-Verlag*, vol. 2355, pp. 16-27, 2002.

[12] E. Biham, O. Dunkelman and N. Keller,"Enhancing differential-linear cryptanalysis", *ASIACRYPT 2002, Lecture Notes in Computer Science, Springer-Verlag*, vol. 2501, pp. 254-266, 2002.

[13] E. Biham, O. Dunkelman and N. Keller,"New results on boomerang and rectangle attacks", *FSE 2002, Lecture Notes in Computer Science, Springer-Verlag*, vol. 2365, pp. 1-16, 2002.

[14] E. Biham, O. Dunkelman and N. Keller,"Differential-linear cryptanalysis of Serpent", *FSE 2003, Lecture Notes in Computer Science, Springer-Verlag*, vol. 2887, pp. 9-21, 2003.

[15] E. Biham, O. Dunkelman and N. Keller,"New combined attacks on block ciphers", *FSE 2005, Lecture Notes in Computer Science, Springer-Verlag*, vol. 3557, pp. 126-144, 2005.

[16] E. Biham, O. Dunkelman and N. Keller,"Related-Key Boomerang and Rectangle Attacks, *EUROCRYPT 2005, Lecture Notes in Computer Science, Springer-Verlag*, vol. 3494, pp. 507-525, 2005.

[17] E. Biham, O. Dunkelman and N. Keller,"A Related-Key Rectangle Attack on the Full KASUMI", *ASIACRYPT 2005, Lecture Notes in Computer Science, Springer-Verlag*, vol. 3788, pp. 443-461, 2005.

[18] E. Biham, O. Dunkelman and N. Keller,"A New Attack on 6-Round IDEA", *Fast Software Encryption - FSE '07, Lecture Notes in Computer Science, Springer-Verlag*, vol. 4593, pp. 211-224, 2007.

[19] A. Biryukov and D. Khovratovich D,"Related-Key Attack on the Full AES-192 and AES-256", *Advances in cryptology ASIACRYPT09. Lecture Notes in Computer Science, Springer-Verlag*, vol. 5912, pp.1-18, 2009.

[20] A. Biryukov and E. Kushilevitz,"Improved Cryptanalysis of RC5", *Advances in Cryptology - Eurocrypt'98, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1403, pp. 85-99, 1998.

[21] A. Biryukov and A. Shamir,"Structural cryptanalysis of SASAS", *Advances in Cryptology - EUROCRYPT 2001, Lecture Notes in Computer Science,Springer-Verlag*, vol. 2045, pp. 394405, 2001.

[22] A. Biryukov, C. De Cannière and M. Quisqater,"On Multiple Linear Approximations", *Advances in Cryptology - Crypto 2004, Lecture Notes in Computer Science, Springer-Verlag*, vol. 3152, pp. 1-22, 2004.

[23] A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. Robshaw, Y. Seurin, and C. Vikkelsoe,"PRESENT: An ultralightweight block cipher" *Cryptographic Hardware and Embedded Systems - CHES 2007, Lecture Notes in Computer Science, Springer-Verlag*, vol. 4727, pp. 450-466, 2007.

[24] A. Bogdanov and V. Rijmen,"Linear Hulls with Correlation Zero and Linear Cryptanalysis of Block Cipher", *Cryptology ePrint Archive, Report 2011/123, URL:https://eprint.iacr.org/2011/123*, 2011.

[25] J. Borst, "Differential-Linear Cryptanalysis of IDEA", *ESAT–COSIC Technical Report 96-2*, 1997.

[26] J. Borst, L. R. Knudsen and V. Rijmen, "Two Attacks on Reduced IDEA", *Advances in Cryptology - EUROCRYPT '97, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1233, pp. 1-13, 1997.

[27] J. Borst, B. Preneel and J. Vandewalle,"Linear Cryptanalysis of RC5 and RC6", *FSE'99, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1636, pp. 16-30, 1999.

[28] W. E. Burr,"Data Encryption Standard", *A Century of Excellence in Measurements, Standards, and Technology, A Chronicle of Selected NBS/NIST Publications, 1901-2000*, NIST Special Publication 958, pp. 250-253, 2001.

[29] J.C.H. Castro, J.M. Sierra, A. Seznec, A. Izquierdo and A. Ribagorda, "The strict avalanche criterion randomness test", *Mathematics and Computers in Simulation, Elsevier*, vol. 68, pp. 1-7, 2005.

[30] N.T. Courtois and G.V. Bard, "Algebraic cryptanalysis of the data encryption standard", *Cryptography and Coding, Springer*, pp. 152-169, 2007.

[31] J. Daemen, L. R. Knudsen, and V. Rijmen, "The block cipher SQUARE", *Fast Software Encryption - FSE97, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1267, pp. 149165, 1997.

[32] J.Daemen and V. Rijmen,"The Design of Rijndael AES - The Advanced Encryption Standard", *Information Security and Cryptography, Springer-Verlag*, 2001.

[33] J. Daemen and V. Rijmen, "AES Proposal: Rijndael", *National Institute of Standards and Technology*, 2003.

[34] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim and S. Chee,"HIGHT: A New Block Cipher Suitable for Low-Resource Device", *Cryptographic Hardware and Embedded Systems - CHES 2006, Lecture Notes in Computer Science, Springer-Verlag*, vol. 4249, pp. 46-59, 2006.

[35] W. Diffie and M. Hellman, "New directions in cryptography", *IEEE Transactions on Information Theory, IT-22*, pp. 644-654, 1976.

[36] W. Diffie and M. Hellman, "Exhaustive Cryptanalysis of the NBS Data Encryption Standard", *Computer, IEEE*, vol. 10, issue 6, pp. 74-84, 1977.

[37] Distributed.net,"US Government's Encrpytion Standard Broken In Less Than A Day", *Formal Press Release, 19th January 1999. URL: http://www.distributed.net/images/d/d7/19990119_-_PR_-_release-des3.pdf*

[38] O. Dunkelman, S. Indesteege and N. Keller,"A differential-linear attack on 12-round Serpent", *INDOCRYPT 2008, Lecture Notes in Computer Science, Springer-Verlag*, vol. 5365, pl. 308-321, 2008.

[39] O. Dunkelman and N. Keller,"Cryptanalysis of ctc2. *Topics in Cryptology CTRSA 2009, Lecture Notes in Computer Science, Springer-Verlag*, vol. 5473, pp 226239, 2009.

[40] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann and L. Uhsadel, "A Survey of Lightweight-Cryptography Implementations", *IEEE Design & Test of Computers 24(6)*, pp. 522-533, 2007.

[41] Electronic Frontier Foundation,"Cracking DES - Secrets of Encryption Research, Wiretap Politics & Chip Design", *O'Reilly Media*, 1st Edition, 1998.

[42] J.C. Faugère, and L. Perret and P.J. Spaenlehauer, "Algebraic-differential cryptanalysis of DES", *Western European Workshop on Research in Cryptology-WEWoRC*, pp. 1-5, 2009.

[43] N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner, and D. Whiting, "Improved cryptanalysis of Rijndael", *Fast Software Encryption, FSE 2000, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1978, pp. 213230, 2001.

[44] S. Fluhrer, I. Mantin and A. Shamir,"Weaknesses in the key scheduling algorithm of RC4" *Selected areas in cryptography, Lecture Notes in Computer Science, Springer-Verlag*, vol. 2259, pp. 1-24, 2001.

[45] R. Forre, "The strict avalanche criterion: spectral properties of boolean functions and an extended definition", *Advances in Cryptology - Crypto'88, Lecture Notes in Computer Science, Springer-Verlag*, vol. 403, pp. 450-468, 1990.

[46] H. Gilbert and M. Minier, "A collision attack on seven rounds of Rijndael, *Proceedings of the 3rd AES Candidate Conference 2000*, pp. 230241, 2000.

[47] H. Handschuh and H. M. Heys, "A Timing Attack on RC5", *SAC'98, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1556, pp. 306-318, 1999.

[48] J.C. Hernández, J. S. Sierra, A. Ribagorda, B. Ramos and J.C. Mex-Perera, "Distinguishing TEA from a Random Permutation: Reduced Round Versions of TEA Do Not Have the SAC or Do Not Generate Random Numbers", *Cryptography and Coding, Lecture Notes in Computer Science, Springer-Verlag*, vol. 2260, pp. 374-377, 2001.

[49] J.C. Hernández, J. S. Sierra, A. Seznec,"The SAC Test: A New Randomness Test, with Some Applications to PRNG Analysis", *ICCSA 2004, Lecture Notes in Computer Science, Springer-Verlag*, vol. 3043, pp. 960-967, 2004.

[50] M. Hellman,"DES will be totally insecure within ten years", *IEEE Spectrum* vol. 16, no. 7, pp. 31-41, 1979.

[51] H.M. Heys, "Linearly Weak Keys of RC5", *IEE Electronics Letters*, vol. 33, no. 10, pp. 836-837, 1997.

[52] B.S. Kaliski Jr. and M.J.B. Robshaw, "Linear Cryptanalysis Using Multiple Linear Approximations", *Advances in Cryptology - Proceedings of Crypto'94, Lecture Notes in Computer Science, Springer-Verlag*, Vol. 839, pp. 26-39, 1994.

[53] B.S. Kaliski and M.J.B. Robshaw, "Linear Cryptanalysis Using Multiple Linear Approximations and FEAL", *Fast Software Encryption, Lecture Notes in Computer Science, Springer-Verlag*, Vol. 1008, pp. 249-264, 1995.

[54] D. Kahn, "The Codebreakers: The Story of Secret Writing", *Macmillan Press*, 1967

[55] B.S. Kaliski and Y.L. Yin, "On Differential and Linear Cryptanalysis of the RC5 Encryption Algorithm", *Advances in Cryptology - Crypto'95, Lecture Notes in Computer Science, Springer-Verlag*, vol. 963, pp. 171-184, 1995.

[56] J. Kelsey, T. Kohno and B. Schneier,"Amplified Boomerang Attacks Against Reduced-Round MARS and Serpent", *FSE 2000, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1978, pp. 75-93, 2000.

[57] J. Kelsey, B. Schneier and D. Wagner,"Key-Schedule Cryptanalysis of IDEA, G-DES, GOST, SAFER and Triple-DES", *Advances in Cryptology - Crypto'96, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1109, pp. 237-251, 1996.

[58] J. Kelsey, B. Schneier and D. Wagner,"Related-Key Cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2 and TEA", *Lecture Notes in Computer Science, Springer-Verlag*, vol. 1334, pp. 233-246, 1997.

[59] L. R. Knudsen,"Cryptanalysis of LOKI91". *Advances in cryptography ASIACRYPT92, Lecture Notes in Computer Science, Springer-Verlag*, vol. 718, pp 22-35, 1993.

[60] L. R. Knudsen," Block Ciphers Analysis, Design and Applications", *Ph.D Thesis, Aarhus University*, 1994.

[61] L. Knudsen, "Truncated and higher order differentials", *FSE, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1008, pp.196-211, 1995.

[62] L. R. Knudsen, "Block ciphers - a survey", *State of the Art in Applied Cryptography, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1528, pp. 18-48, 1998.

[63] L. R. Knudsen and J. E. Mathiassen,"A chosen-plaintext linear attack on DES", *Fast Software Encryption, FSE 2000, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1978, pp. 262272, 2001.

[64] L. R. Knudsen and J.E. Mathiassen, "On the Role of Key Schedules in Attacks on Iterated Ciphers", *Computer Security - ESORICS 2004, Lecture Notes in Computer Science, Springer-Verlag*, vol. 3193, pp. 322-334, 2004.

[65] L. R. Knudsen and W. Meier, "Improved Differential Attack on RC5", *Advances in Cryptology - Crypto'96, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1109, pp. 216-228, 1996.

[66] L.R. Knudsen and M.J.B. Robshaw, "Non-Linear Approximations in Linear Cryptanalysis", *Advances in Cryptology - Eurocrypt'96, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1070, pp. 224-236, 1996.

[67] L. Knudsen and D. Wagner,"Integral Cryptanalysis (Extended Abstract)", *FSE 2002, Lecture Notes in Computer Science, Springer-Verlag*, vol. 2365, pp. 112127, 2002.

[68] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems", *Advances in Cryptology - CRYPTO 96, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1109, pp. 104113, 1996.

[69] P. C. Kocher, J. Jaffe and B. Jun, "Differential power analysis", *Advances in Cryptology - CRYPTO 99, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1666, pp. 388397, 1999.

[70] S.K. Langford, M.E. Hellman, "Differential-Linear Cryptanalysis", *Advances in Cryptology - Crypto'94, Lecture Notes in Computer Science, Springer-Verlag*, vol. 839, pp. 17-25, 1994.

[71] S.K. Langford, "Differential-Linear Cryptanalysis and Threshold Signatures", *Ph.D Thesis, Stanford University*, 1995.

[72] J. Lu,"A Methodology for Differential-Linear Cryptanalysis and Its Applications". *Fast Software Encryption, Lecture Notes in Computer Science, Springer-Verlag*, vol. 7549, pp. 6989, 2012.

[73] S. Lucks,"Attacking seven rounds of Rijndael under 192-bit and 256-bit keys", *Proceedings of the 3rd AES Candidate Conference 2000*, pp. 215229, 2000.

[74] M. Matsui, "Linear Cryptanalysis Method for DES Cipher", *Advances in Cryptology - Eurocrypt'93, Lecture Notes in Computer Science, Springer-Verlag*, vol. 765, pp. 386-397, 1993.

[75] M. Matsui, "The First Experimental Cryptanalysis of the Data Encryption Standard", *Advances in Cryptology - Crypto'94, Lecture Notes in Computer Science, Springer-Verlag*, vol. 839, pp. 1-11, 1994.

[76] M. Matsui, "On Correlation between the Order of S-Boxes and the strength of DES", *Advances in Cryptology - Eurocrypt'94, Lecture Notes in Computer Science, Springer-Verlag*, vol. 950, pp. 366-375, 1995.

[77] S. Murphy and M.J.B. Robshaw, "Key-Dependent S-Boxes and Differential Cryptanalysis", *Design, Codes and Cryptography*, vol. 27, no. 3, pp. 229-255, 2002.

[78] A. G. Naim,"Multiple Linear Approximations in Differential-Linear Cryptanalysis of 8-Round DES", *Scientia Bruneiana*, 2013.

[79] New European Schemes for Signatures, Integrity, and Encryption,"Test Vectors for All NESSIE Candidates", *2004. URL: https://www.cosic.esat.kuleuven.be/nessie/testvectors/*

[80] National Bureau of Standards,"Data Encryption Standard", *Federal Information Processing Standard 46*, 1977.

[81] OpenMP,"The OpenMP® API Specification for Parallel Programming", *Version 4.5. URL: http://www.openmp.org/mp-documents/openmp-4.5.pdf*

[82] United States National Institute of Standards and Technology (NIST), "Announcing the ADVANCED ENCRYPTION STANDARD (AES)" , *Federal Information Processing Standards Publication 197*, 2001.

[83] Request for Comments: 1321,"The MD5 Message-Digest Algorithm", *April 1992, URL: https://tools.ietf.org/html/rfc1321*

[84] R.L. Rivest, "The RC5 Encryption Algorithm", *Fast Software Encryption, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1008, pp. 86-96, 1994.

[85] R.L. Rivest, M.J.B. Robshaw, R. Sidney and Y.L. Yin, "The RC6™ Block Cipher", *Version 1.1, August 1998. URL: http://people.csail.mit.edu/rivest/pubs/RRSY98.pdf*

[86] B. Schneier,"Applied Cryptography Source Code", *URL: https://www.schneier.com/books/applied_cryptography/source.html*

[87] C. E. Shannon, "Communication Theory of Secrecy Systems", *Bell System Technical Journal*, vol. 28, pp. 656715, 1949.

[88] T. Shimoyama and T. Kaneko,"Quadratic relation of s-box and its application to the linear attack of full round DES", *Advances in Cryptology - CRYPTO98, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1462, pp. 200211,1998.

[89] Y. Shin, J. Kim, G. Kim, S. Hong, and S. Lee,"Differential-Linear Type Attacks on Reduced Rounds of SHACAL-2", *Information Security and Privacy, Lecture Notes in Computer Science, Springer-Verlag*, vol. 3108, pp. 110122, 2004.

[90] D.R.Stinson, Cryptography: Theory and Practice, 2nd Ed., *CRC Press*, 2002.

[91] A. A. Selcuk,"New Results in Linear Cryptanalysis of RC5", *Fast Software Encryption, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1372, pp. 1-16, 1998.

[92] D. Wagner,"The Boomerang Attack", *Fast Software Encryption, FSE99, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1636, pp. 156170, 1999.

[93] M. J. Weiner,"Exhaustive Key Search", *Encyclopedia of Cryptography and Security, Editors: H. C. A. van Tilborg, S. Jajodia, Springer-Verlag*, pp. 431-433, 2011.

[94] W. Wu and D. Feng,"Differential-Linear Cryptanalysis of Camellia", *Progress on Cryptography, The International Series in Engineering and Computer Science*, vol. 769, pp. 173-180, 2004.

[95] H. Wu and B. Preneel, "Differential-Linear Attacks Against the Stream Cipher Phelix", *Fast Software Encryption, Lecture Notes in Computer Science, Springer-Verlag*, vol. 4593, pp. 87100, 2007.

A C Source Code for DES Implementation

A.1 des.h

```
/*************************************************************
des.h - Header file for DES

*************************************************************/


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>



/*************************************************************
getbit - this function is used to extract a particular
         bit from "bits" given by the bit position "pos"

*************************************************************/

char getbit(const unsigned char *bits, int pos);


/*************************************************************
setbit - this function is used to set a particular bit
         at position "pos" in "bits" to a particular value
         "state". Normally, "state" is either 0 or 1

*************************************************************/

void setbit(unsigned char *bits, int pos, int state);


/*************************************************************
bitxor -  this function is used to perform bit xoring of
          "bits1" and "bits2" and store the result in
          "bitsx". The number of bits to be xored is given
          by "size"

*************************************************************/

void bitxor(const unsigned char *bits1, const unsigned char *bits2,
unsigned char *bitsx, int size);
```

```
/*********************************************************
lrotbit - This function is used to perform the left
          cicular rotation of "bits"of size "size". The
          number of left rotations is given by "count".

*********************************************************/

void lrotbit(unsigned char *bits, int size, int count);


/*********************************************************
permute - This function is used to permute "bits"
          according to the function "mapping". "n" gives
          the size of "bits".

*********************************************************/

void permute(unsigned char *bits, const char *mapping, int n);


/*********************************************************
substitute - This function is the non linear substitution
             function in function F

*********************************************************/

void substitute(unsigned char *in, unsigned char *out);

/*********************************************************
f - This is the function which incorporates the
    substitution permutation network for DES

*********************************************************/

void  f(unsigned char input[4], unsigned char subkey[6],
unsigned char output[4]);


/*********************************************************
round1 - This is the round function which incorporates the
        F function. The input consists of the left and
        right halves (each 32 bits) of the input, namely,
        "left" and "right". "nleft" and "nright" are the
        input for the next round. "subkey" gives the
        subkey used in each round.

*********************************************************/
```

```
void round1(unsigned char *left, unsigned char *right, unsigned char *nleft,
unsigned char *nright, unsigned char *subkey);

/*********************************************************
keyschedule - This function is used to generate the 48-bit
              subkeys for each round of DES. The
              keyschedule is stored in "ks" and generated
              from 64-bit "key"

*********************************************************/

void keyschedule(unsigned char key[8], unsigned char ks[16][6]);



/*********************************************************
deseord - This function is used to perform the DES
          encryption or decryption depending on "yn". If
          "yn" is 1, we perform encryption and if "yn" is
          0 then we perform decryption. "p" is the 64-bit
          plaintext, "key" is 64-bit key and "out" is
          64-bit output/ciphertext. "rnd" is the number of
          rounds for encryption/decryption.

*********************************************************/


void deseord(unsigned char p[8], unsigned char key[8],
unsigned char out[8], int rnd, int yn);


/*********************************************************
deseord1 - This function is the same as the function
           deseord above with the removal of initial and
           final permutations removed for differential
           cryptanalysis

*********************************************************/


void deseord1(unsigned char p[8], unsigned char key[8],
unsigned char out[8], int rnd, int yn);
```

A.2 des.c

```
/*********************************************************
```

```
des.c - this is C implementation of DES. We need to
        compile this file using gcc -o3 -c des.c to
        obtain des.o file so that we can use in other
        files.

**********************************************************/



/* start of tables */

/**********************************************************
ip - Initial Permutation (IP) table on 64 bit input
     (This will be ignored in differential cryptanalysis)

**********************************************************/

char ip[64] = { 58, 50, 42, 34, 26, 18, 10, 2,
                60, 52, 44, 36, 28, 20, 12, 4,
                62, 54, 46, 38, 30, 22, 14, 6,
                64, 56, 48, 40, 32, 24, 16, 8,
                57, 49, 41, 33, 25, 17,  9, 1,
                59, 51, 43, 35, 27, 19, 11, 3,
                61, 53, 45, 37, 29, 21, 13, 5,
                63, 55, 47, 39, 31, 23, 15, 7 };

/**********************************************************
fp - Final Permutation (FP) table on 64 bit input.
     (This is also ignored in differential cryptanalysis)

**********************************************************/

char fp[64] = { 40, 8, 48, 16, 56, 24, 64, 32,
                39, 7, 47, 15, 55, 23, 63, 31,
                38, 6, 46, 14, 54, 22, 62, 30,
                37, 5, 45, 13, 53, 21, 61, 29,
                36, 4, 44, 12, 52, 20, 60, 28,
                35, 3, 43, 11, 51, 19, 59, 27,
                34, 2, 42, 10, 50, 18, 58, 26,
                33, 1, 41,  9, 49, 17, 57, 25 };

/**********************************************************
ex - Expansion table to expand 32 bit input into 48 bit
     output in the F function

**********************************************************/

char ex[48] = { 32,  1,  2,  3,  4,  5,
```

```
                 4,  5,  6,  7,  8,  9,
                 8,  9, 10, 11, 12, 13,
                12, 13, 14, 15, 16, 17,
                16, 17, 18, 19, 20, 21,
                20, 21, 22, 23, 24, 25,
                24, 25, 26, 27, 28, 29,
                28, 29, 30, 31, 32,  1 };

/*********************************************************
s - The famous s-boxes (8 of them i.e. S1 to S8) in the
    function F.

*********************************************************/

char s[8][4][16] = {
{{ 14,  4, 13,  1,  2, 15, 11,  8,  3, 10,  6, 12,  5,  9,  0,  7 },
{  0, 15,  7,  4, 14,  2, 13,  1, 10,  6, 12, 11,  9,  5,  3,  8 },
{  4,  1, 14,  8, 13,  6,  2, 11, 15, 12,  9,  7,  3, 10,  5,  0 },
{  15, 12,  8,  2,  4,  9,  1,  7,  5, 11,  3, 14, 10,  0,  6, 13 }},

{{ 15,  1,  8, 14,  6, 11,  3,  4,  9,  7,  2, 13, 12,  0,  5, 10 },
{  3, 13,  4,  7, 15,  2,  8, 14, 12,  0,  1, 10,  6,  9, 11,  5 },
{  0, 14,  7, 11, 10,  4, 13,  1,  5,  8, 12,  6,  9,  3,  2, 15 },
{ 13,  8, 10,  1,  3, 15,  4,  2, 11,  6,  7, 12,  0,  5, 14,  9 }},

{{ 10,  0,  9, 14,  6,  3, 15,  5,  1, 13, 12,  7, 11,  4,  2,  8 },
{ 13,  7,  0,  9,  3,  4,  6, 10,  2,  8,  5, 14, 12, 11, 15,  1 },
{ 13,  6,  4,  9,  8, 15,  3,  0, 11,  1,  2, 12,  5, 10, 14,  7 },
{  1, 10, 13,  0,  6,  9,  8,  7,  4, 15, 14,  3, 11,  5,  2, 12 }},

{{  7, 13, 14,  3,  0,  6,  9, 10,  1,  2,  8,  5, 11, 12,  4, 15 },
{ 13,  8, 11,  5,  6, 15,  0,  3,  4,  7,  2, 12,  1, 10, 14,  9 },
{ 10,  6,  9,  0, 12, 11,  7, 13, 15,  1,  3, 14,  5,  2,  8,  4 },
{  3, 15,  0,  6, 10,  1, 13,  8,  9,  4,  5, 11, 12,  7,  2, 14 }},

{{  2, 12,  4,  1,  7, 10, 11,  6,  8,  5,  3, 15, 13,  0, 14,  9 },
{ 14, 11,  2, 12,  4,  7, 13,  1,  5,  0, 15, 10,  3,  9,  8,  6 },
{  4,  2,  1, 11, 10, 13,  7,  8, 15,  9, 12,  5,  6,  3,  0, 14 },
{ 11,  8, 12,  7,  1, 14,  2, 13,  6, 15,  0,  9, 10,  4,  5,  3 }},

{{ 12,  1, 10, 15,  9,  2,  6,  8,  0, 13,  3,  4, 14,  7,  5, 11 },
{ 10, 15,  4,  2,  7, 12,  9,  5,  6,  1, 13, 14,  0, 11,  3,  8 },
{  9, 14, 15,  5,  2,  8, 12,  3,  7,  0,  4, 10,  1, 13, 11,  6 },
{  4,  3,  2, 12,  9,  5, 15, 10, 11, 14,  1,  7,  6,  0,  8, 13 }},

{{  4, 11,  2, 14, 15,  0,  8, 13,  3, 12,  9,  7,  5, 10,  6,  1 },
{ 13,  0, 11,  7,  4,  9,  1, 10, 14,  3,  5, 12,  2, 15,  8,  6 },
{  1,  4, 11, 13, 12,  3,  7, 14, 10, 15,  6,  8,  0,  5,  9,  2 },
```

```
{  6, 11, 13,  8,  1,  4, 10,  7,  9,  5,  0, 15, 14,  2,  3, 12  }},

{{ 13,  2,  8,  4,  6, 15, 11,  1, 10,  9,  3, 14,  5,  0, 12,  7  },
{  1, 15, 13,  8, 10,  3,  7,  4, 12,  5,  6, 11,  0, 14,  9,  2  },
{  7, 11,  4,  1,  9, 12, 14,  2,  0,  6, 10, 13, 15,  3,  5,  8  },
{  2,  1, 14,  7,  4, 10,  8, 13, 15, 12,  9,  0,  3,  5,  6, 11  }} };


/*******************************************************
p - Permutation table in the F function applied to the
    output from the s boxes.

*******************************************************/

char p[32] = { 16,  7, 20, 21, 29, 12, 28, 17,
               1, 15, 23, 26,  5, 18, 31, 10,
               2,  8, 24, 14, 32, 27,  3,  9,
               19, 13, 30,  6, 22, 11,  4, 25 };

/*******************************************************
pc1 - Permuted Choice 1 table. This permutation is used
      on the key to permute 56 bits (after removal of
      8 parity bits) in the process of producing subkeys

*******************************************************/


char pc1[56] = { 57, 49, 41, 33, 25, 17,  9,
                 1, 58, 50, 42, 34, 26, 18,
                 10,  2, 59, 51, 43, 35, 27,
                 19, 11,  3, 60, 52, 44, 36,
                 63, 55, 47, 39, 31, 23, 15,
                 7, 62, 54, 46, 38, 30, 22,
                 14,  6, 61, 53, 45, 37, 29,
                 21, 13,  5, 28, 20, 12,  4 };




/*******************************************************
pc2 - Permuted Choice 2 table. This permutation is used
      in producing the subkeys in each round.

*******************************************************/

char pc2[56] = { 14, 17, 11, 24,  1,  5,  3, 28,
                 15,  6, 21, 10, 23, 19, 12,  4,
                 26,  8, 16,  7, 27, 20, 13,  2,
```

```
                41, 52, 31, 37, 47, 55, 30, 40,
                51, 45, 33, 48, 44, 49, 39, 56,
                34, 53, 46, 42, 50, 36, 29, 32 };



/*********************************************************
lrot - This table gives the number of left circular
       rotations used for each round in producing
       subkeys

*********************************************************/

char lrot[16] = { 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1 };


/* end of tables */

/*********************************************************
getbit - this function is used to extract a particular
         bit from "bits" given by the bit position "pos"

*********************************************************/

char getbit(const unsigned char *bits, int pos)
{
  unsigned char mask;
  int i;

  mask = 0x80;
  /* adjust the mask according to position of required bit */
  for(i=0;i<(pos%8);i++)
    mask = mask >> 1;

  return(((mask & bits[(int)(pos/8)]) == mask) ? 1: 0);
}

/*********************************************************
setbit - this function is used to set a particular bit
         at position "pos" in "bits" to a particular value
         "state". Normally, "state" is either 0 or 1

*********************************************************/

void setbit(unsigned char *bits, int pos, int state)
{
  unsigned char mask;
  int i;
```

```
  mask = 0x80;
  /* set the mask according to position of required bit */
  for(i=0;i<(pos%8);i++)
    mask = mask >> 1;

  if(state)
    bits[pos/8] |= mask;
  else
    bits[pos/8] &= (~mask);


}

/*********************************************************
bitxor -  this function is used to perform bit xoring of
          "bits1" and "bits2" and store the result in
          "bitsx". The number of bits to be xored is given
          by "size"
*********************************************************/

void bitxor(const unsigned char *bits1, const unsigned char *bits2,
unsigned char *bitsx, int size)
{
  int i;

  for(i=0;i<size;i++)
    {
      if (getbit(bits1,i) != getbit(bits2,i))
setbit(bitsx, i, 1);
      else
        setbit(bitsx, i, 0);
    }


}

/*********************************************************
lrotbit - This function is used to perform the left
          cicular rotation of "bits"of size "size". The
          number of left rotations is given by "count".

*********************************************************/

void lrotbit(unsigned char *bits, int size, int count)
{

  int i,j;
```

```
  char fbit, lbit;

  if (size > 0)
    {
      for(i=0;i<count;i++)
{
          for(j=0;j<=((size - 1)/8);j++)
    {
              lbit = getbit(&bits[j], 0);
              if(j==0)
{
                  fbit = lbit;
}
              else
{
                  setbit(&bits[j-1], 7, lbit);
}

              bits[j] = bits[j] << 1;
    }
  setbit(bits, size - 1, fbit);
}
    }

}

/*********************************************************
permute - This function is used to permute "bits"
          according to the function "mapping". "n" gives
          the size of "bits".

*********************************************************/

void permute(unsigned char *bits, const char *mapping, int n)
{
  unsigned char temp[8];
  int i;

  /* initialize "temp" to zero */
  memset(temp, 0, (int)ceil(n/8));
  /* perform permutation according to "mapping" */
  for(i=0;i<n;i++)
    setbit(temp, i, getbit(bits, mapping[i] - 1));

  /* copy "temp" back to "bits" */
  memcpy(bits, temp, (int)ceil(n/8));
```

```
}

/*********************************************************
substitute - This function is the non linear substitution
             function in function F

*********************************************************/

void substitute(unsigned char *in, unsigned char *out)
    {
      int i, j;
      unsigned char p, row, col, sblk;


      p = 0;
      /* in is divided into 8 blocks of 6 bits */
      for(i=0;i<8;i++)
{
 /* row is formed from the 1st and 6th bits of each of
     the 8 blocks of "in"*/
          row = (getbit(in, (i*6)+0)*2) + (getbit(in, (i*6)+5)*1);
/* col is formed from the 2nd,3rd,4th and 5th bits of each of
     the 8 blocks of "in" */
          col = (getbit(in, (i*6)+1)*8) + (getbit(in, (i*6)+2)*4) +
                (getbit(in, (i*6)+3)*2) + (getbit(in, (i*6)+4)*1);
          sblk = s[i][row][col];
          for(j=4;j<8;j++)
    {
              /*p holds the bit position in "in" having bit 1 */
              /* the last four bits of "sblk" are the required bits */
              setbit(out, p, getbit(&sblk, j));
              p++;
    }
}
    }

/*********************************************************
f - This is the function which incorporates the
    substitution permutation network for DES

*********************************************************/

  void  f(unsigned char input[4], unsigned char subkey[6],
  unsigned char output[4])
    {

      unsigned char xblk[6], tblk[6];
```

```
      /* copy "input" to "tblk" */
      memcpy(tblk, input, 4);
      /* expand "tblk" using the expansion table "ex" to
      48 bits using "permute"*/
      permute(tblk, ex, 48);
      /* perform bit xoring of the round subkey and "tblk"
       and put into "xblk" */
      bitxor(tblk, subkey, xblk, 48);
      /* copy "xblk" to "tblk" */
      memcpy(tblk, xblk, 6);
      /*perform the substition function */
      substitute(tblk, output);
      /* lastly perform the p permutation function */
      permute(output, p, 32);

    }

/*********************************************************
round1 - This is the round function which incorporates the
        F function. The input consists of the left and
        right halves (each 32 bits) of the input, namely,
        "left" and "right". "nleft" and "nright" are the
        input for the next round. "subkey" gives the
        subkey used in each round.

*********************************************************/

  void round1(unsigned char *left, unsigned char *right,
  unsigned char *nleft, unsigned char *nright, unsigned char *subkey)
    {
      unsigned char temp[4];

/* perform the F function on the right half "right"  to produce new
 right half "nright" */
      f(right, subkey, nright);
/* perform bit xoring of the left half "left" with "nright" and store
  it in "temp"*/
      bitxor(nright, left, temp, 32);
      /* copy "temp" back to "nright" */
      memcpy(nright, temp, 4);
/* the right half "right" becomes the left half in the next
round "nleft" */
      memcpy(nleft, right, 4);

    }

/*********************************************************
keyschedule - This function is used to generate the 48-bit
```

```
              subkeys for each round of DES. The
              keyschedule is stored in "ks" and generated
              from 64-bit "key"

**********************************************************/

   void keyschedule(unsigned char key[8], unsigned char ks[16][6])
    {
      int i,j;
      unsigned char temp[8], c1[4], d1[4];

      /* copy "key" to "temp" */
      memcpy(temp, key, 8);
      /* use pc1 table to permute "temp" */
      permute(temp, pc1, 56);
      /* c1 and d1 are the left and right halves of "temp" */
      /* initialize c1 and d1 to zero */
      memset(c1, 0, 4);
      memset(d1, 0, 4);
      /* copy the left half of "temp" to "c1" */
      for(i=0;i<28;i++)
         setbit(c1, i, getbit(temp, i));
      /* copy the right half of "temp" to "d1" */
      for(i=0;i<28;i++)
        setbit(d1, i, getbit(temp, i+28));

      for(i=0;i<16;i++)
{
/* perform left circular rotation according to "lrot" independently
     on "c1" and "d1" */
         lrotbit(c1, 28, lrot[i]);
         lrotbit(d1, 28, lrot[i]);

/* copy the bits from "c1" and "d1" to respective round subkey "ks" */
         for(j=0;j<28;j++)
    setbit(ks[i], j, getbit(c1, j));
         for(j=0;j<28;j++)
            setbit(ks[i], j+28, getbit(d1, j));

/* permute respective round subkey using pc2 permutation function */
         permute(ks[i], pc2, 48);
}

    }


/*********************************************************
deseord - This function is used to perform the DES
```

```
         encryption or decryption depending on "yn". If
         "yn" is 1, we perform encryption and if "yn" is
         0 then we perform decryption. "p" is the 64-bit
         plaintext, "key" is 64-bit key and "out" is
         64-bit output/ciphertext. "rnd" is the number of
         rounds for encryption/decryption.

**********************************************************/


  void deseord(unsigned char p[8], unsigned char key[8],
  unsigned char out[8], int rnd, int yn)
    {

      int i;
      unsigned char temp[8], left[4], right[4], nleft[4], nright[4], work[4];
      unsigned char ks[16][6];

      /* generate the key schedule first */
      keyschedule(key, ks);
      /* copy the plaintext "p" to "temp" */
      memcpy(temp, p, 8);
      /* perform the initial permutation on "temp" */
      permute(temp, ip, 64);
      /* copy the first 4 bytes to "left" and next 4 bytes to "right" */
      memcpy(left, &temp[0], 4);
      memcpy(right, &temp[4], 4);

      for(i=0;i<rnd;i++)
      {
          /* copy the right half "right" to "work" */
  memcpy(work, right, 4);
          if (yn == 1)
    {
      /* perform round function for encryption */
                round(left, work, nleft, nright, ks[i]);
             }
          else
    {
         /* perform round function for decryption */
         round(left, work, nleft, nright, ks[rnd-1-i]);
    }
          /* copy "nleft" to "left" and "nright" to "right" to prepare
             for next round */
          memcpy(left, nleft, 4);
          memcpy(right, nright, 4);
     }
      /* swap "right" with "left" */
```

```
      memcpy(&out[0], right, 4);
      memcpy(&out[4], left, 4);

      /* perform the final permutation "p" on the final output "out" */
      permute(out, fp, 64);
  }

/*********************************************************
deseord1 - This function is the same as the function
           deseord above with the removal of initial and
           final permutations removed for differential
           cryptanalysis

*********************************************************/


 void deseord1(unsigned char p[8], unsigned char key[8],
 unsigned char out[8], int rnd, int yn)
    {

      int i;
      unsigned char temp[8], left[4], right[4], nleft[4], nright[4], work[4];
      unsigned char ks[16][6];

      /* generate the key schedule first */
      keyschedule(key, ks);
      /* copy the plaintext "p" to "temp" */
      memcpy(temp, p, 8);

      /* copy the first 4 bytes to "left" and next 4 bytes to "right" */
      memcpy(left, &temp[0], 4);
      memcpy(right, &temp[4], 4);

      for(i=0;i<rnd;i++)
      {
          /* copy the right half "right" to "work" */
  memcpy(work, right, 4);
          if (yn == 1)
            {
         /* perform round function for encryption */
                round(left, work, nleft, nright, ks[i]);
             }
          else
            {
         /* perform round function for decryption */
         round(left, work, nleft, nright, ks[rnd-1-i]);
            }
          /* copy "nleft" to "left" and "nright" to "right" to prepare
```

```
                for next round */
            memcpy(left, nleft, 4);
            memcpy(right, nright, 4);
}
       /* swap "right" with "left" */
       memcpy(&out[0], right, 4);
       memcpy(&out[4], left, 4);
 }
```

A.3 Timing Comparison of DES Implementations

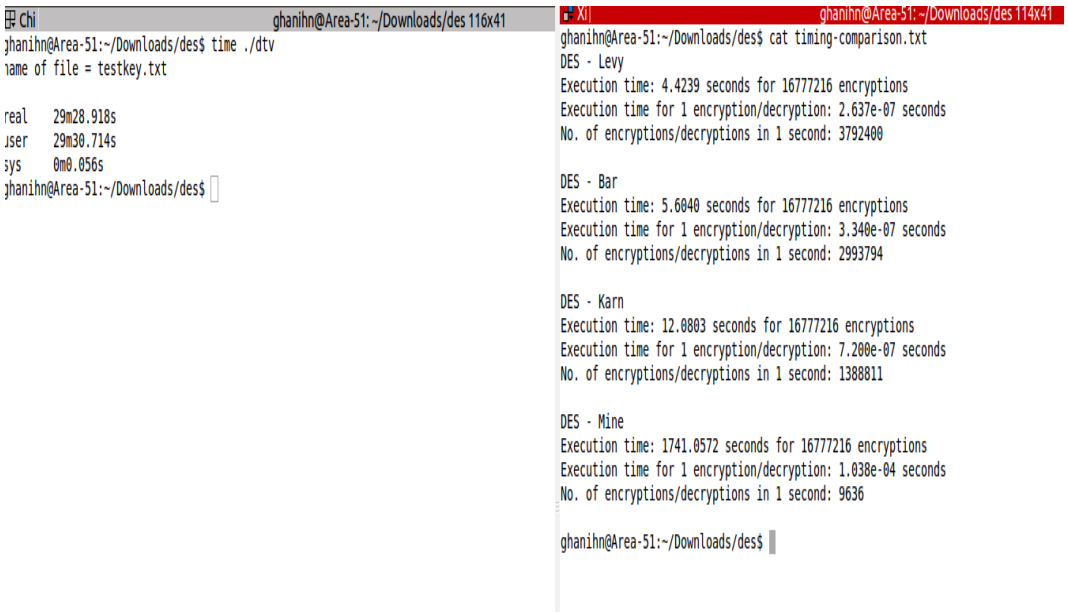

Figure A.3.1: Timing Comparison 4 DES implementations for 16777216 plaintexts

B Experimental Test Results for DES

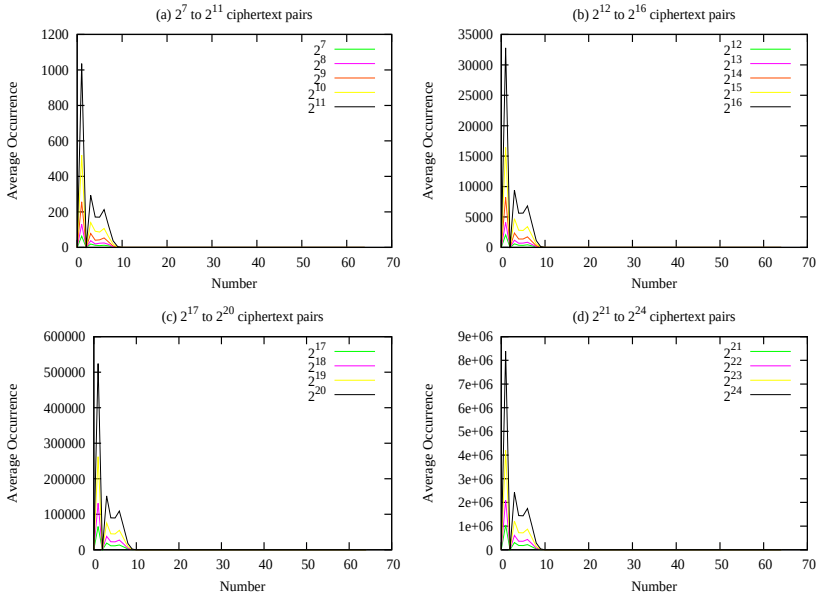

Figure B.1: Average Hamming Weight Distribution between ciphertext pairs for 1-round DES for 10 trials

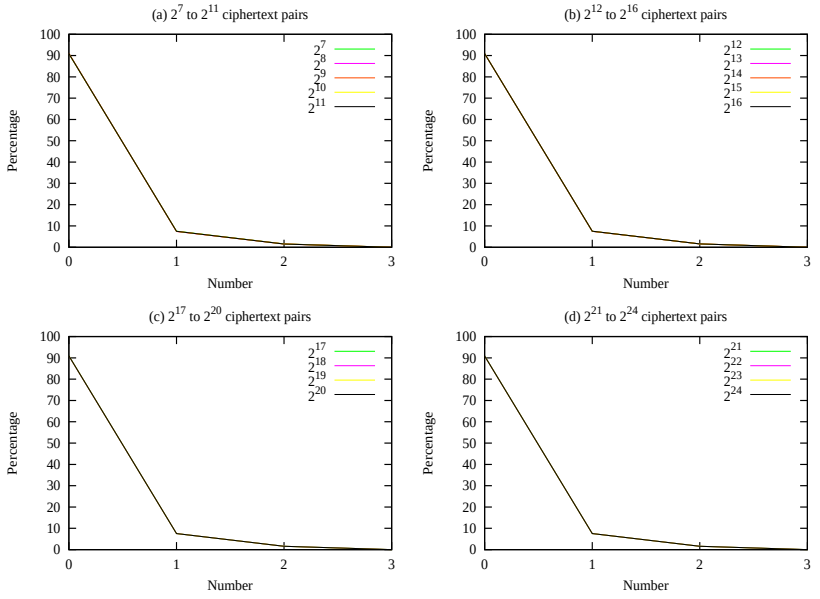

Figure B.2: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 1-round DES for 10 trials

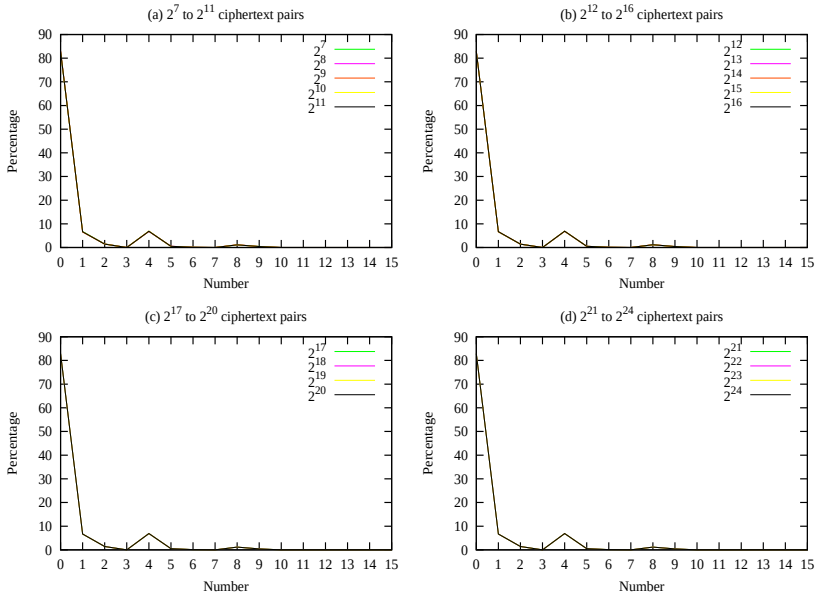

Figure B.3: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 1-round DES for 10 trials

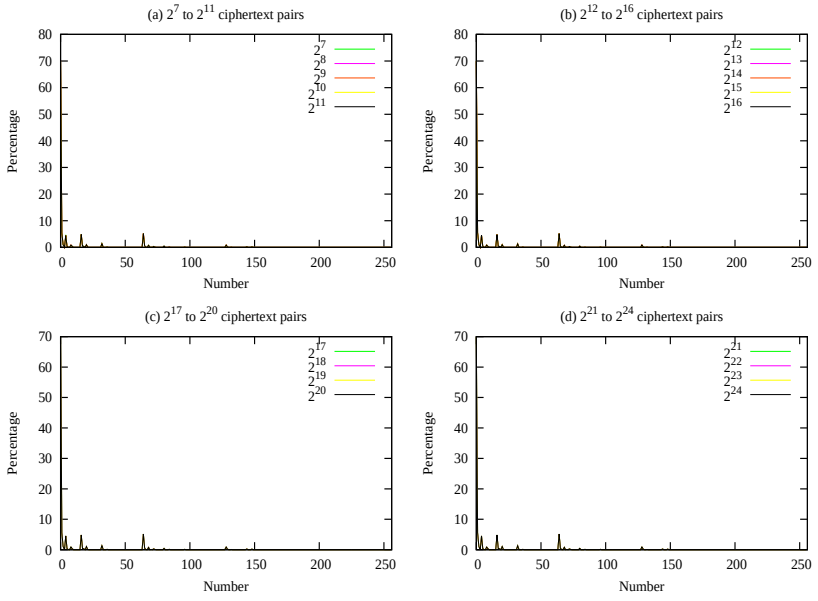

Figure B.4: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 1-round DES for 10 trials

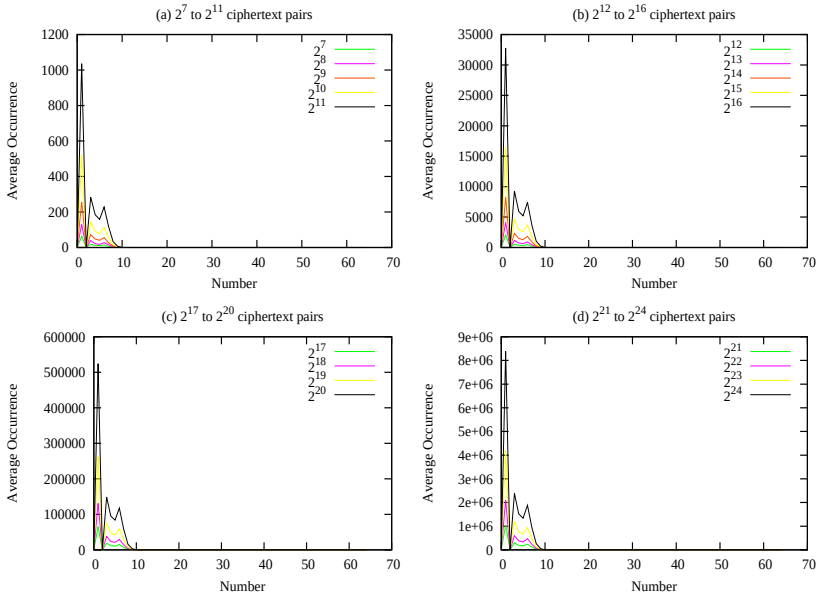

Figure B.5: Average Hamming Weight Distribution between ciphertext pairs for 1-round DES for 10 trials (Autofeeding)

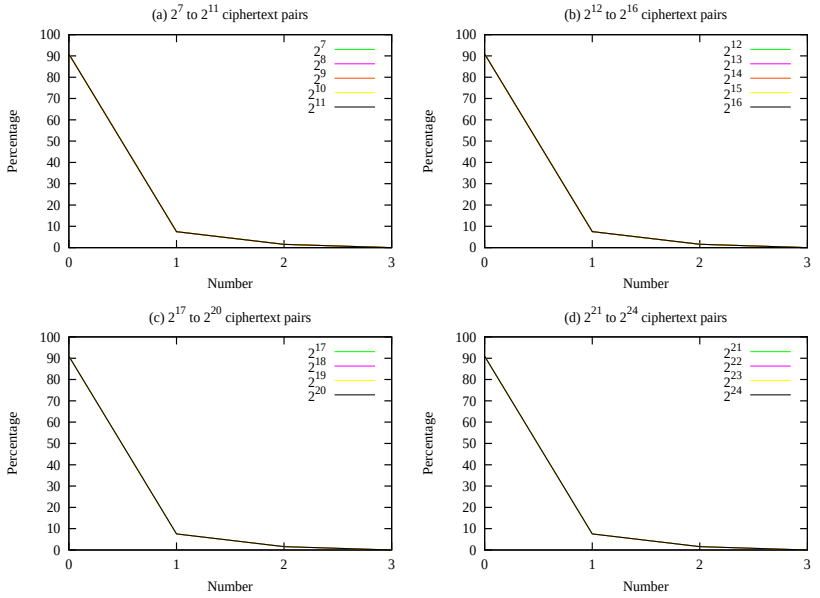

Figure B.6: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 1-round DES for 10 trials (Autofeeding)

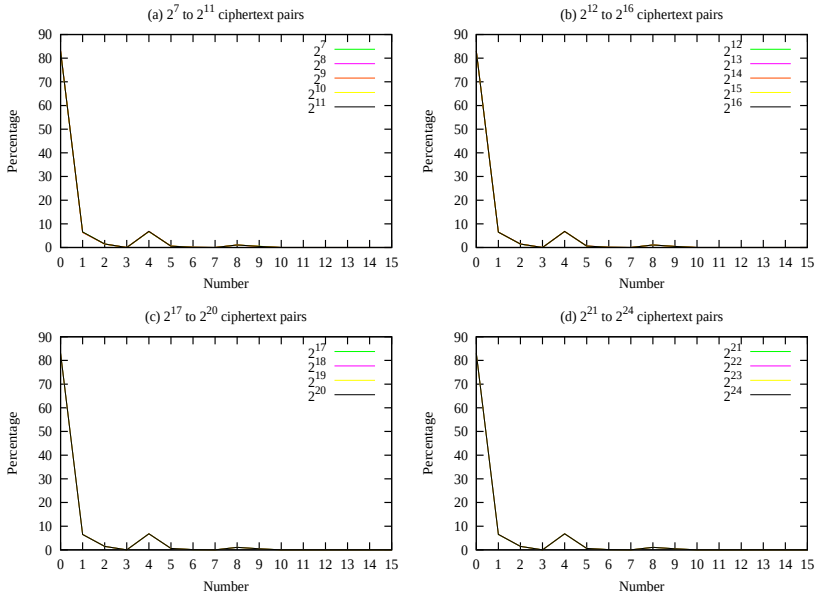

Figure B.7: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 1-round DES for 10 trials (Autofeeding)

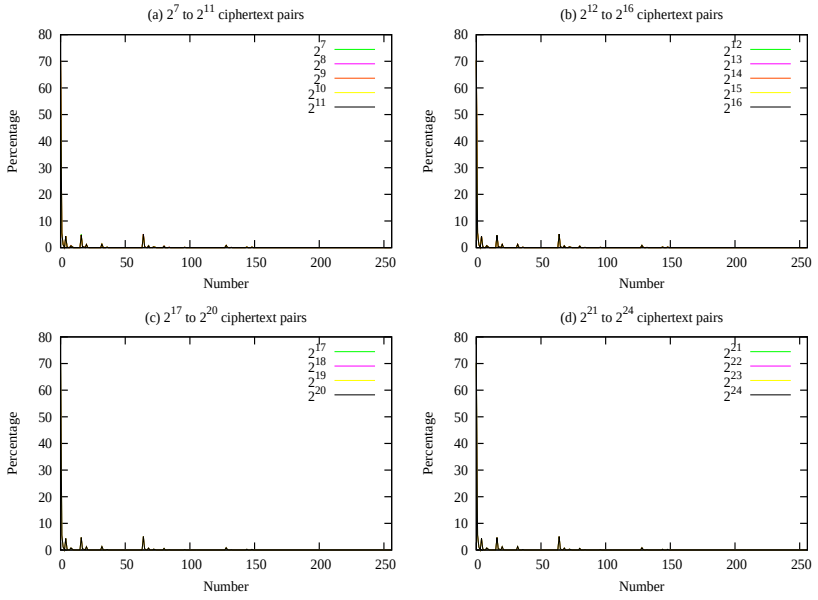

Figure B.8: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 1-round DES for 10 trials (Autofeeding)

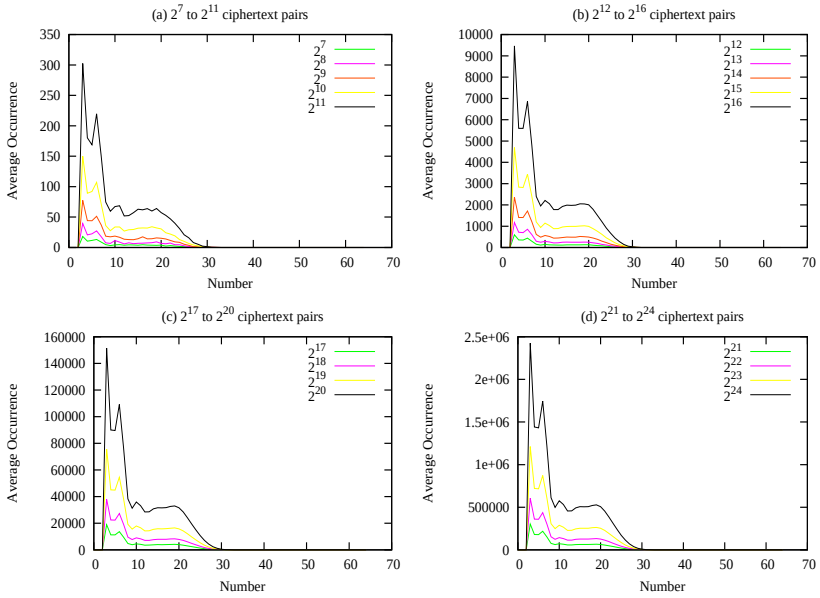

Figure B.9: Average Hamming Weight Distribution between ciphertext pairs for 2-round DES for 10 trials

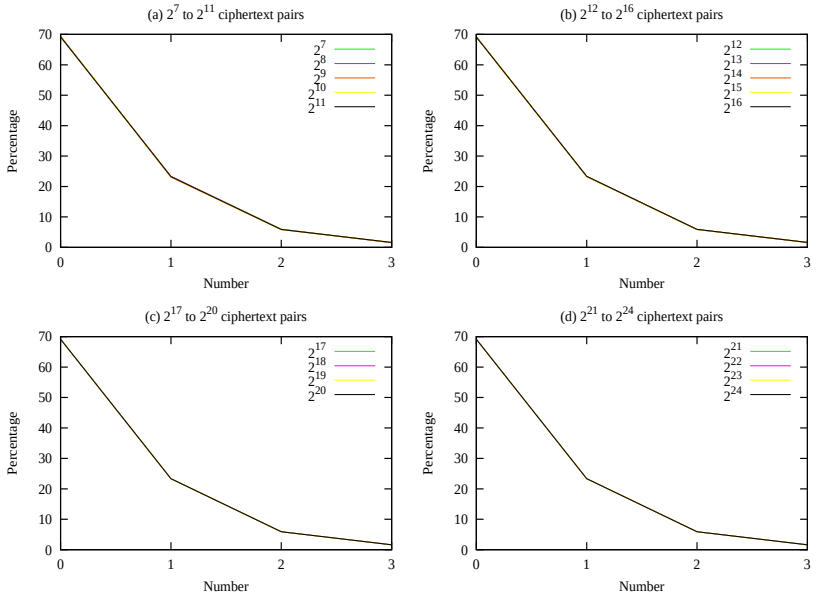

Figure B.10: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 2-round DES for 10 trials

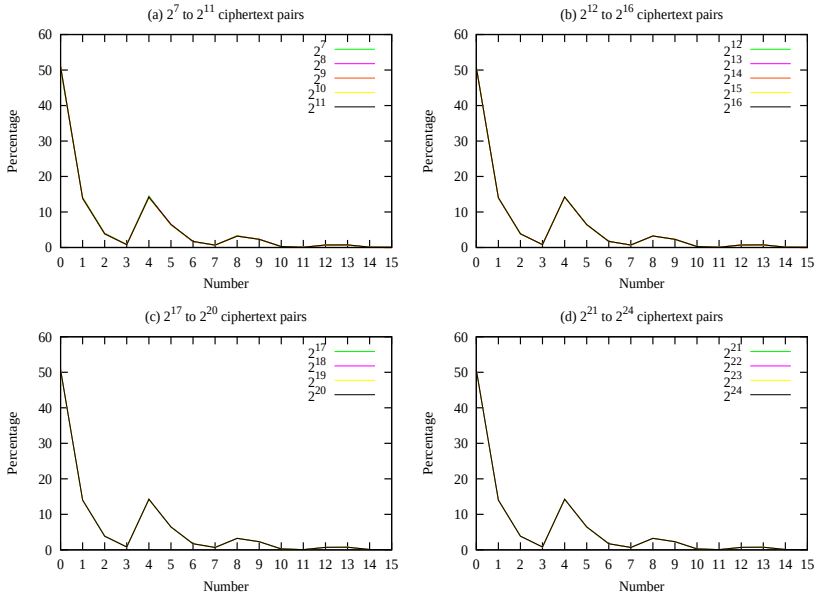

Figure B.11: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 2-round DES for 10 trials

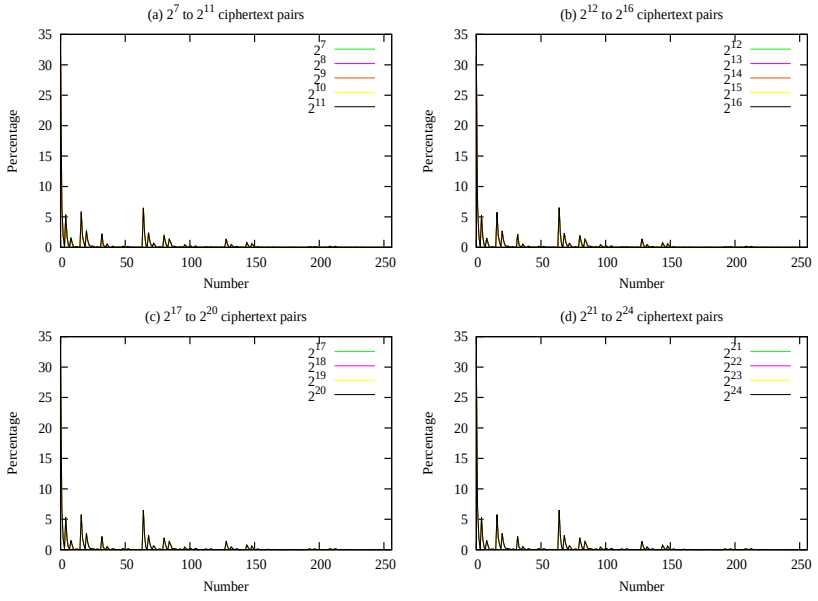

Figure B.12: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 2-round DES for 10 trials

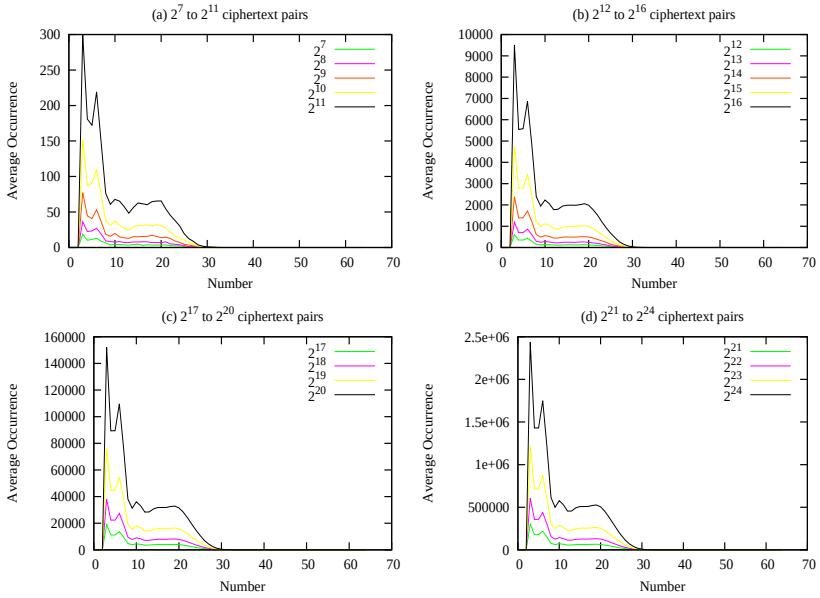

Figure B.13: Average Hamming Weight Distribution between ciphertext pairs for 2-round DES for 10 trials (Autofeeding)

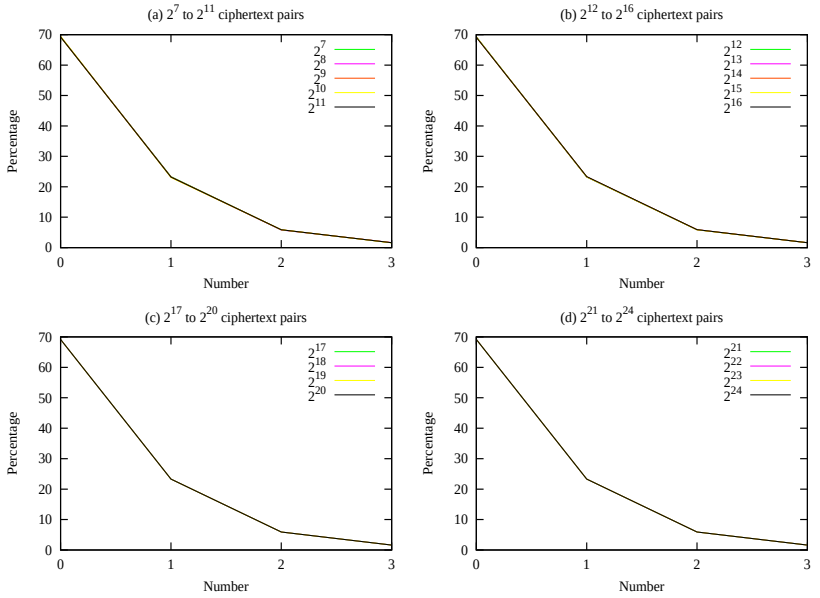

Figure B.14: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 2-round DES for 10 trials (Autofeeding)

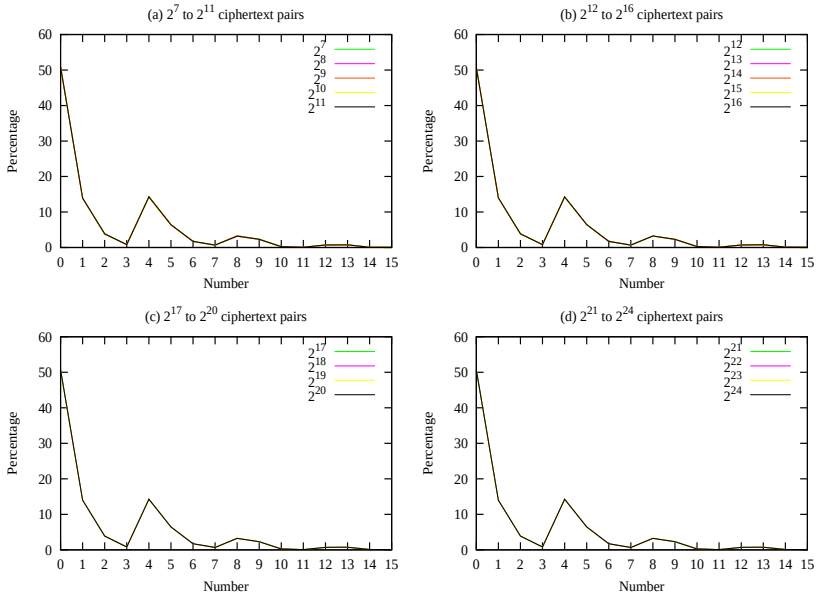

Figure B.15: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 2-round DES for 10 trials (Autofeeding)

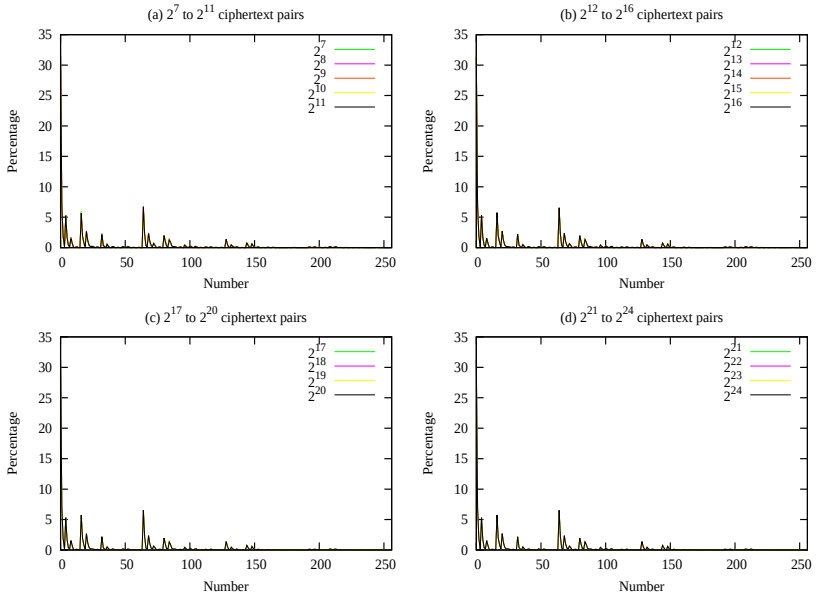

Figure B.16: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 2-round DES for 10 trials (Autofeeding)

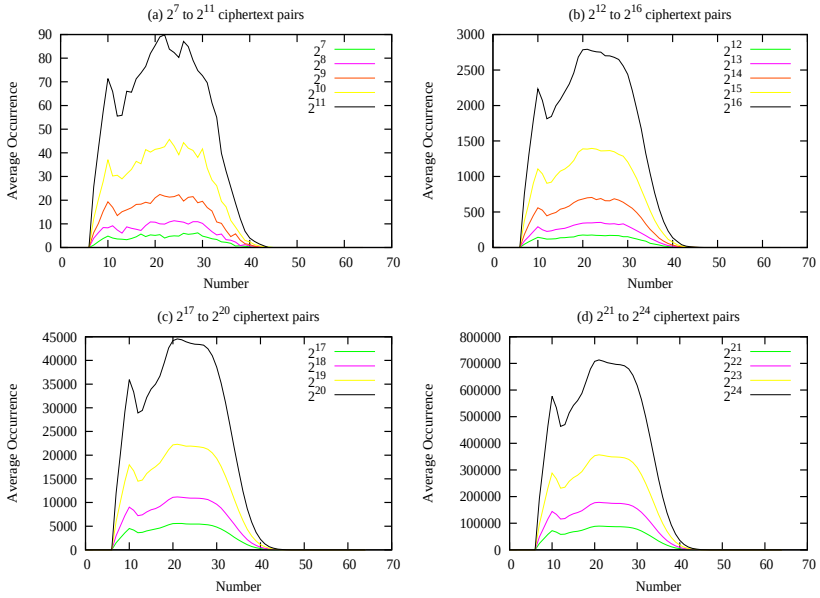

Figure B.17: Average Hamming Weight Distribution between ciphertext pairs for 3-round DES for 10 trials

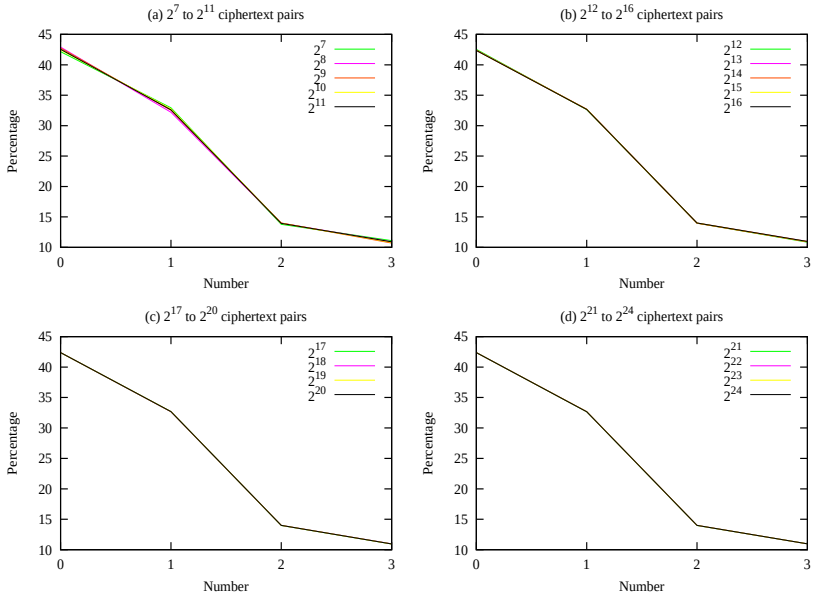

Figure B.18: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 3-round DES for 10 trials

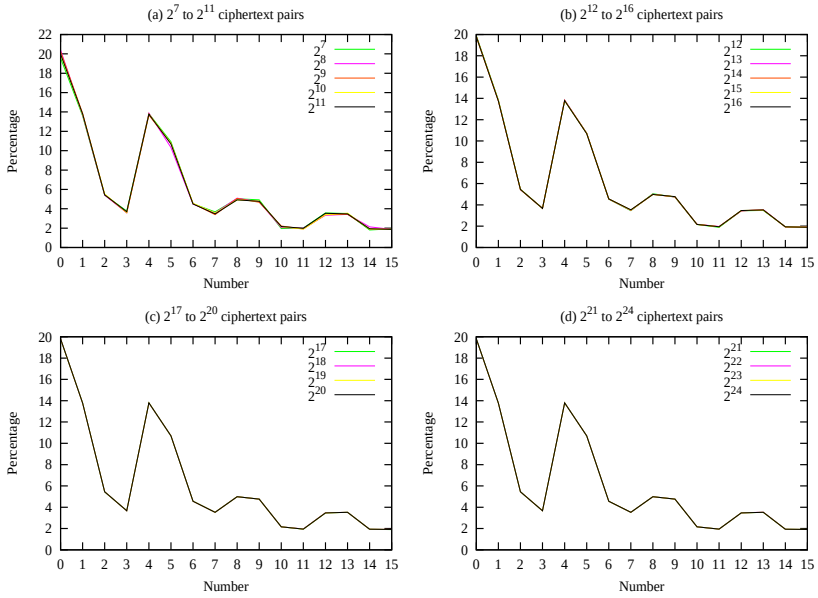

Figure B.19: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 3-round DES for 10 trials

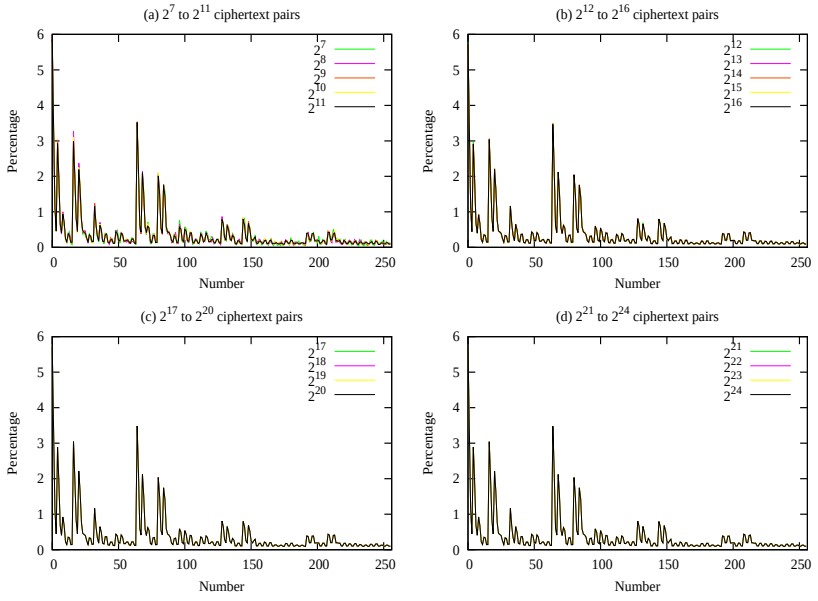

Figure B.20: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 3-round DES for 10 trials

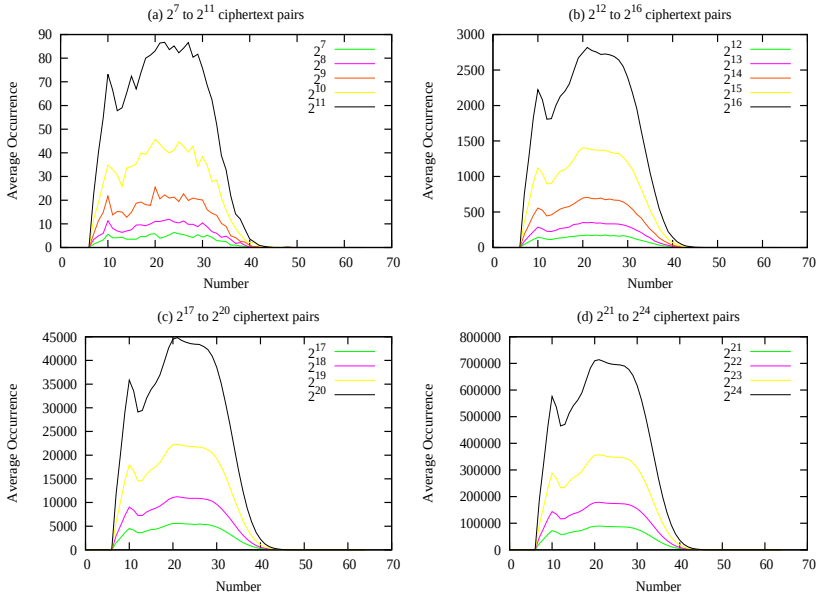

Figure B.21: Average Hamming Weight Distribution between ciphertext pairs for 3-round DES for 10 trials (Autofeeding)

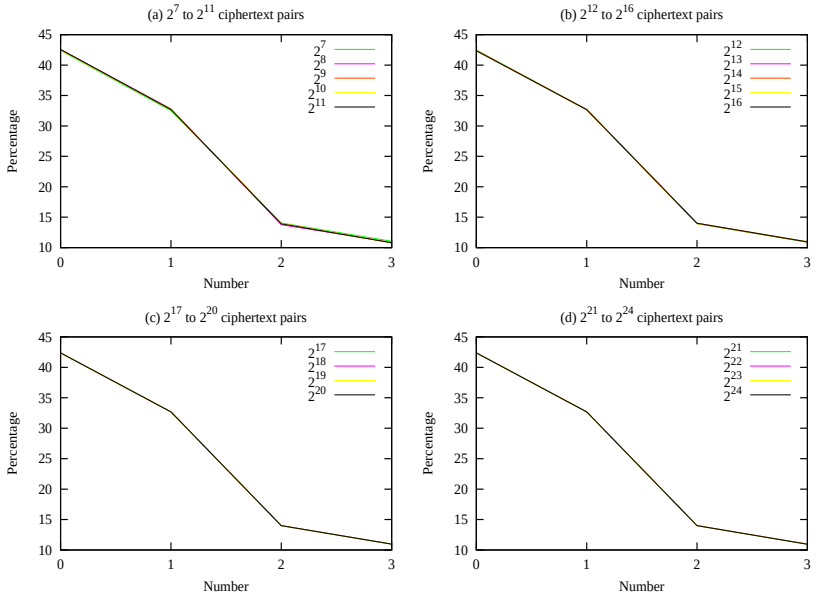

Figure B.22: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 3-round DES for 10 trials (Autofeeding)

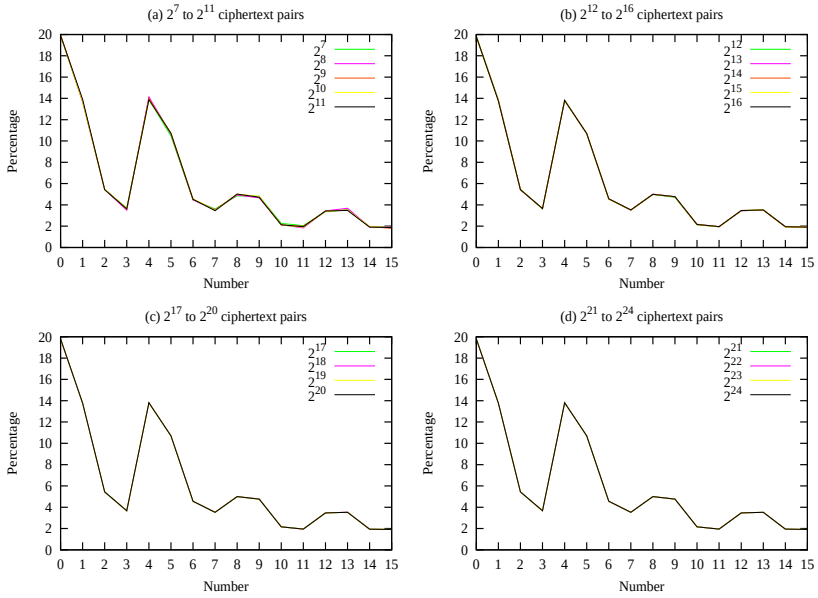

Figure B.23: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 3-round DES for 10 trials (Autofeeding)

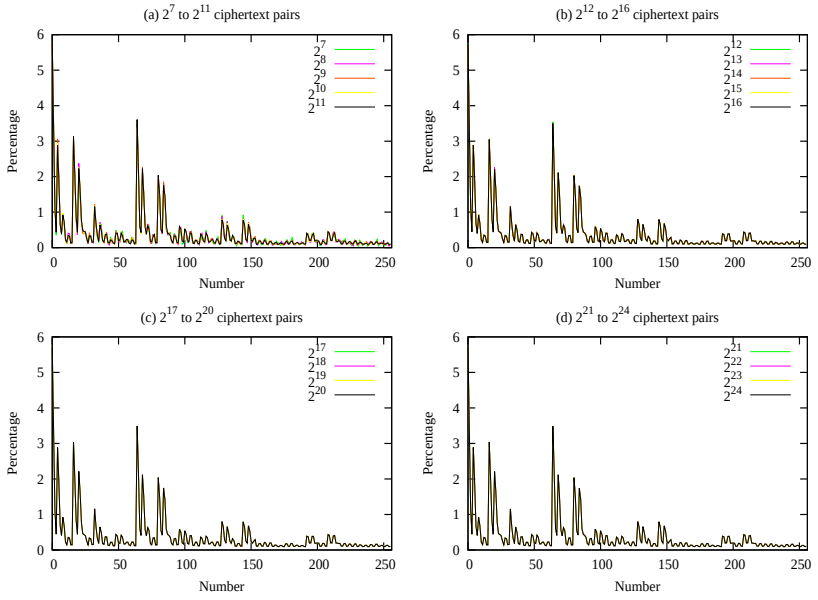

Figure B.24: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 3-round DES for 10 trials (Autofeeding)

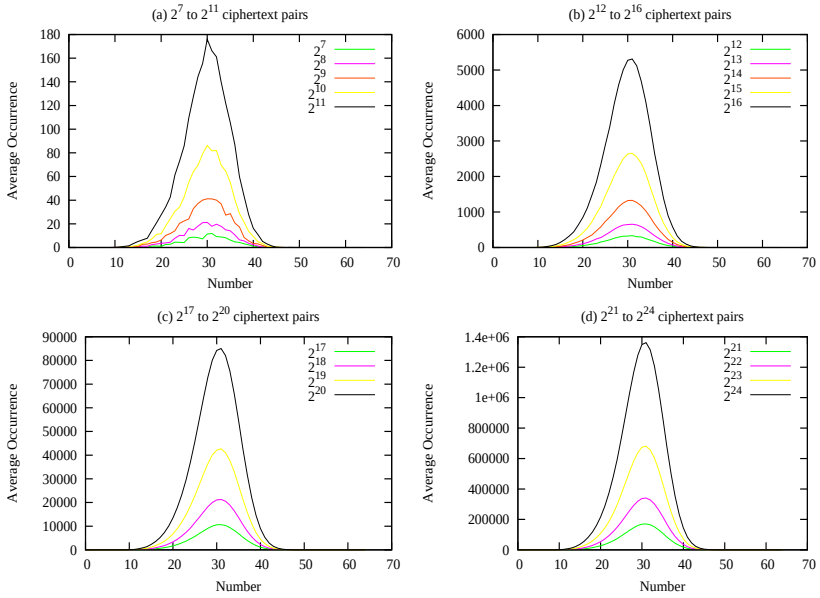

Figure B.25: Average Hamming Weight Distribution between ciphertext pairs for 4-round DES for 10 trials

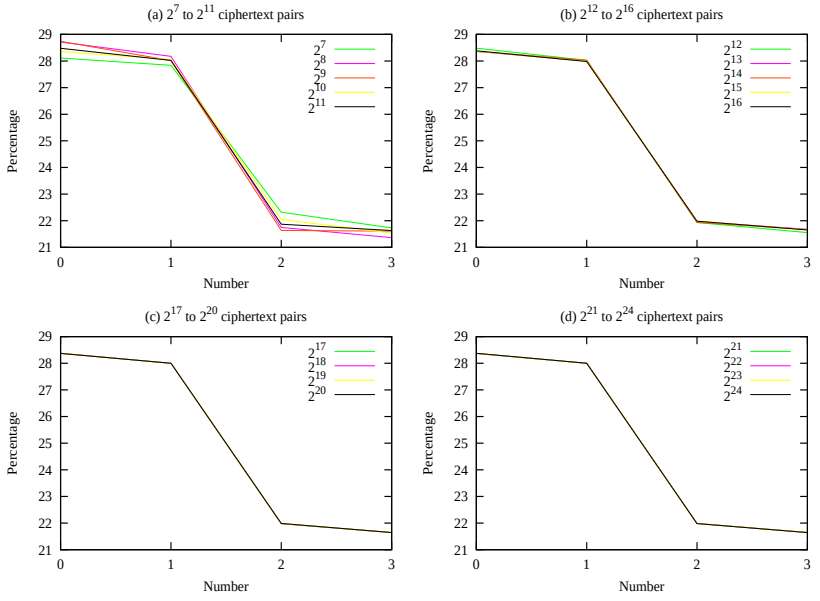

Figure B.26: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 4-round DES for 10 trials

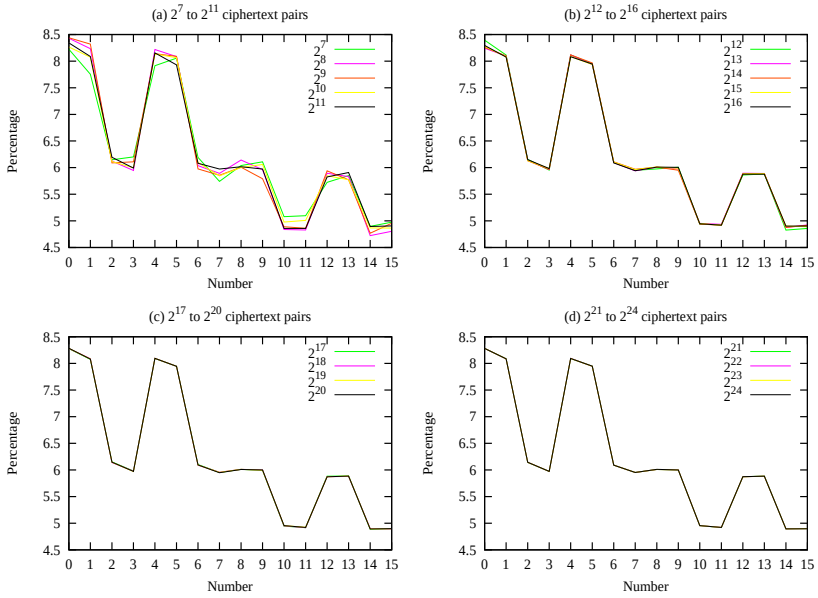

Figure B.27: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 4-round DES for 10 trials

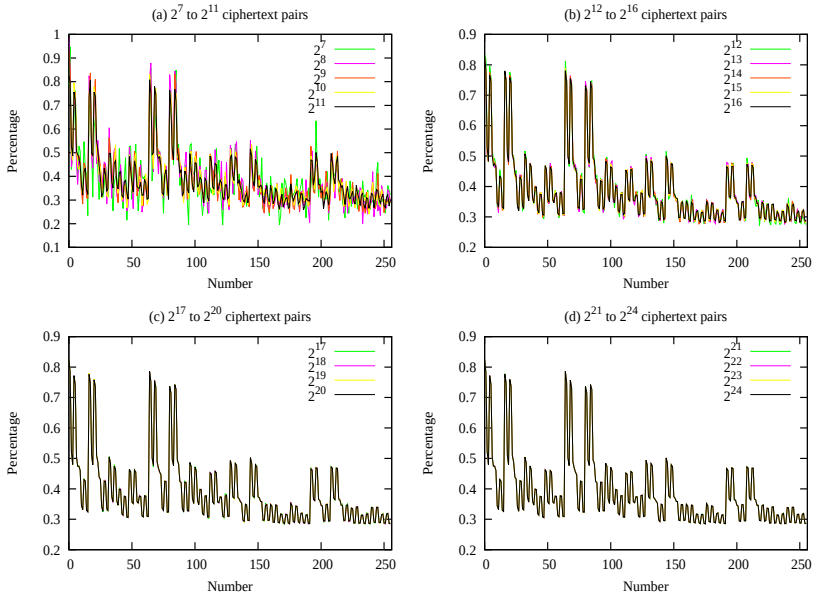

Figure B.28: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 4-round DES for 10 trials

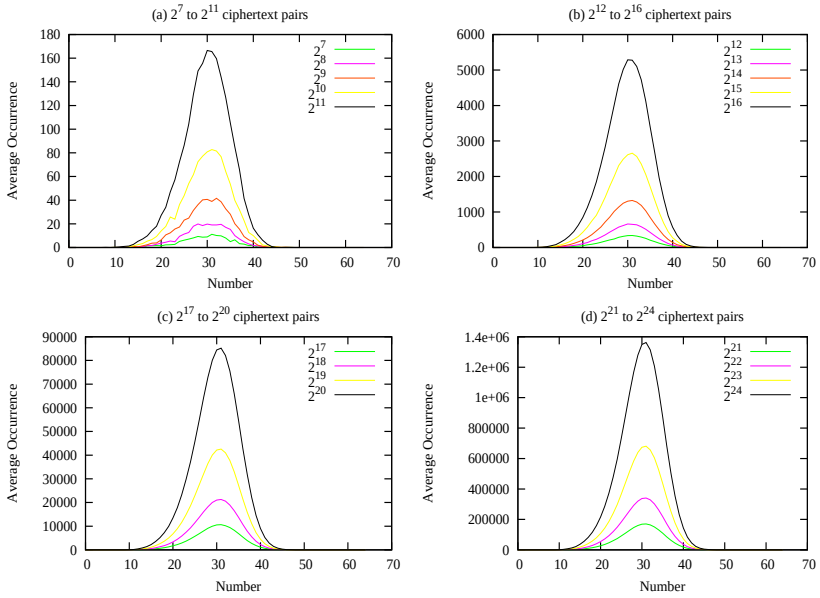

Figure B.29: Average Hamming Weight Distribution between ciphertext pairs for 4-round DES for 10 trials (Autofeeding)

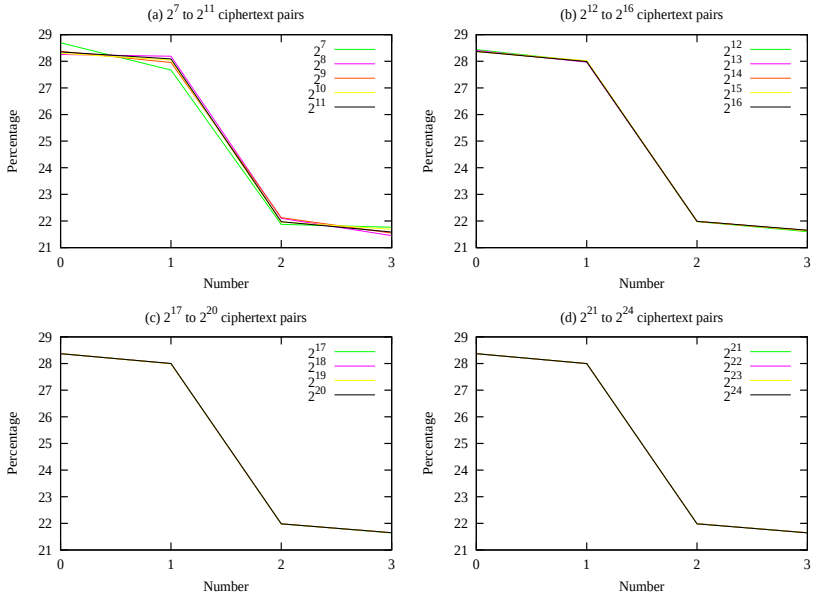

Figure B.30: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 4-round DES for 10 trials (Autofeeding)

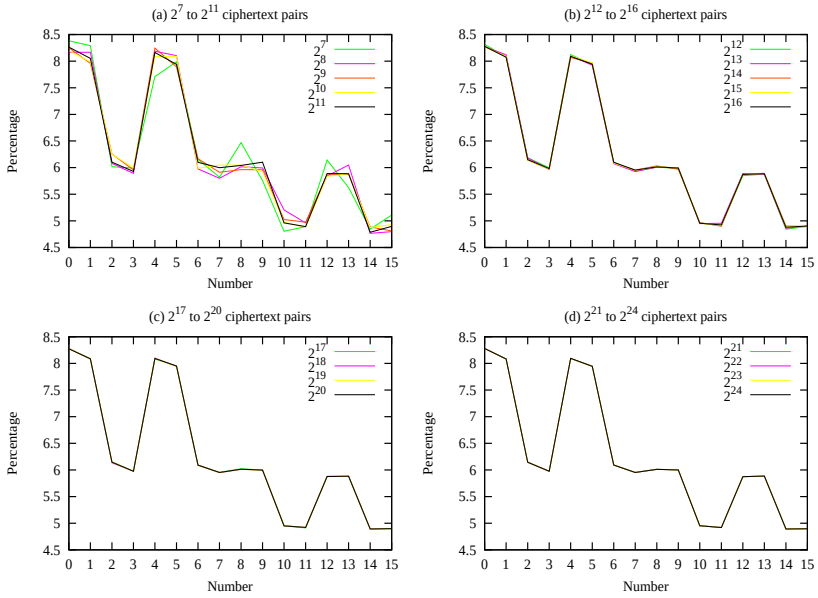

Figure B.31: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 4-round DES for 10 trials (Autofeeding)

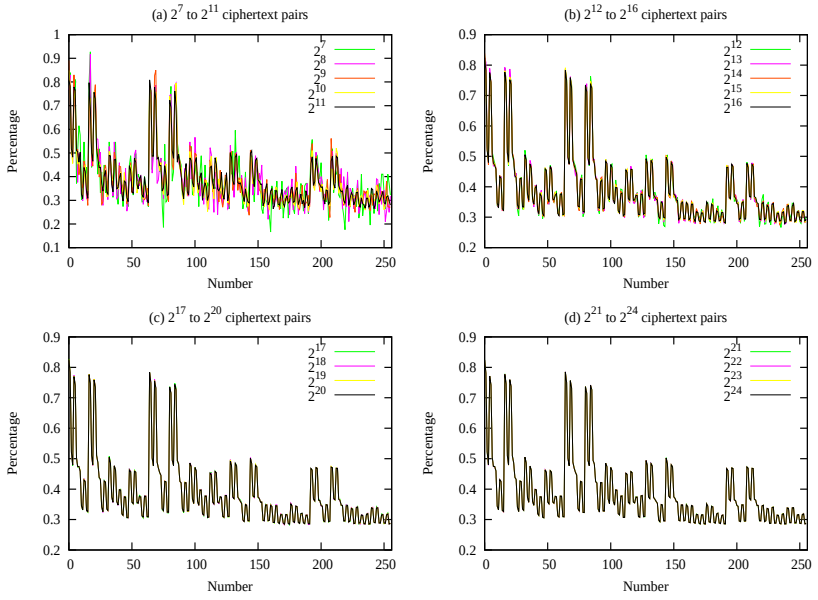

Figure B.32: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 4-round DES for 10 trials (Autofeeding)

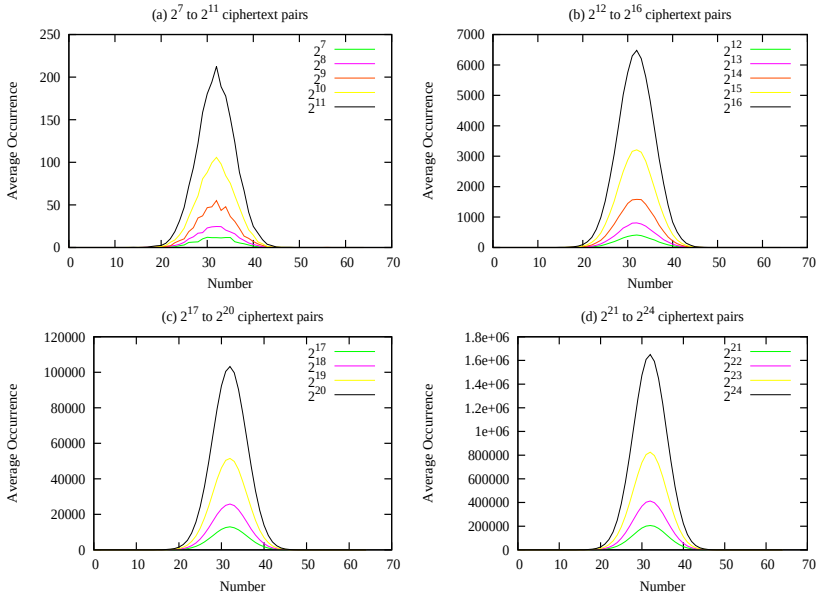

Figure B.33: Average Hamming Weight Distribution between ciphertext pairs for 5-round DES for 10 trials

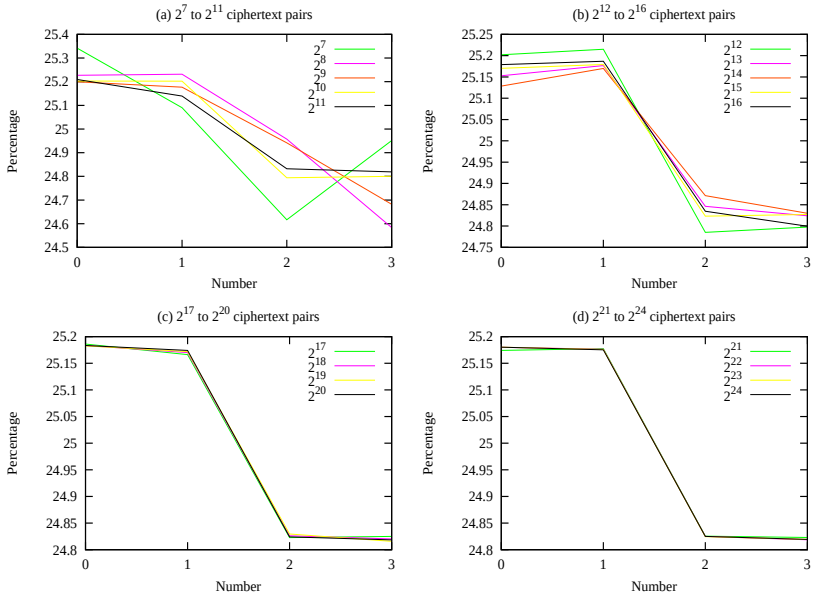

Figure B.34: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 5-round DES for 10 trials

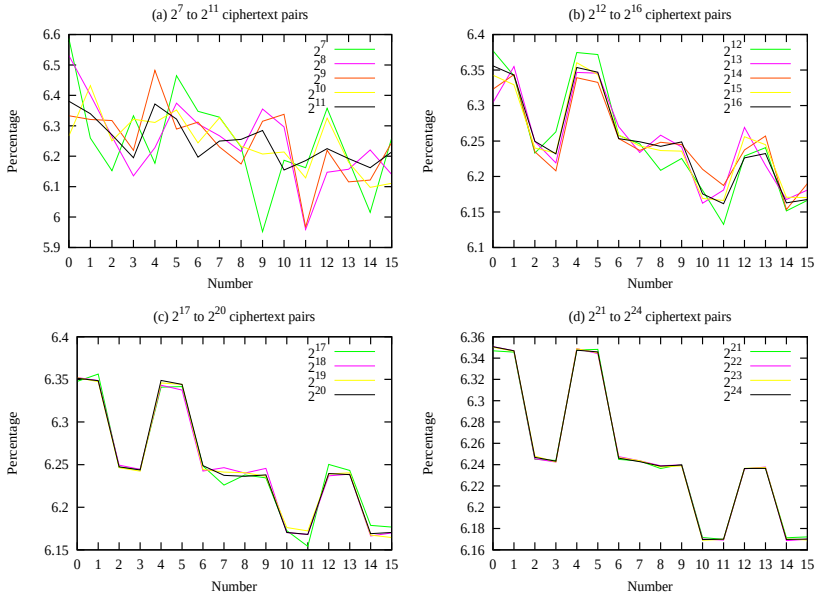

Figure B.35: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 5-round DES for 10 trials

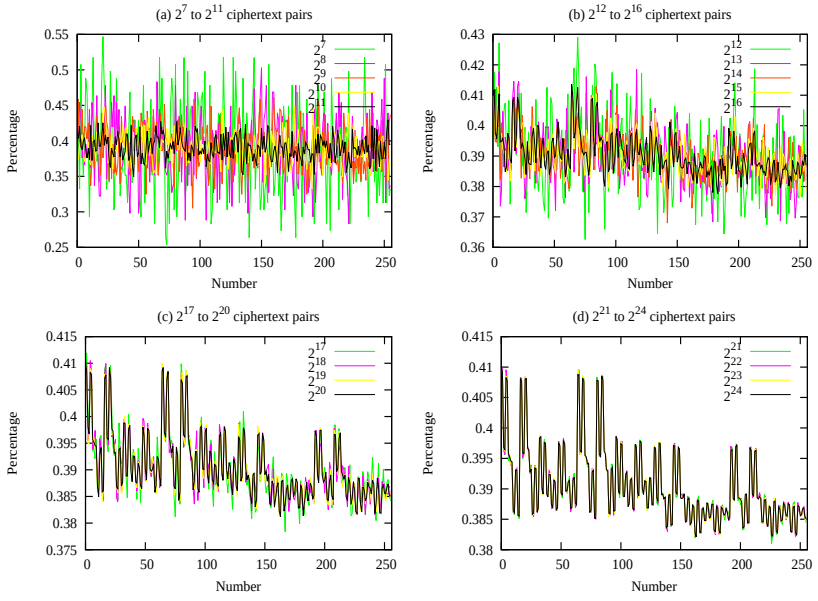

Figure B.36: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 5-round DES for 10 trials

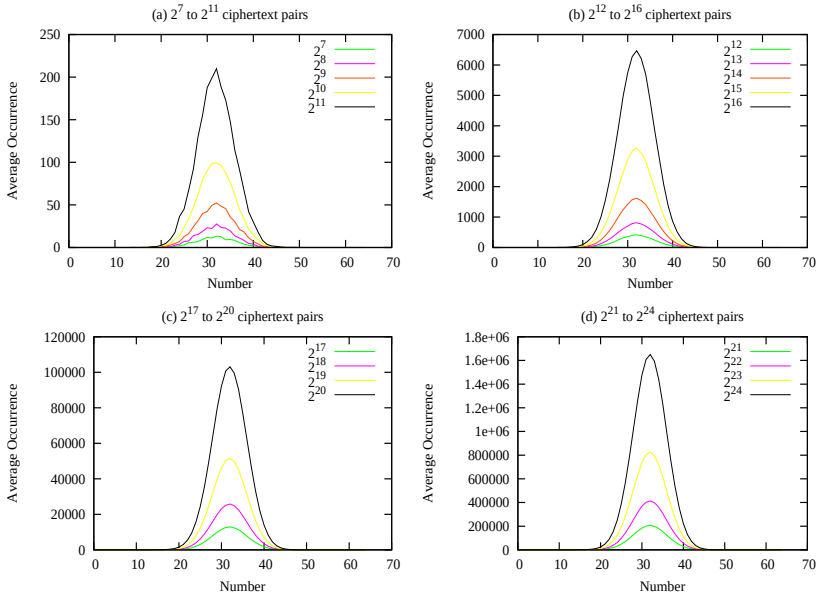

Figure B.37: Average Hamming Weight Distribution between ciphertext pairs for 5-round DES for 10 trials (Autofeeding)

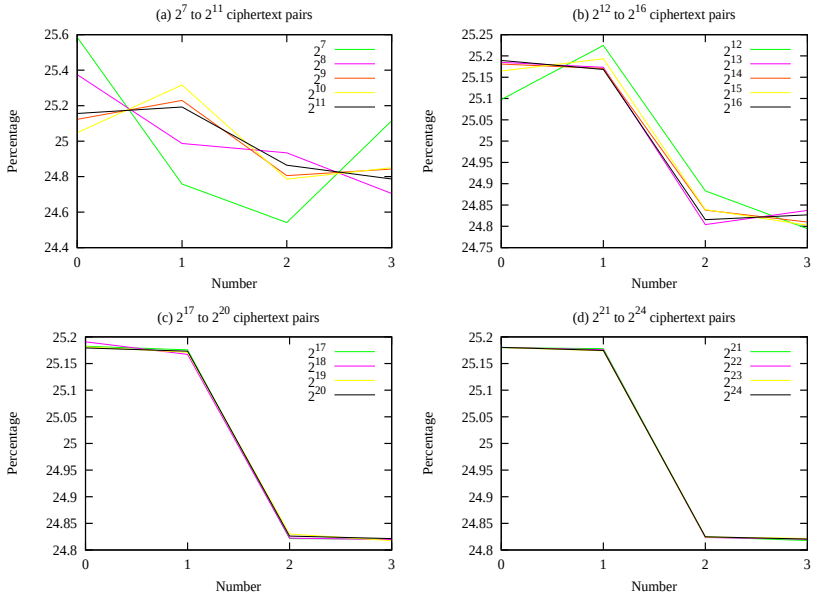

Figure B.38: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 5-round DES for 10 trials (Autofeeding)

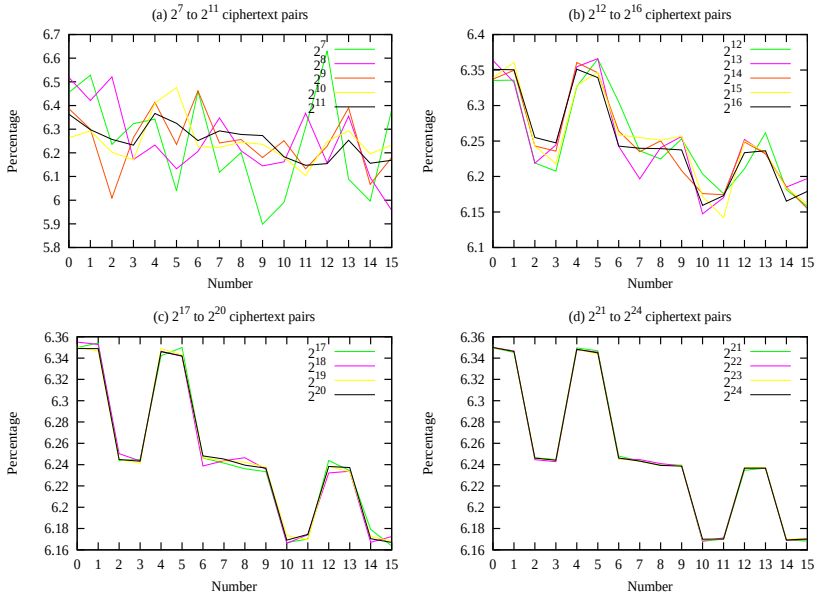

Figure B.39: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 5-round DES for 10 trials (Autofeeding)

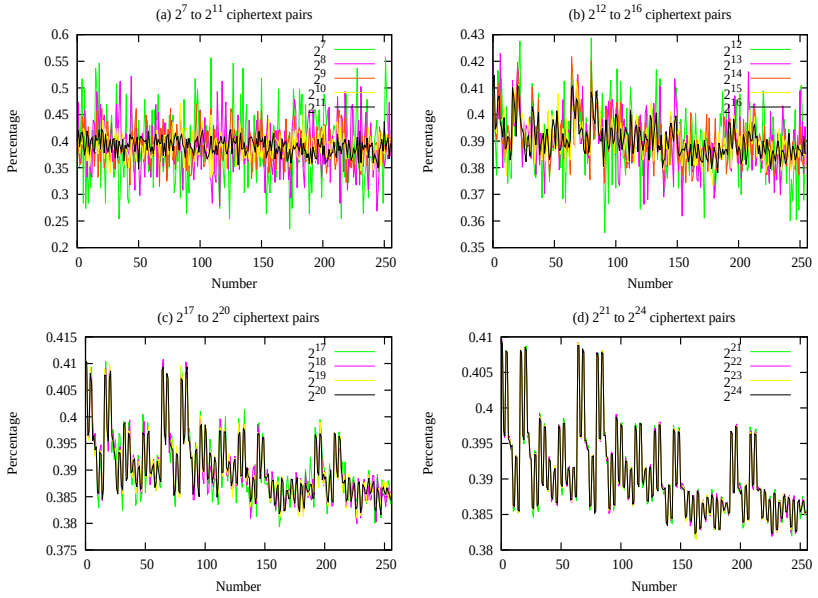

Figure B.40: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 5-round DES for 10 trials (Autofeeding)

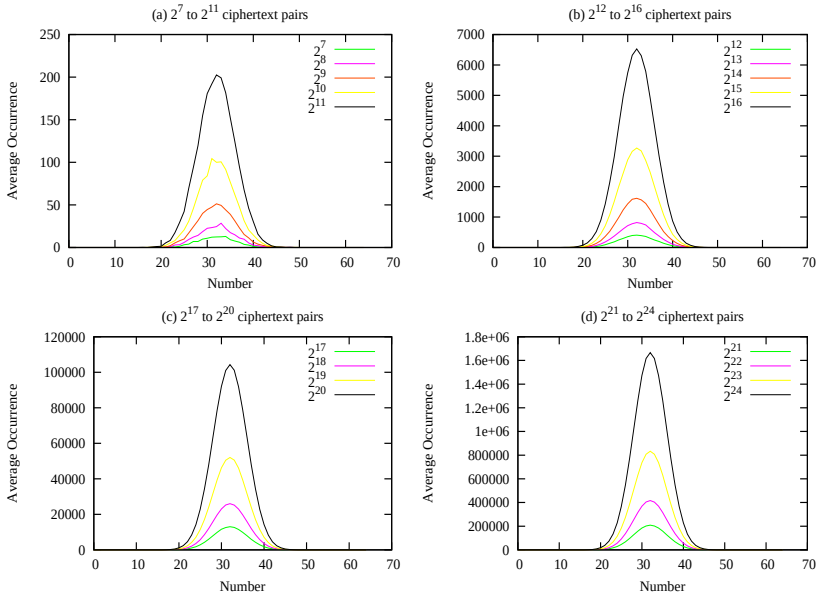

Figure B.41: Average Hamming Weight Distribution between ciphertext pairs for 6-round DES for 10 trials

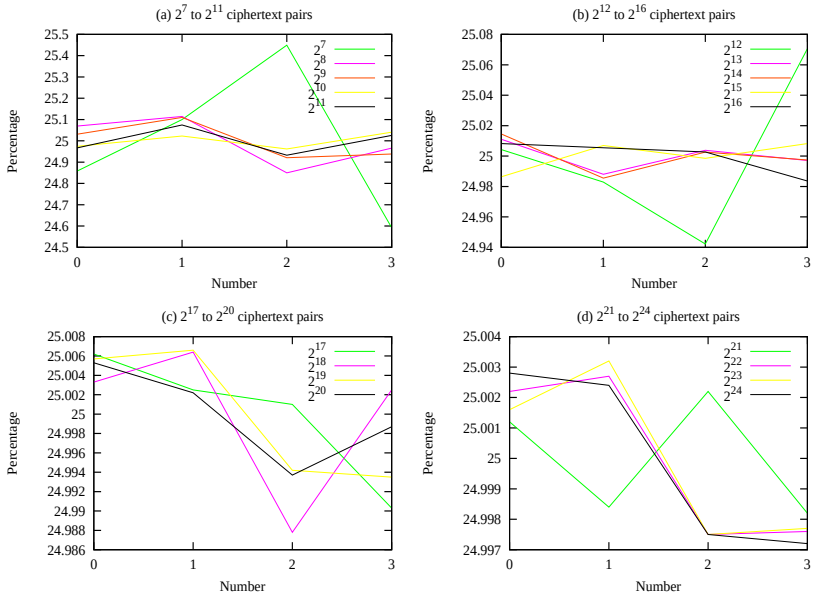

Figure B.42: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 6-round DES for 10 trials

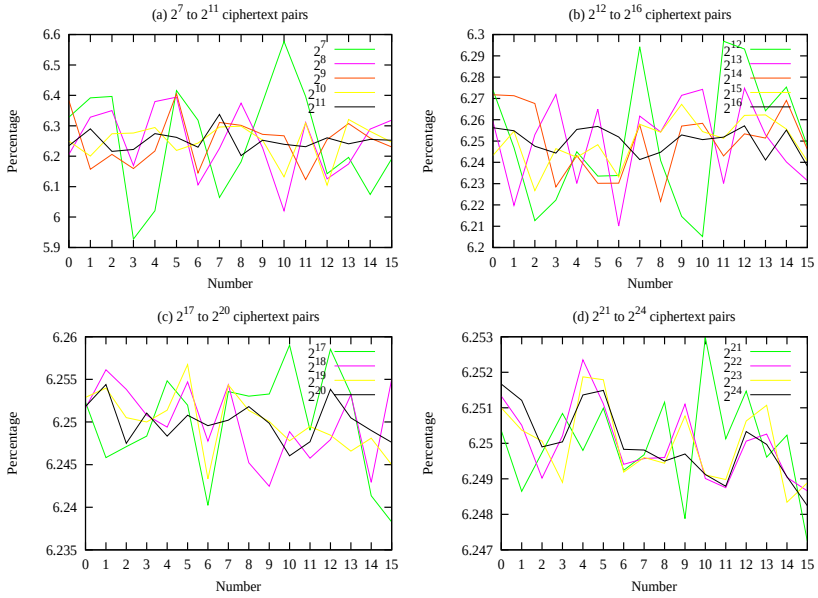

Figure B.43: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 6-round DES for 10 trials

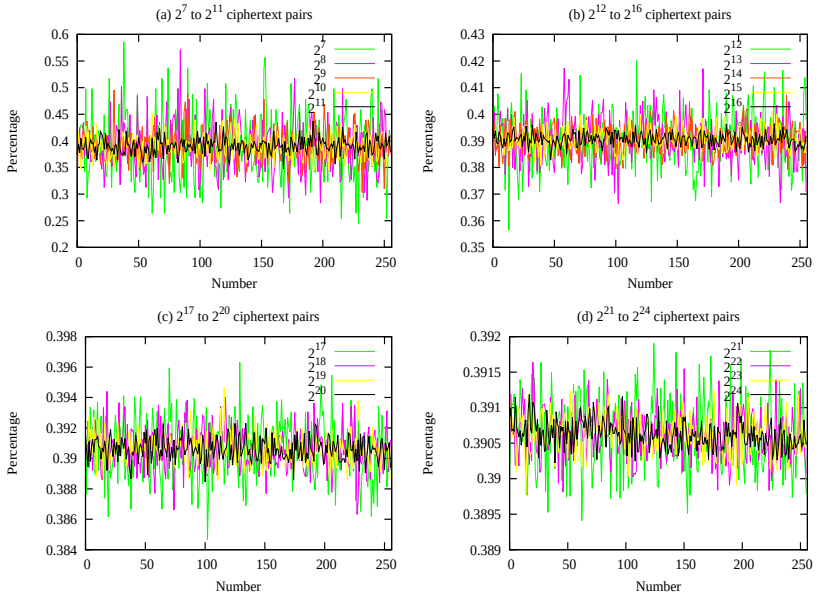

Figure B.44: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 6-round DES for 10 trials

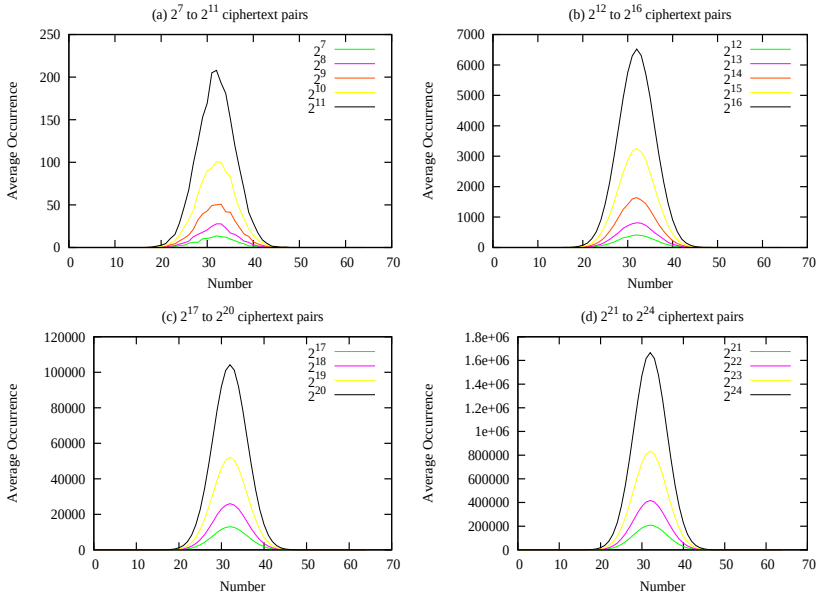

Figure B.45: Average Hamming Weight Distribution between ciphertext pairs for 6-round DES for 10 trials (Autofeeding)

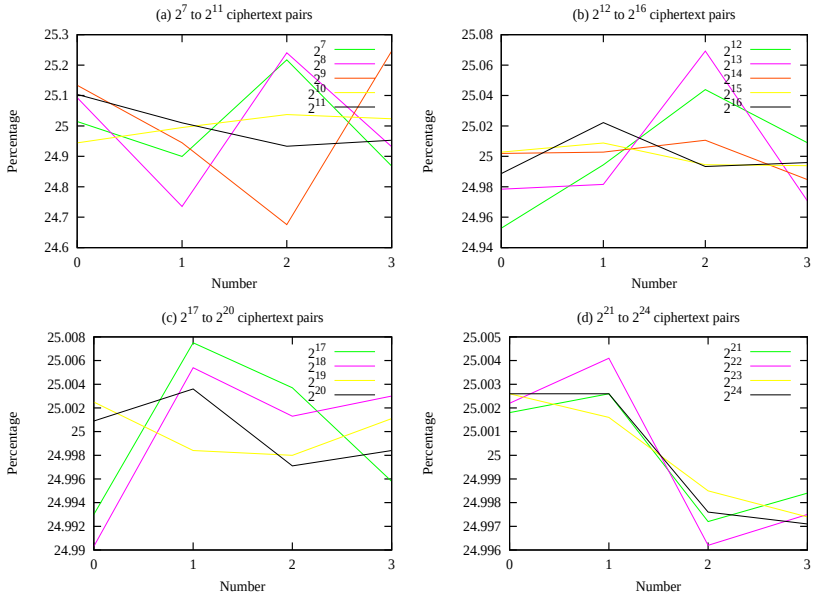

Figure B.46: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 6-round DES for 10 trials (Autofeeding)

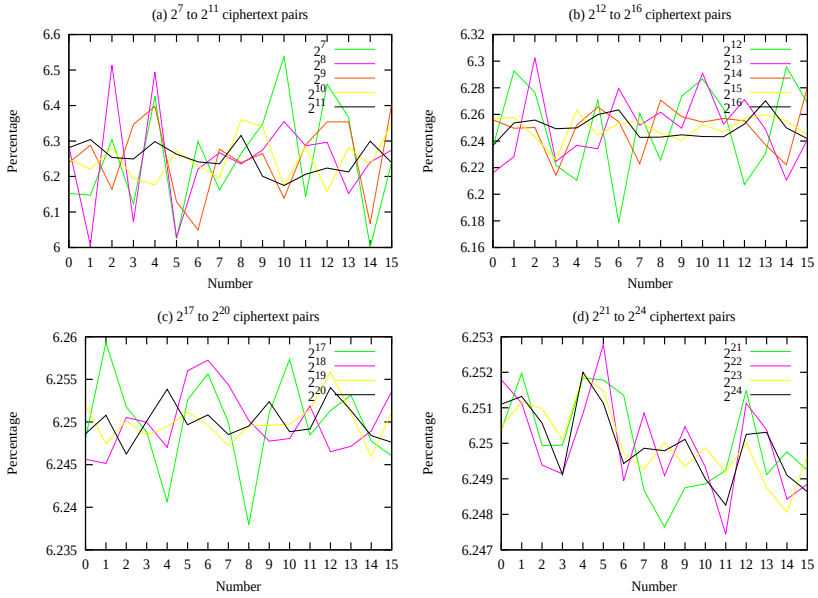

Figure B.47: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 6-round DES for 10 trials (Autofeeding)

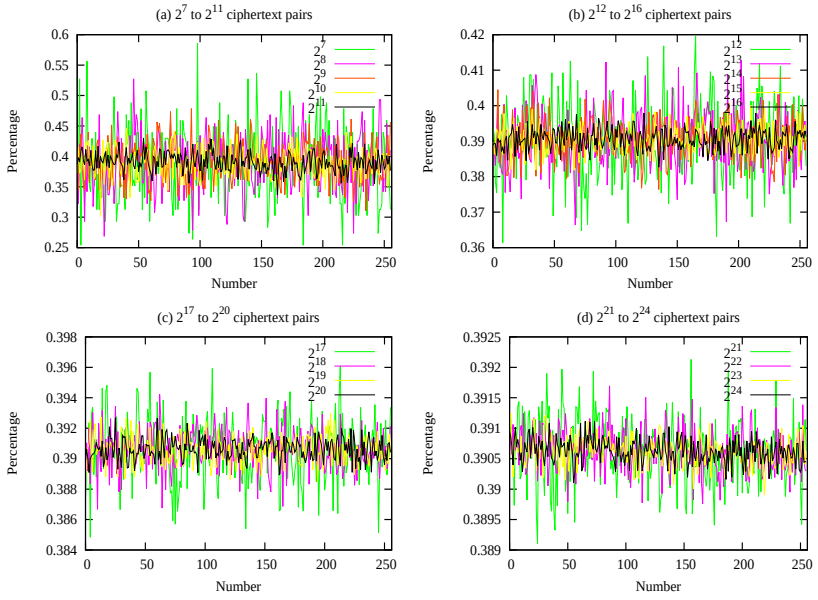

Figure B.48: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 6-round DES for 10 trials (Autofeeding)

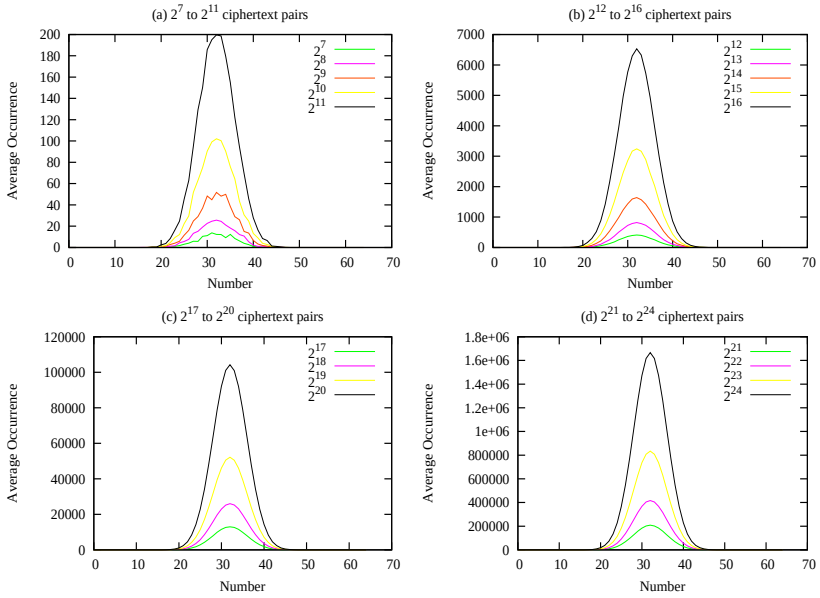

Figure B.49: Average Hamming Weight Distribution between ciphertext pairs for 7-round DES for 10 trials

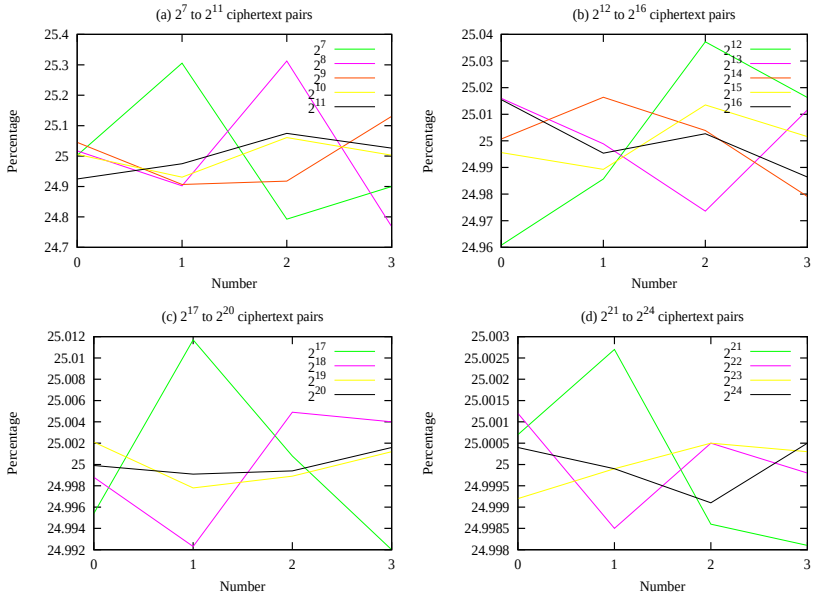

Figure B.50: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 7-round DES for 10 trials

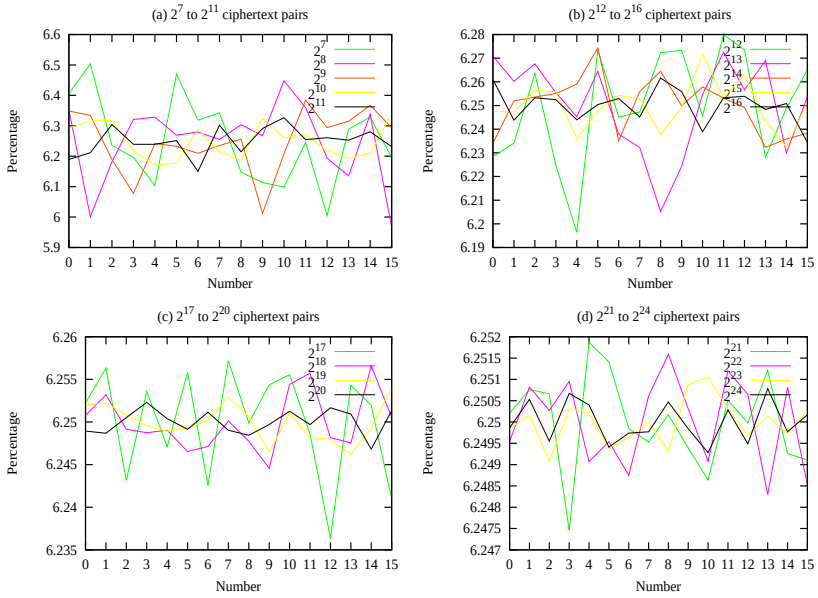

Figure B.51: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 7-round DES for 10 trials

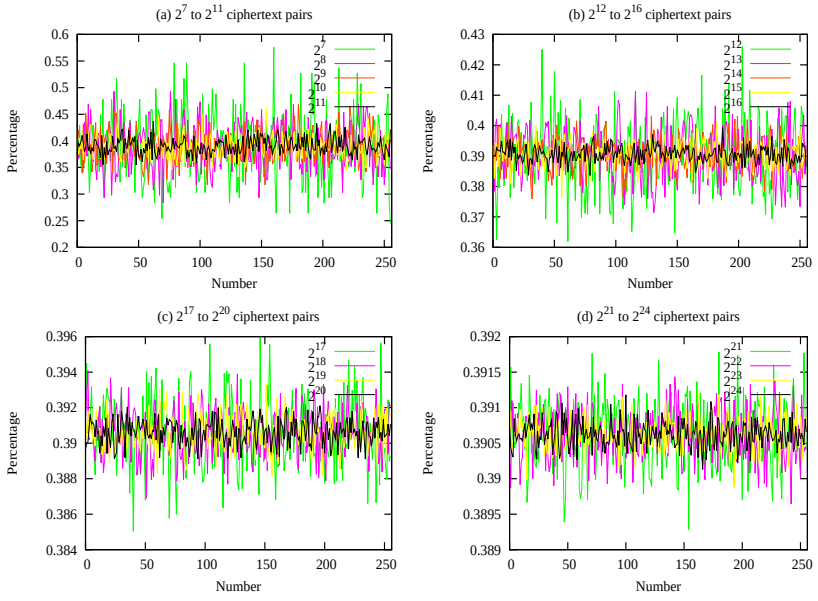

Figure B.52: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 7-round DES for 10 trials

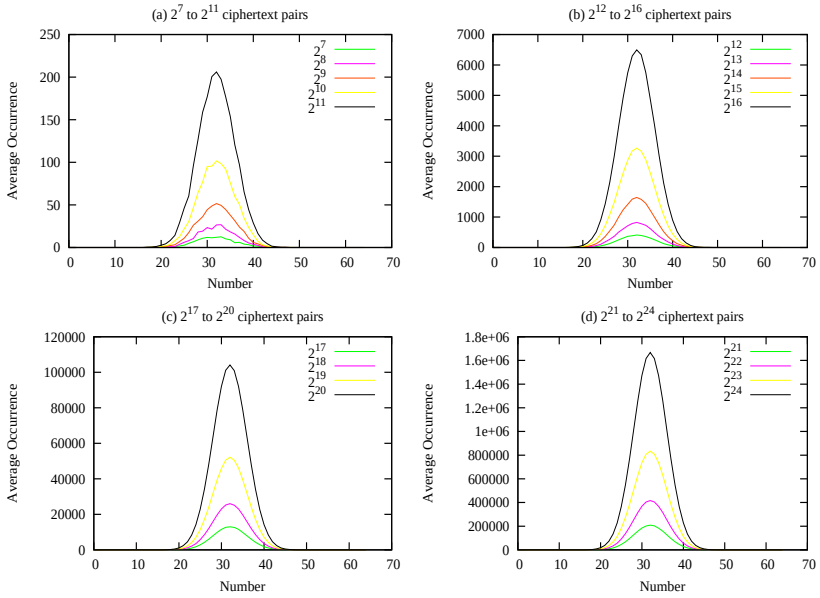

Figure B.53: Average Hamming Weight Distribution between ciphertext pairs for 7-round DES for 10 trials (Autofeeding)

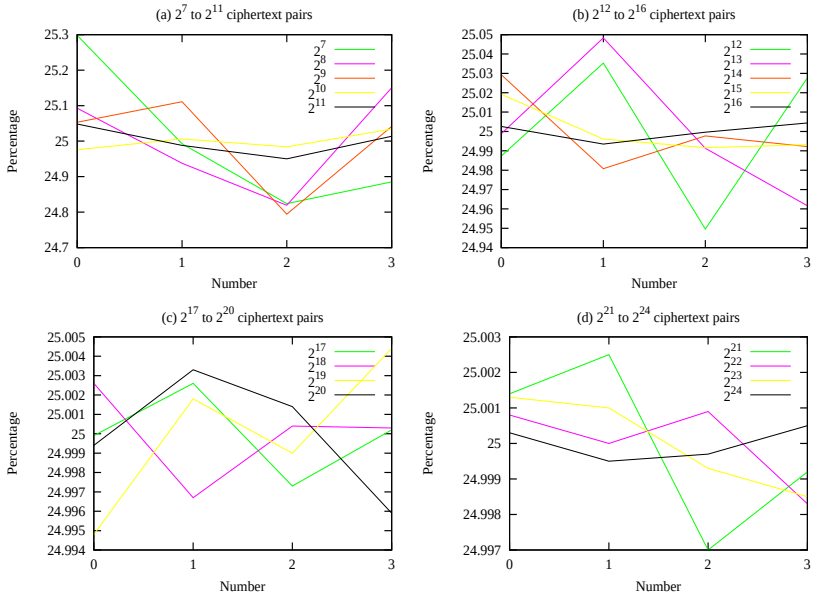

Figure B.54: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 7-round DES for 10 trials (Autofeeding)

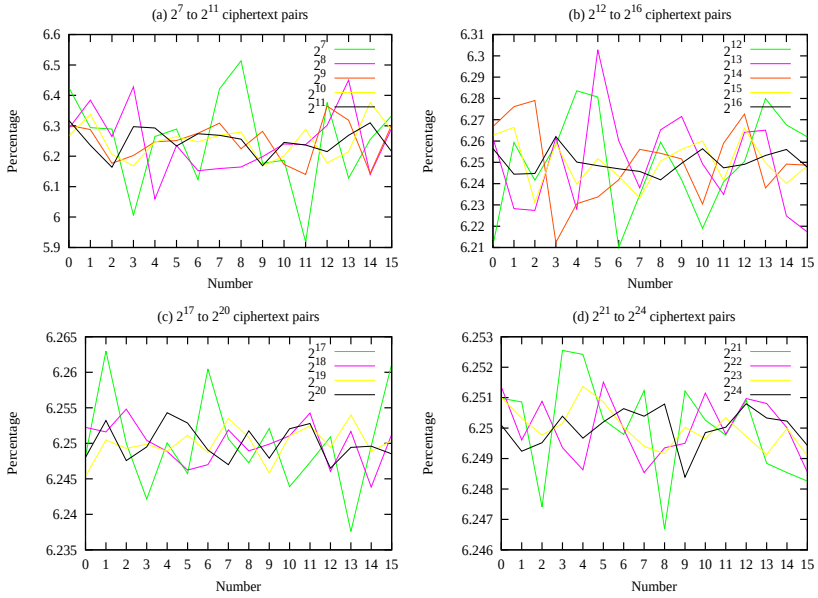

Figure B.55: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 7-round DES for 10 trials (Autofeeding)

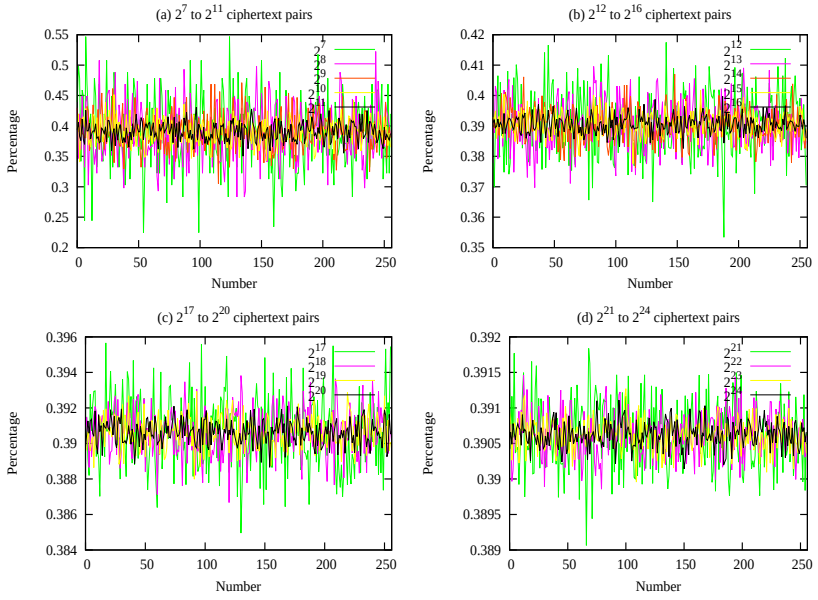

Figure B.56: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 7-round DES for 10 trials (Autofeeding)

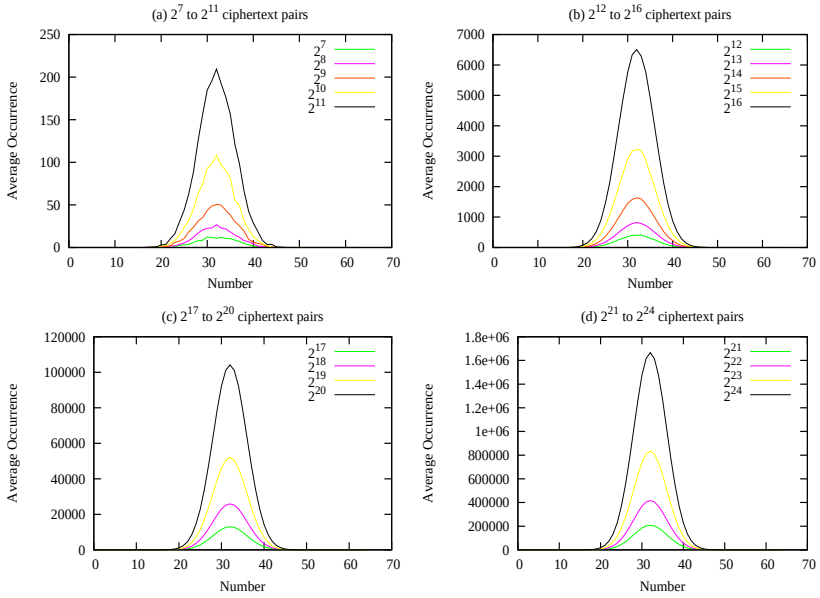

Figure B.57: Average Hamming Weight Distribution between ciphertext pairs for 8-round DES for 10 trials

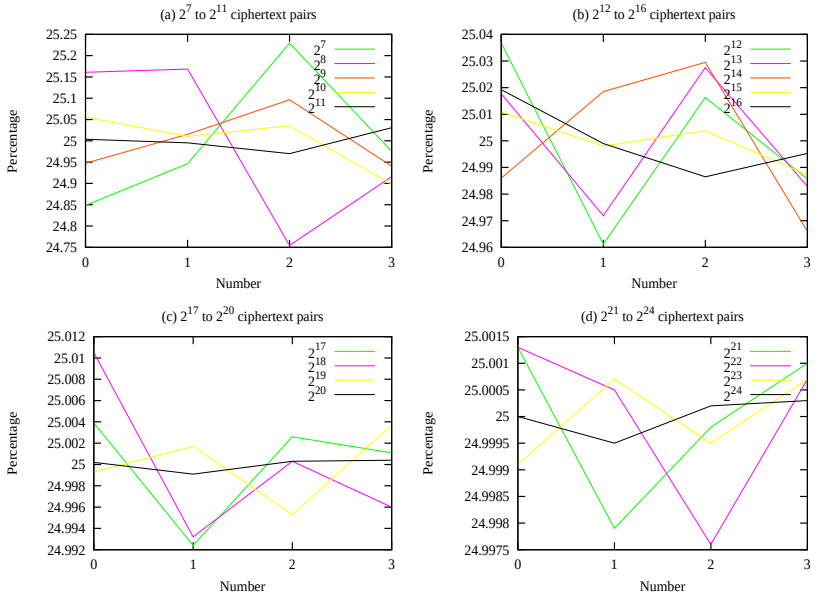

Figure B.58: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 8-round DES for 10 trials

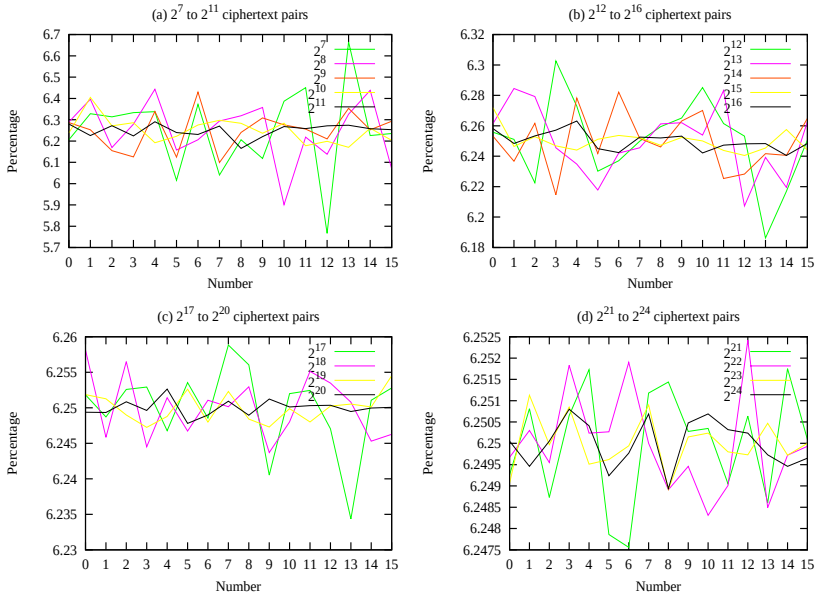

Figure B.59: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 8-round DES for 10 trials

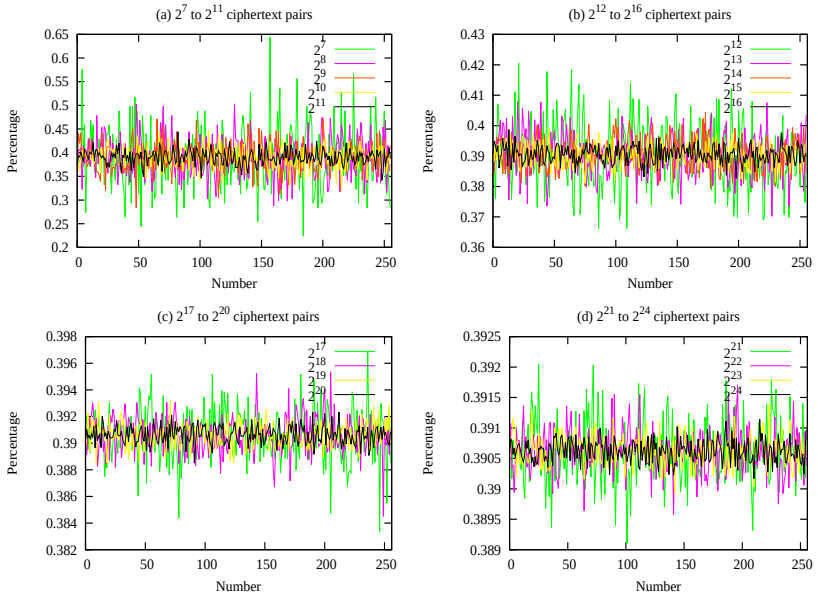

Figure B.60: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 8-round DES for 10 trials

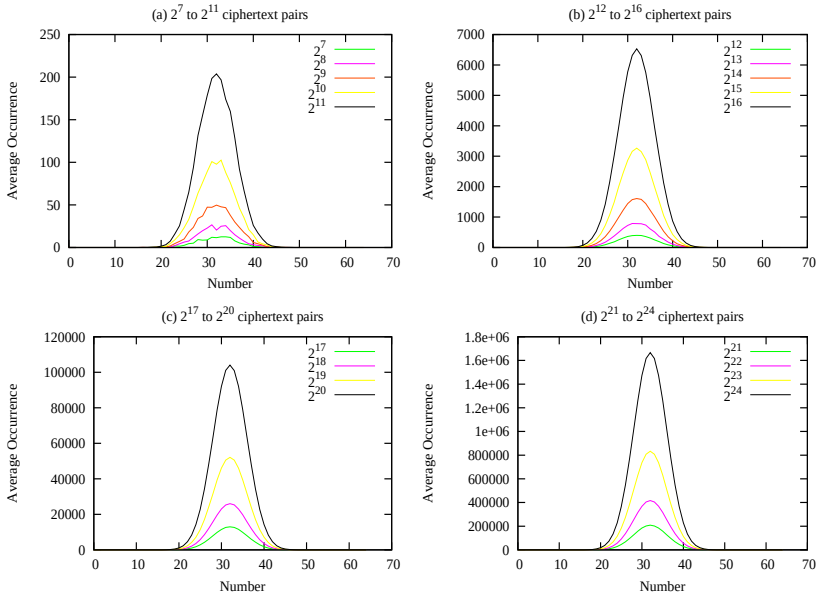

Figure B.61: Average Hamming Weight Distribution between ciphertext pairs for 8-round DES for 10 trials (Autofeeding)

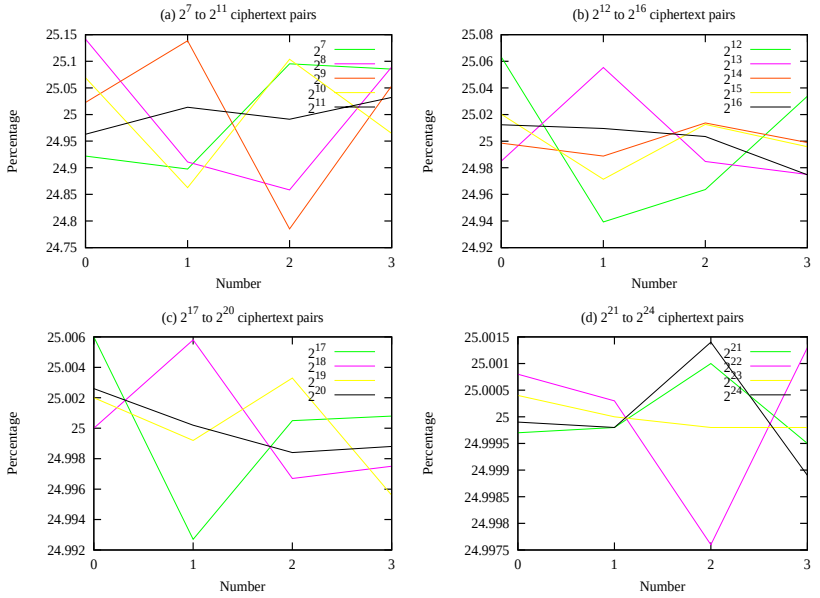

Figure B.62: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 8-round DES for 10 trials (Autofeeding)

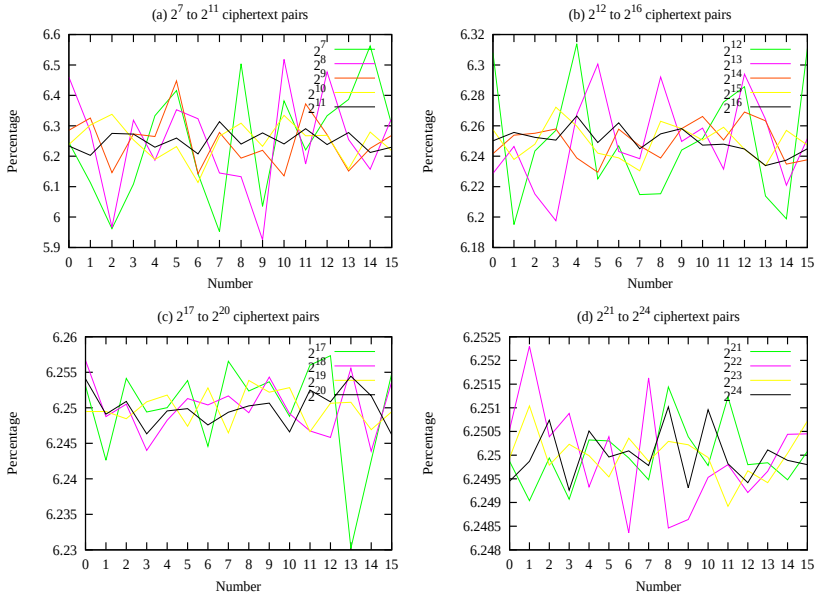

Figure B.63: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 8-round DES for 10 trials (Autofeeding)

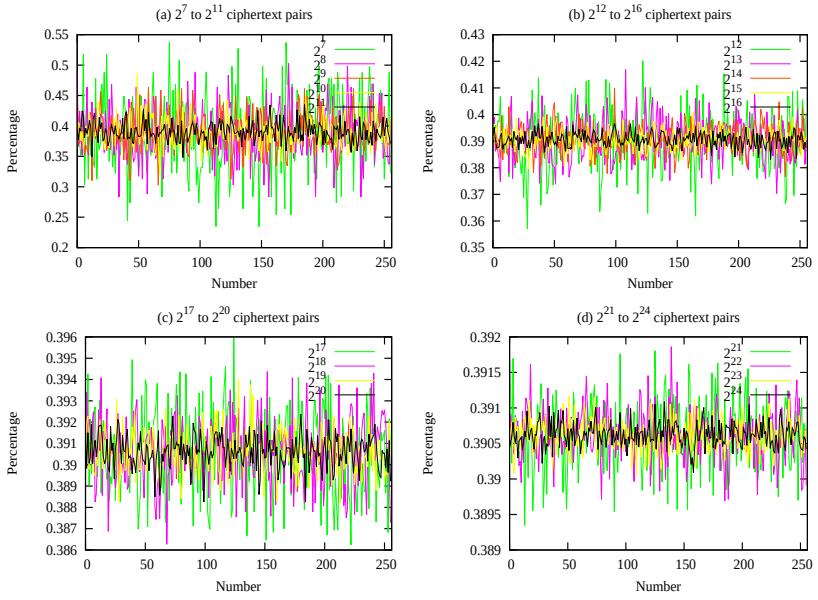

Figure B.64: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 8-round DES for 10 trials (Autofeeding)

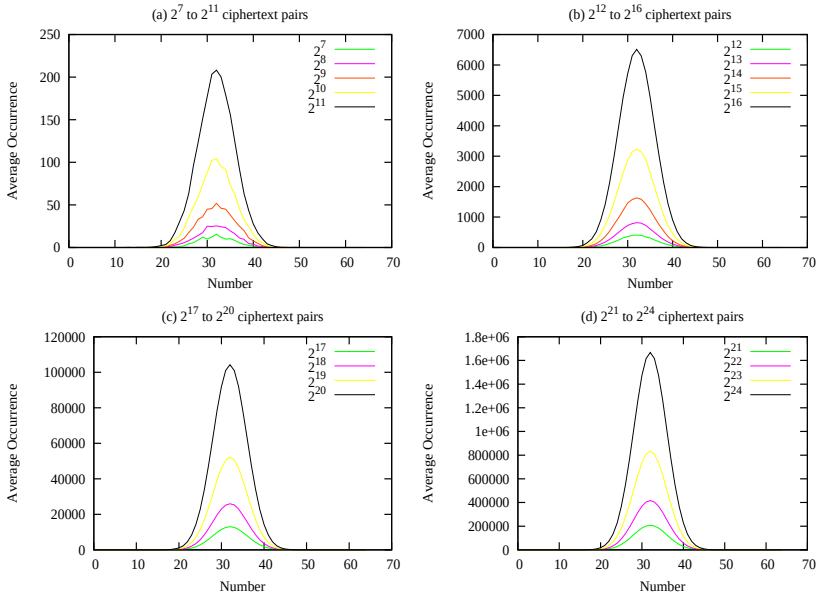

Figure B.65: Average Hamming Weight Distribution between ciphertext pairs for 9-round DES for 10 trials

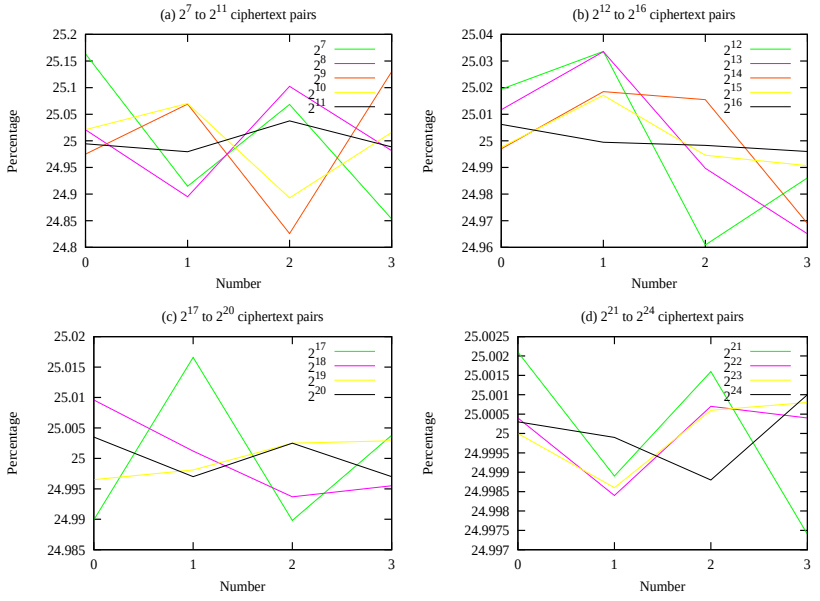

Figure B.66: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 9-round DES for 10 trials

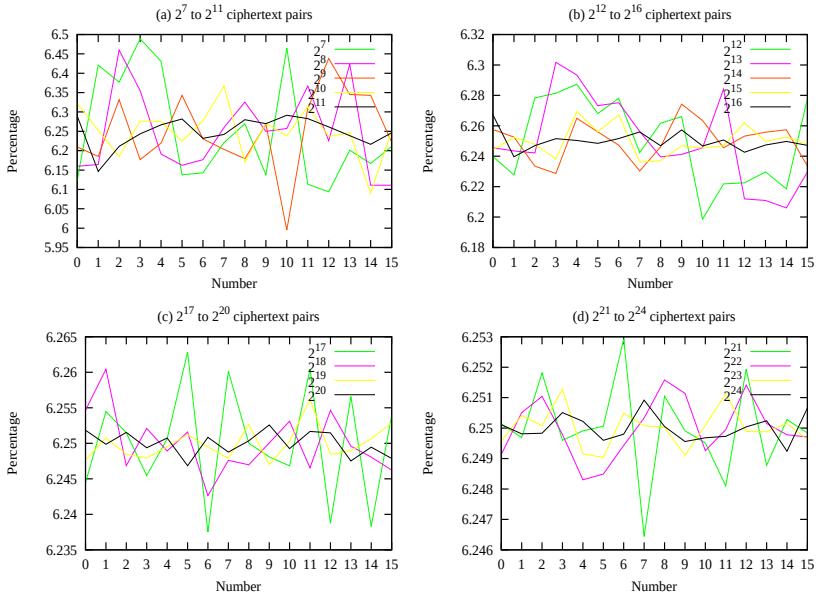

Figure B.67: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 9-round DES for 10 trials

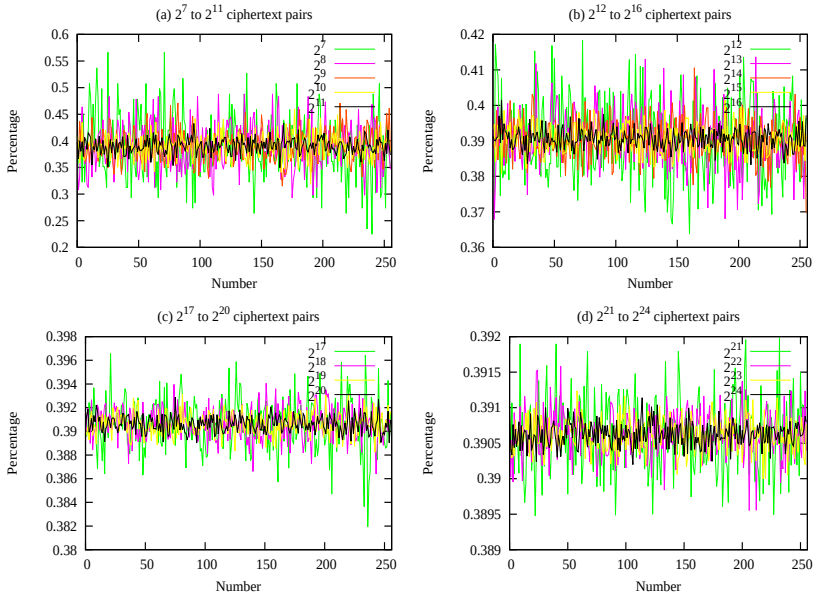

Figure B.68: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 9-round DES for 10 trials

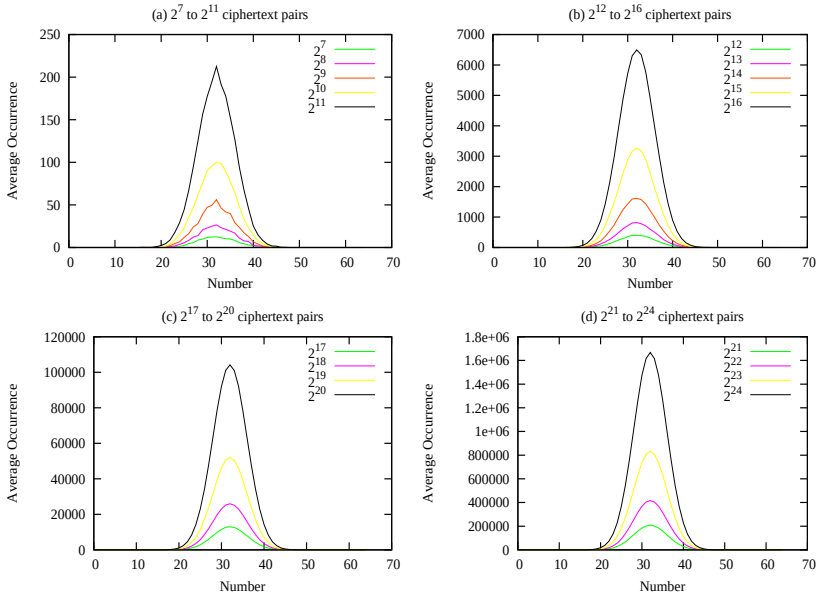

Figure B.69: Average Hamming Weight Distribution between ciphertext pairs for 9-round DES for 10 trials (Autofeeding)

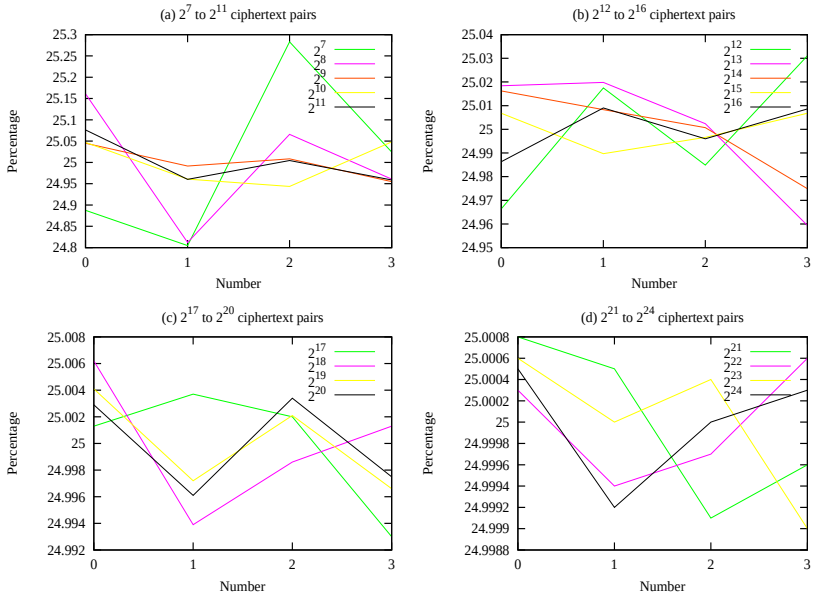

Figure B.70: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 9-round DES for 10 trials (Autofeeding)

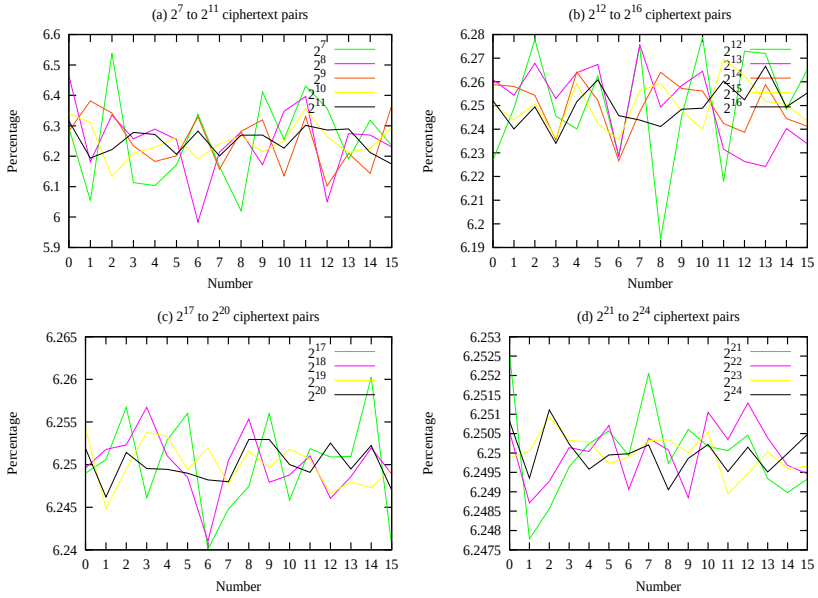

Figure B.71: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 9-round DES for 10 trials (Autofeeding)

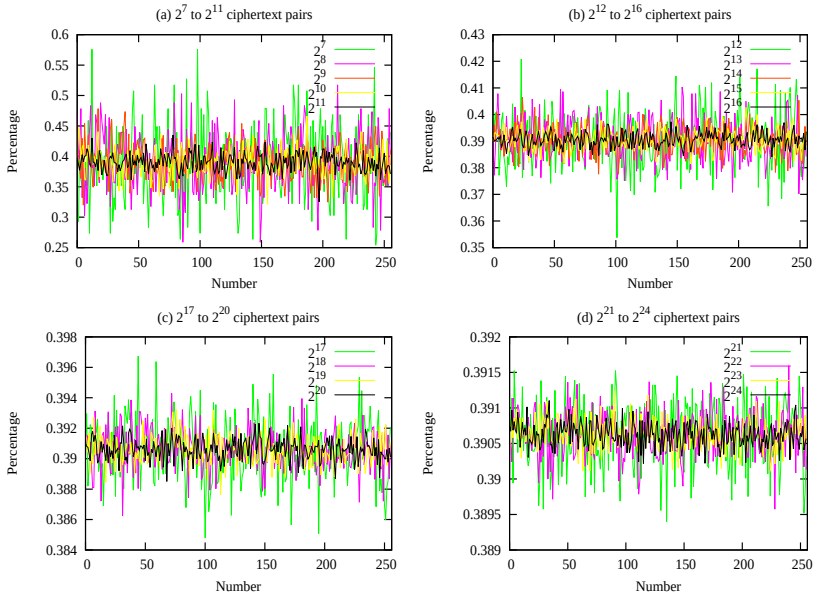

Figure B.72: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 9-round DES for 10 trials (Autofeeding)

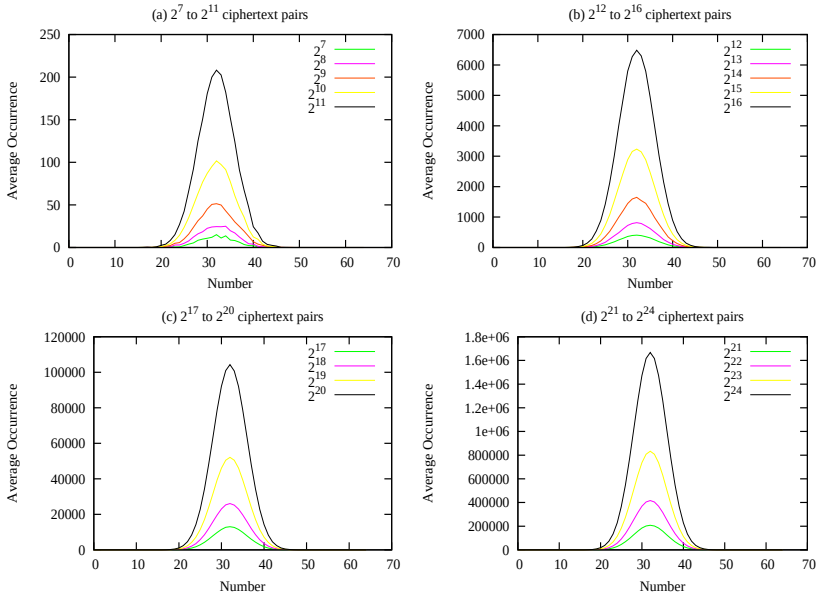

Figure B.73: Average Hamming Weight Distribution between ciphertext pairs for 10-round DES for 10 trials

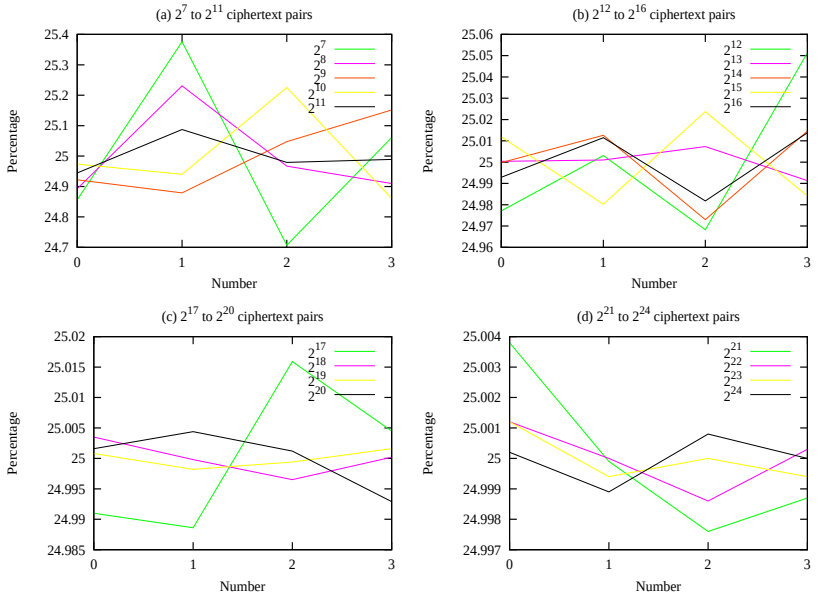

Figure B.74: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 10-round DES for 10 trials

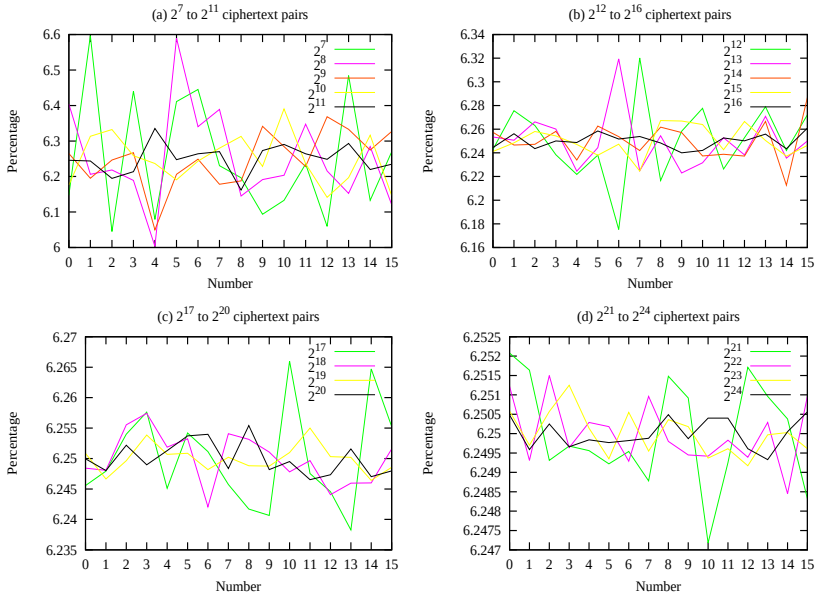

Figure B.75: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 10-round DES for 10 trials

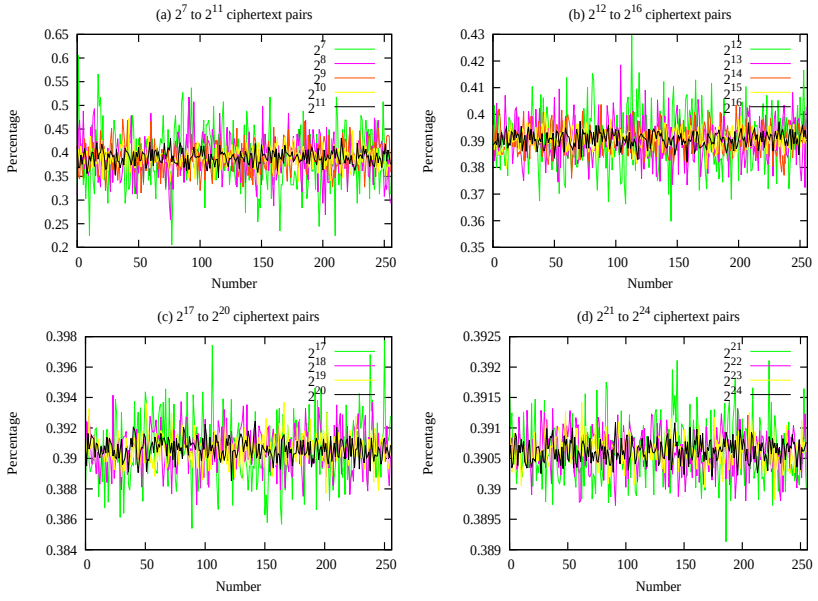

Figure B.76: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 10-round DES for 10 trials

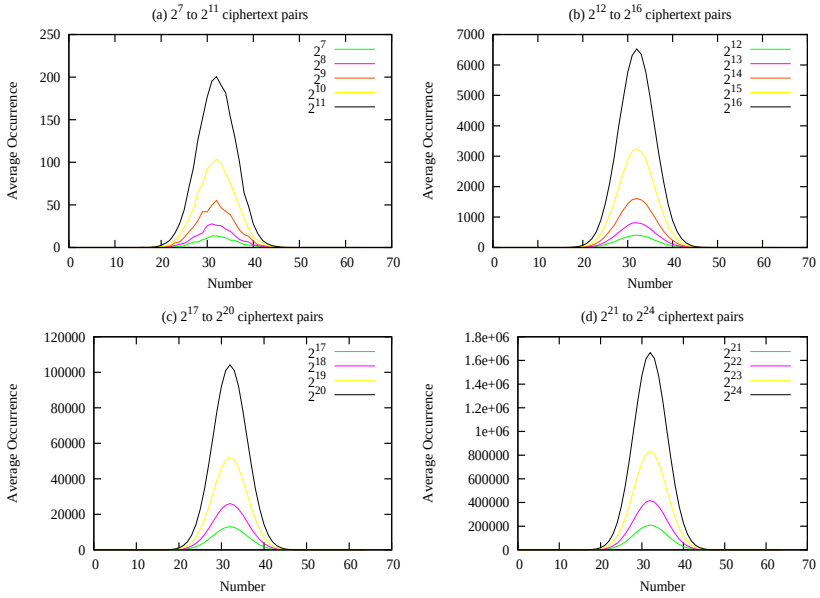

Figure B.77: Average Hamming Weight Distribution between ciphertext pairs for 10-round DES for 10 trials (Autofeeding)

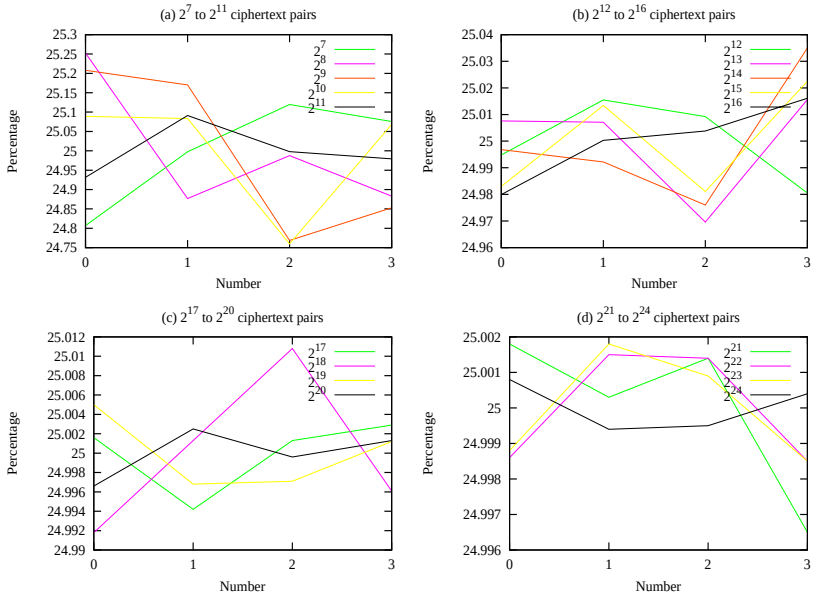

Figure B.78: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 10-round DES for 10 trials (Autofeeding)

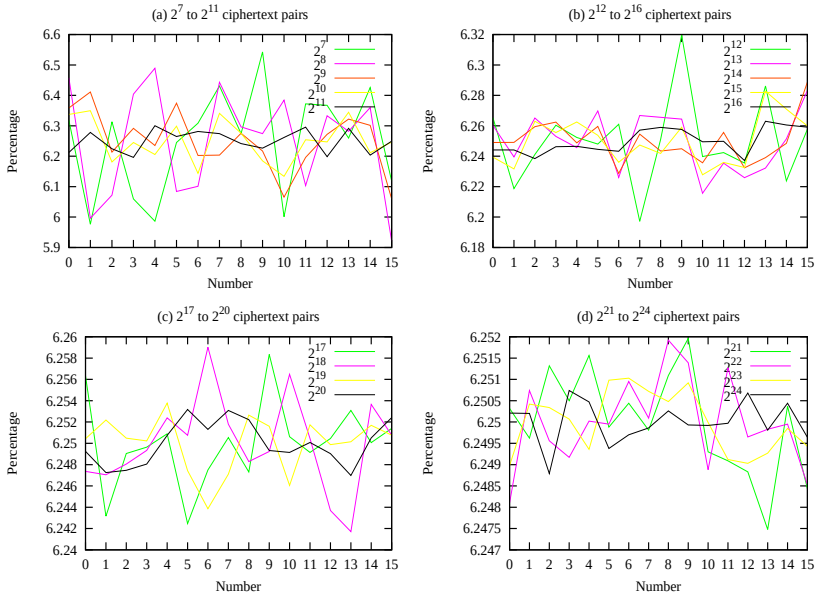

Figure B.79: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 10-round DES for 10 trials (Autofeeding)

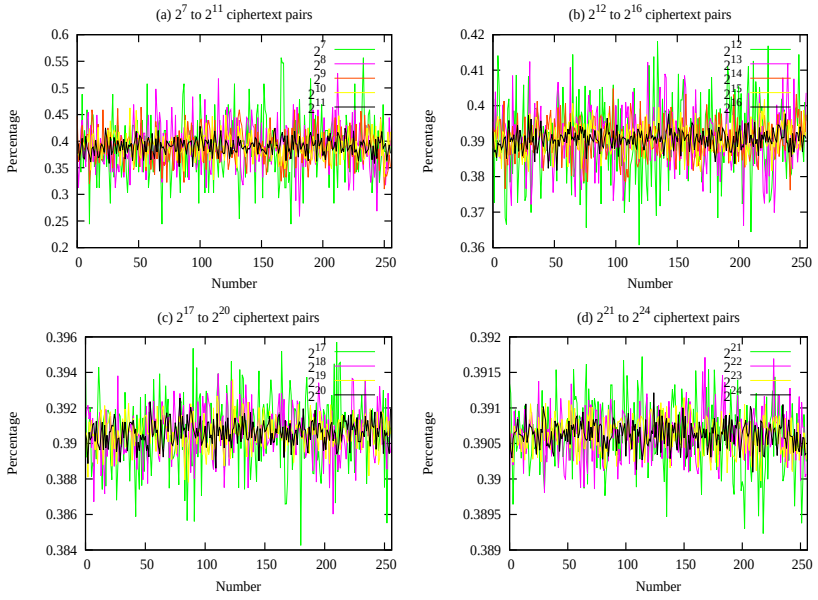

Figure B.80: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 10-round DES for 10 trials (Autofeeding)

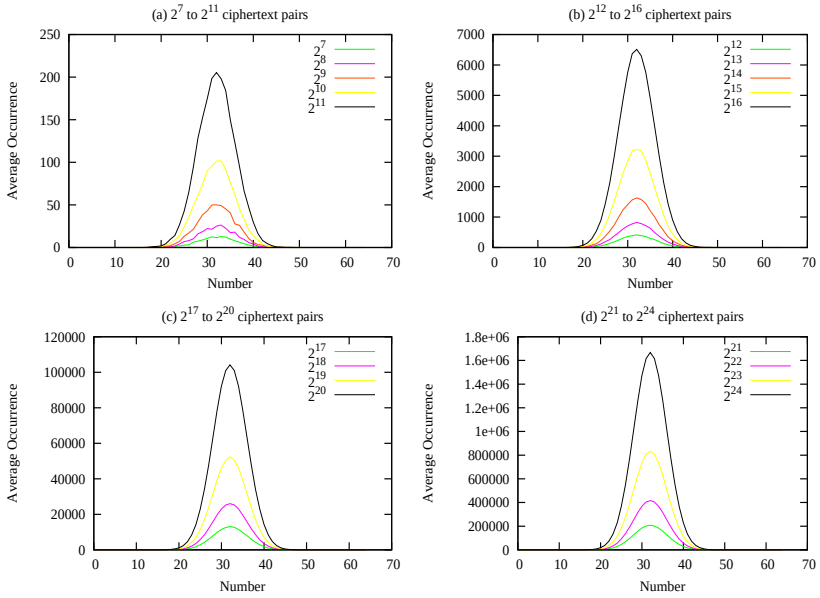

Figure B.81: Average Hamming Weight Distribution between ciphertext pairs for 11-round DES for 10 trials

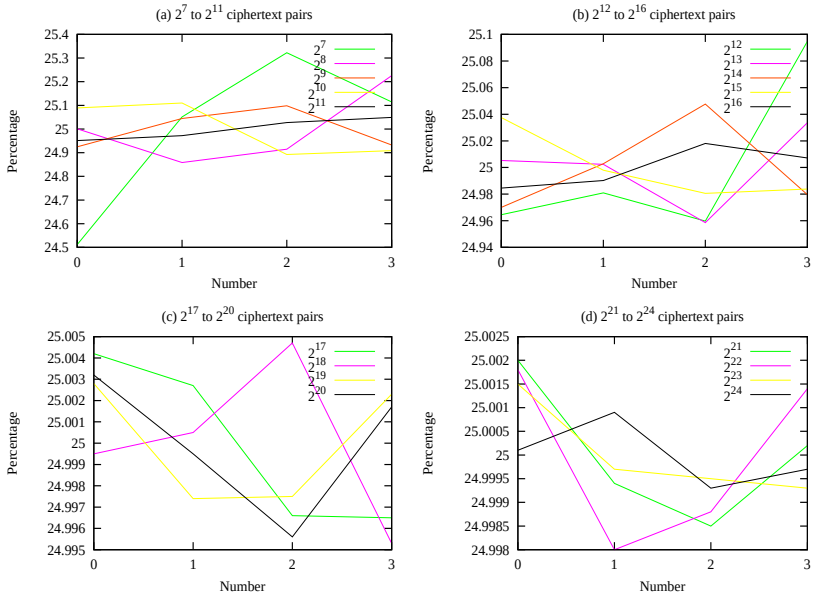

Figure B.82: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 11-round DES for 10 trials

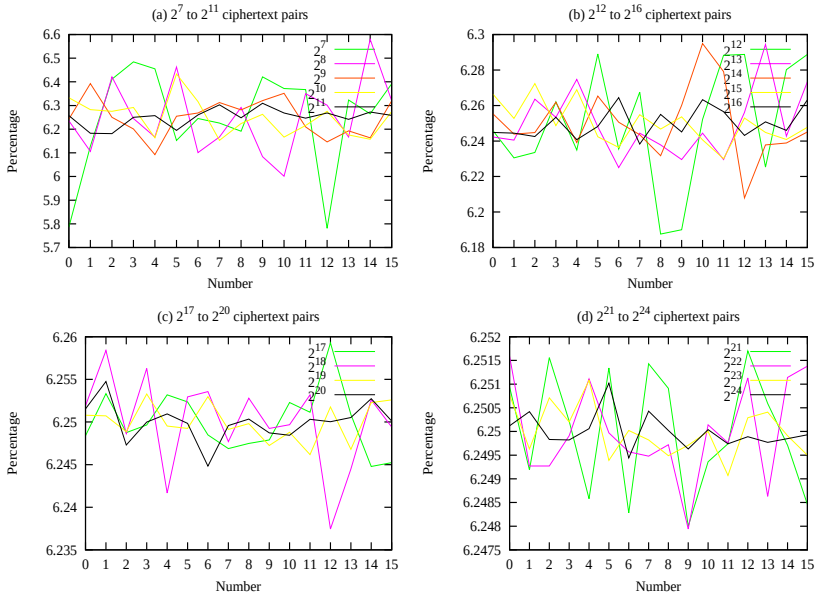

Figure B.83: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 11-round DES for 10 trials

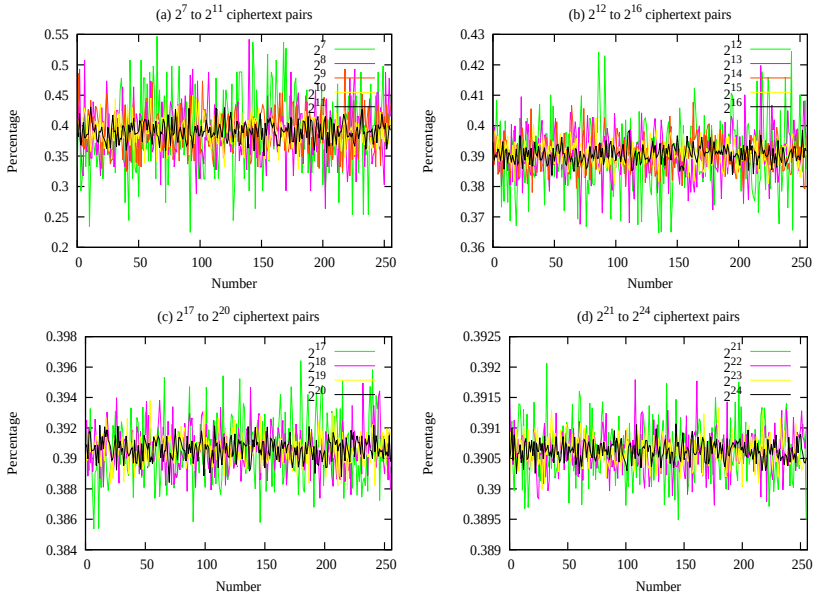

Figure B.84: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 11-round DES for 10 trials

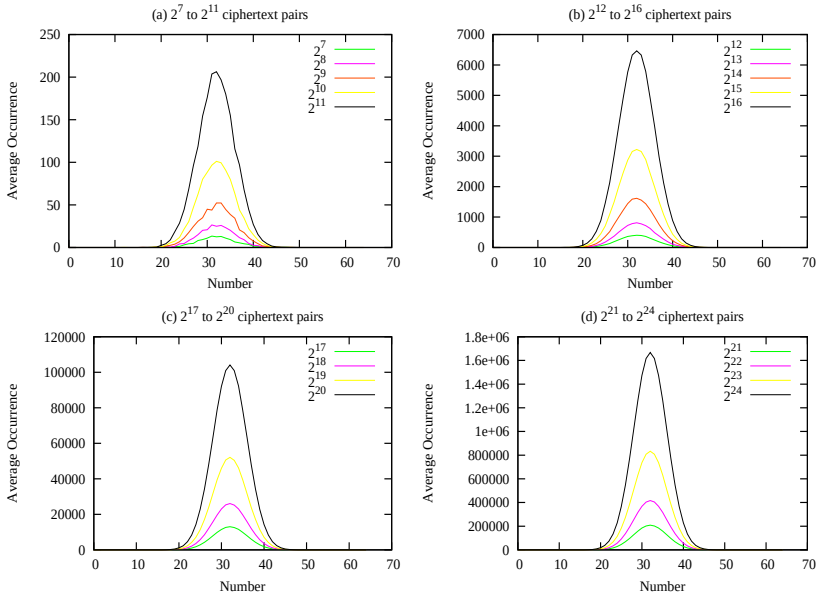

Figure B.85: Average Hamming Weight Distribution between ciphertext pairs for 11-round DES for 10 trials (Autofeeding)

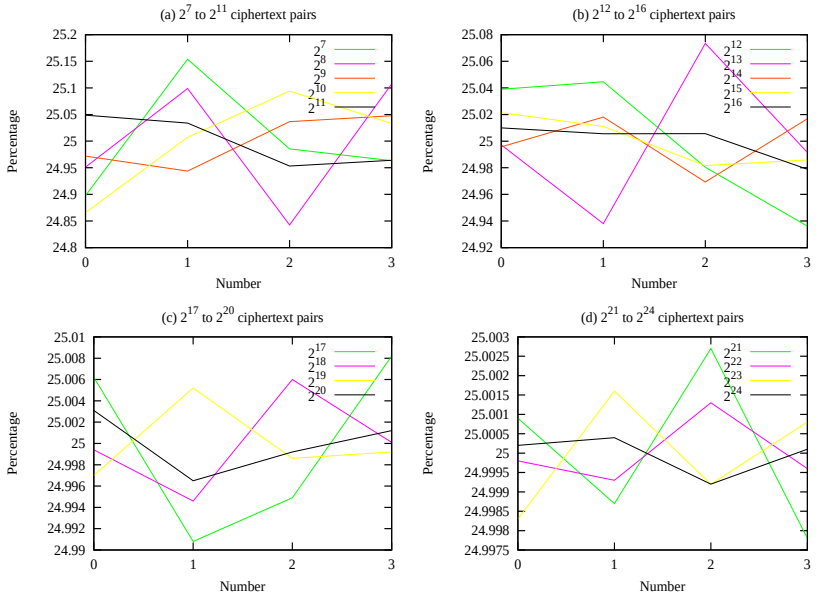

Figure B.86: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 11-round DES for 10 trials (Autofeeding)

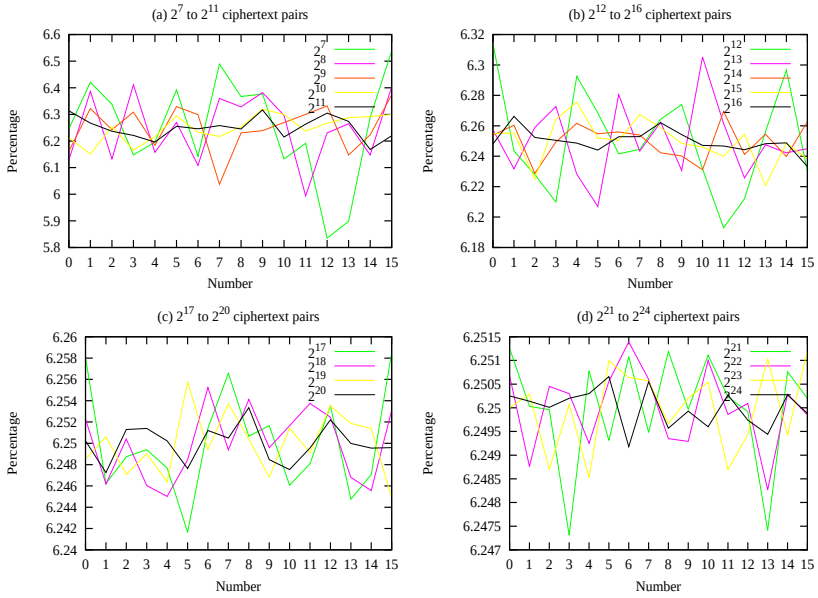

Figure B.87: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 11-round DES for 10 trials (Autofeeding)

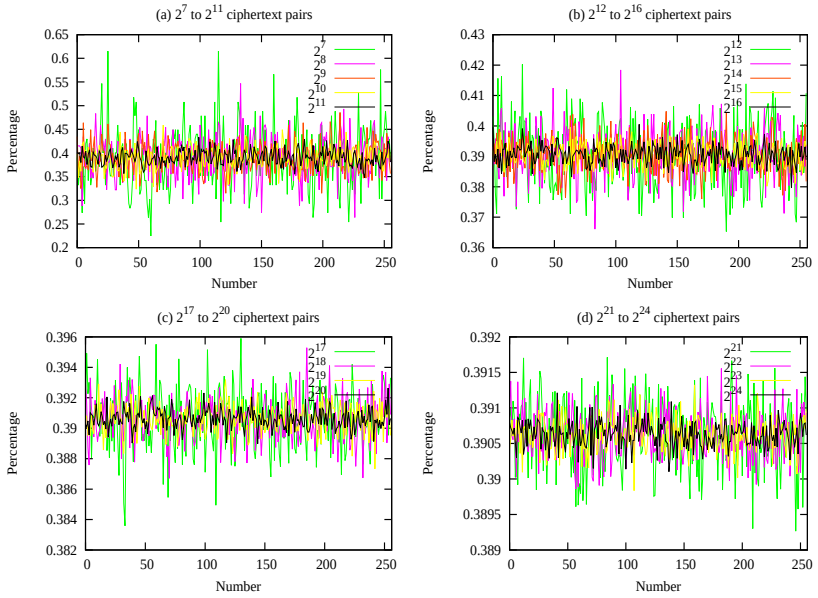

Figure B.88: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 11-round DES for 10 trials (Autofeeding)

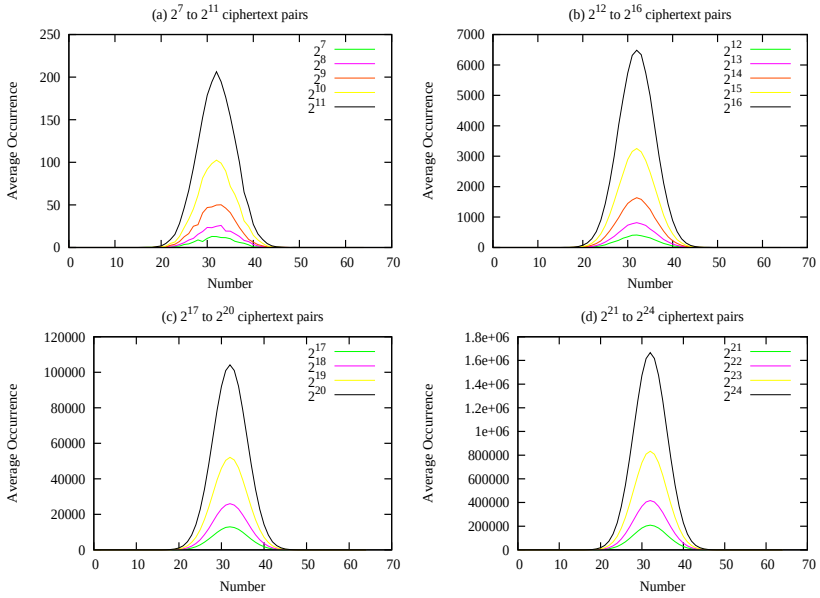

Figure B.89: Average Hamming Weight Distribution between ciphertext pairs for 12-round DES for 10 trials

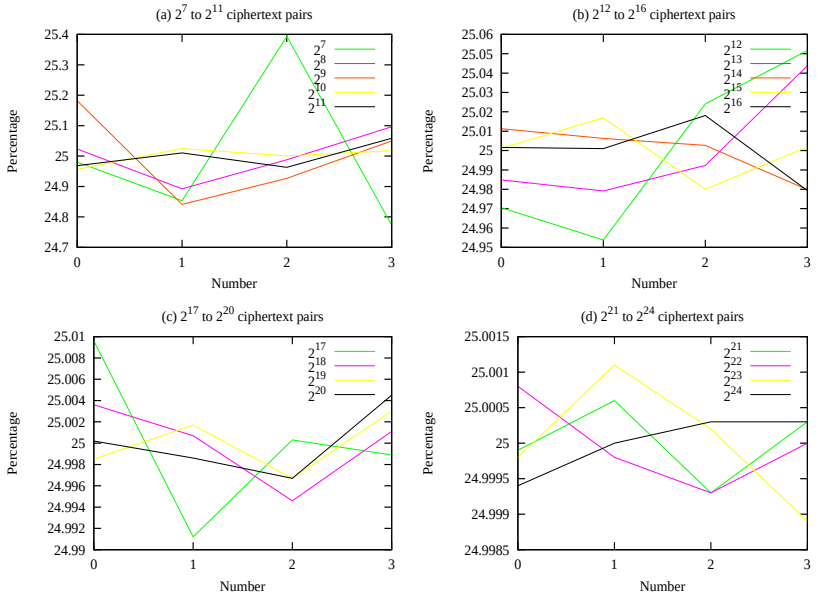

Figure B.90: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 12-round DES for 10 trials

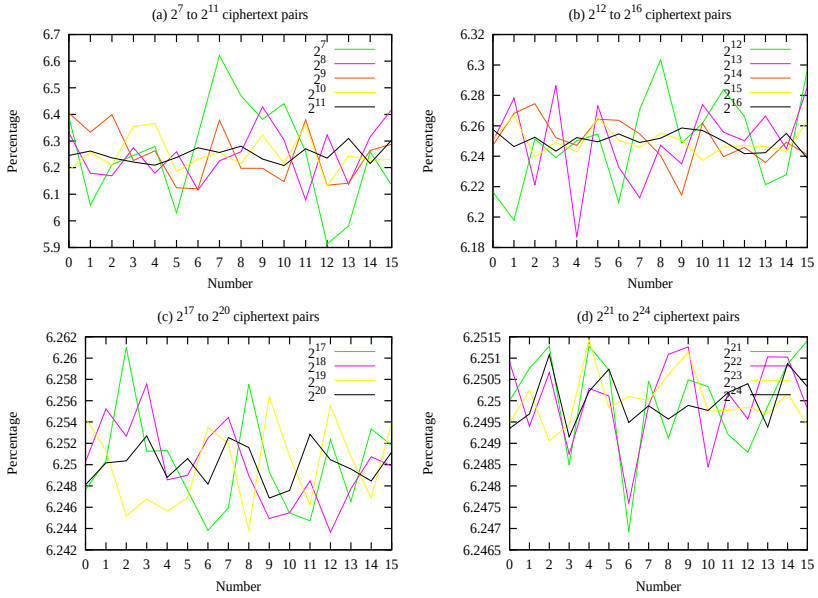

Figure B.91: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 12-round DES for 10 trials

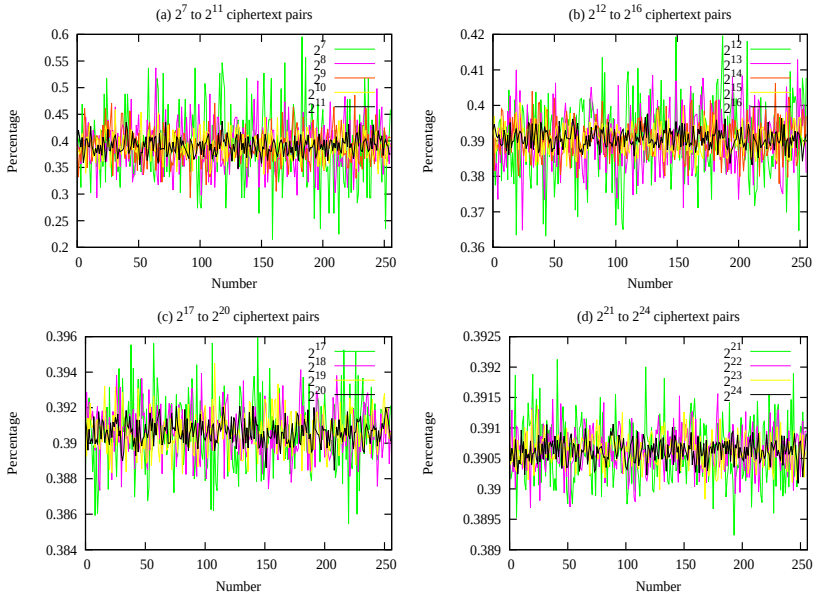

Figure B.92: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 12-round DES for 10 trials

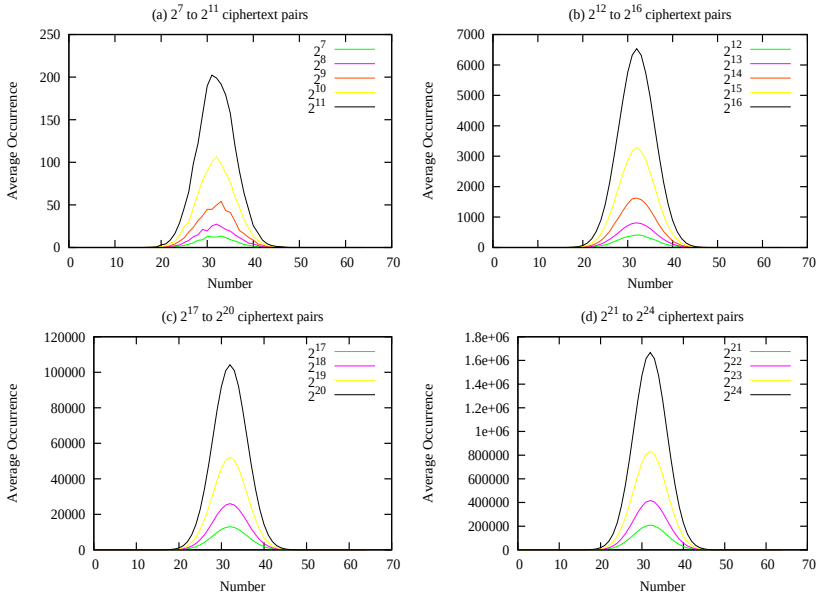

Figure B.93: Average Hamming Weight Distribution between ciphertext pairs for 12-round DES for 10 trials (Autofeeding)

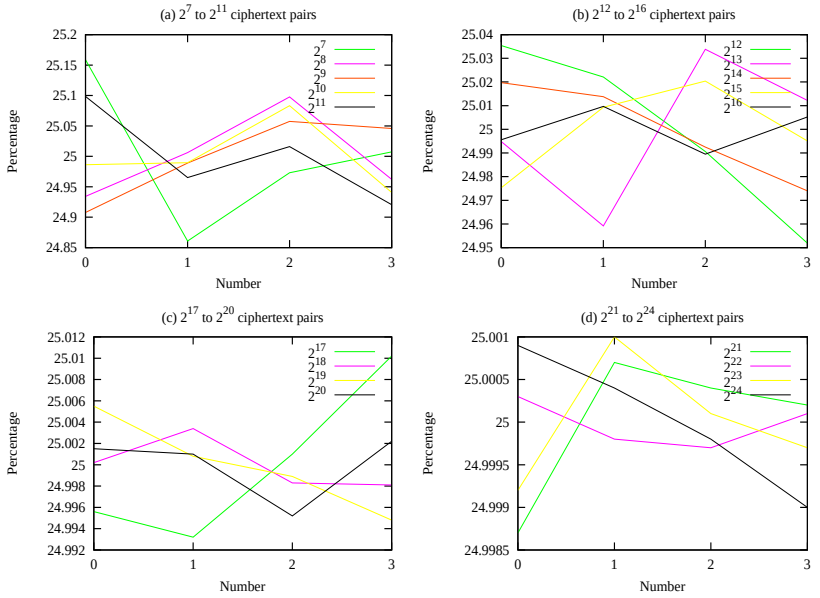

Figure B.94: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 12-round DES for 10 trials (Autofeeding)

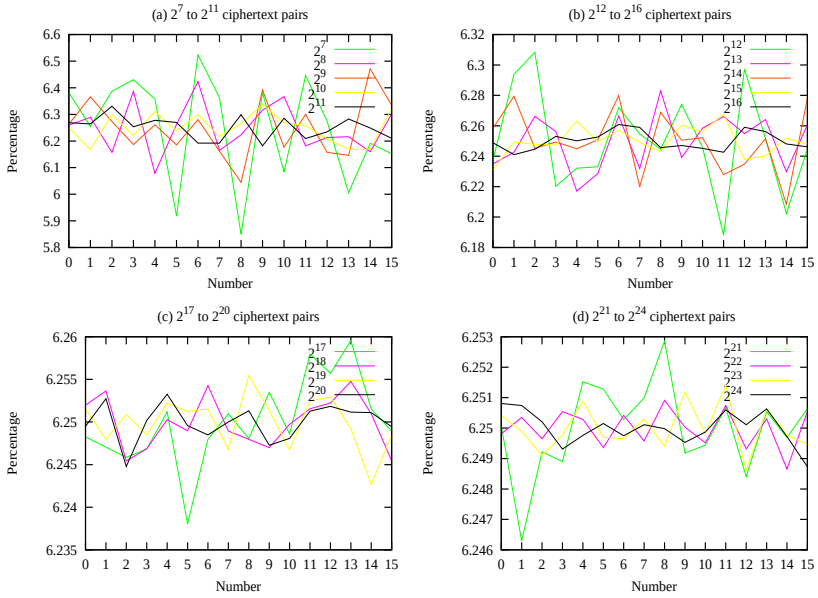

Figure B.95: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 12-round DES for 10 trials (Autofeeding)

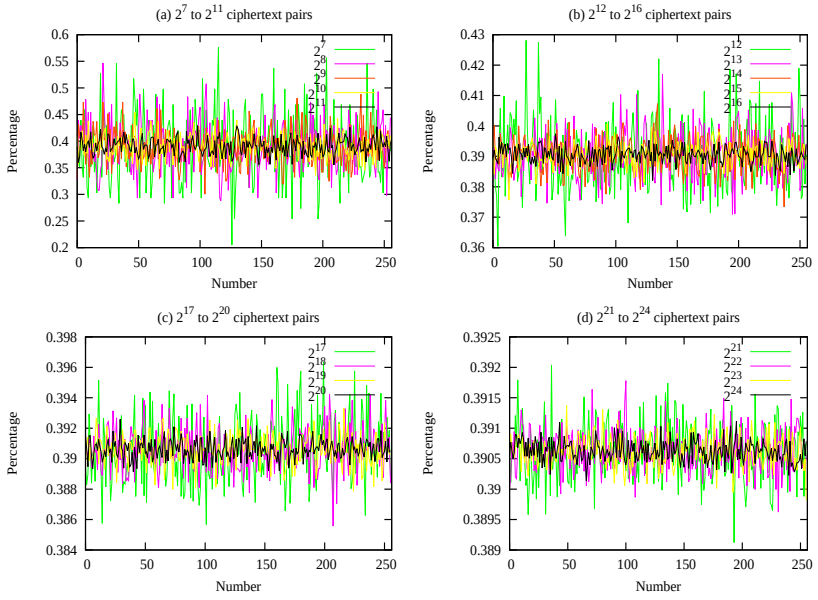

Figure B.96: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 12-round DES for 10 trials (Autofeeding)

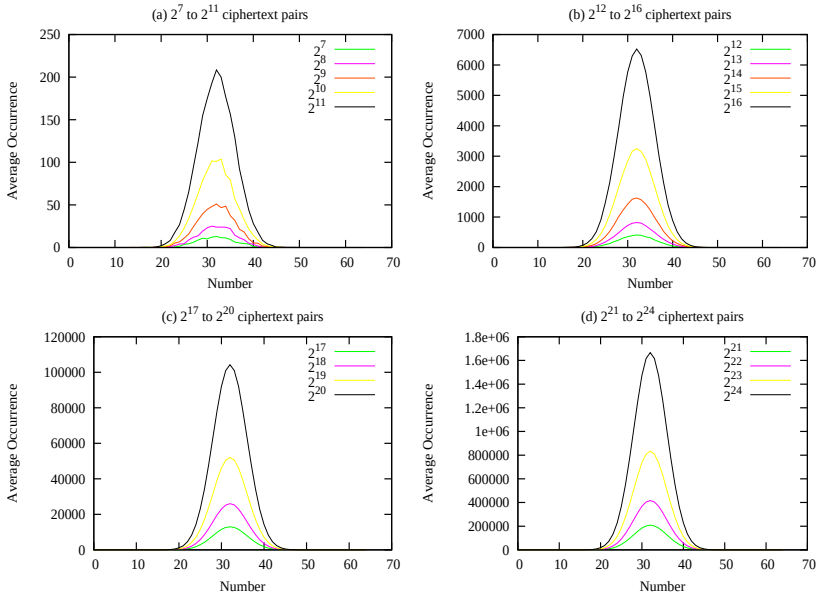

Figure B.97: Average Hamming Weight Distribution between ciphertext pairs for 13-round DES for 10 trials

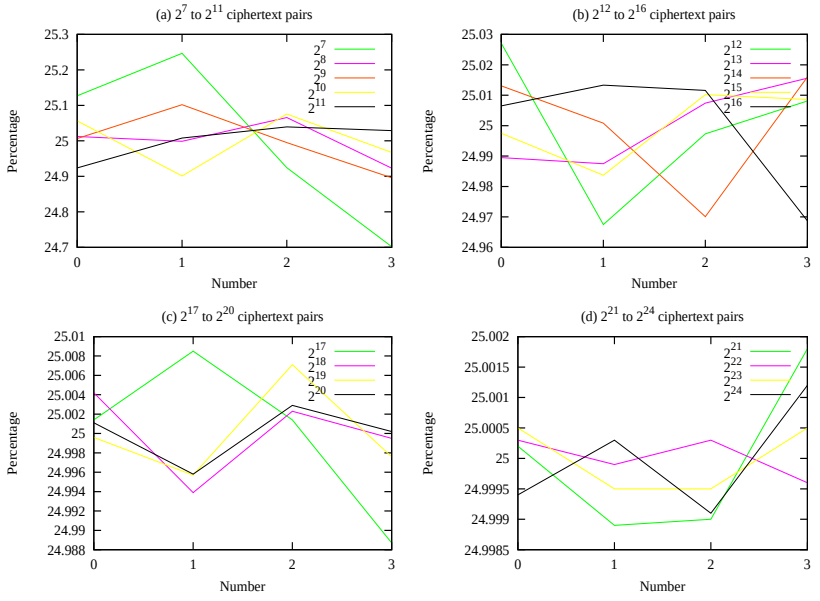

Figure B.98: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 13-round DES for 10 trials

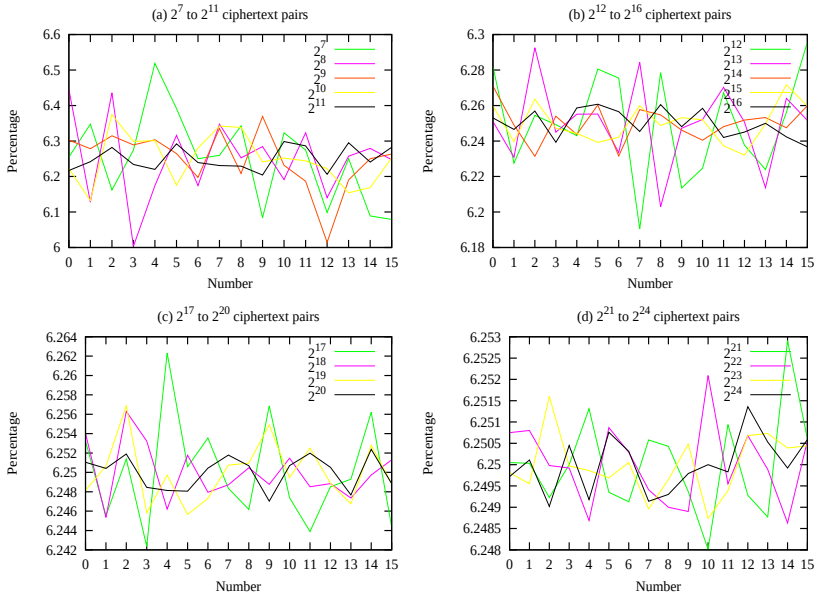

Figure B.99: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 13-round DES for 10 trials

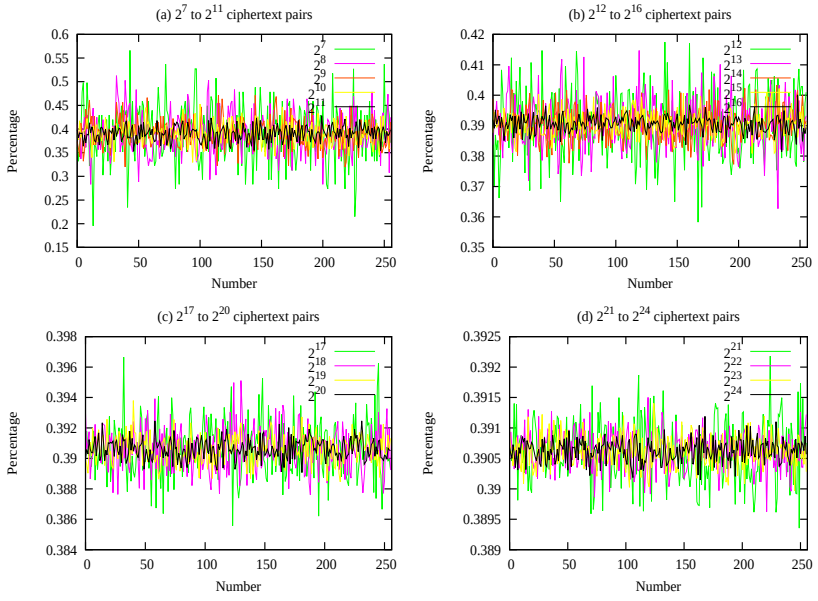

Figure B.100: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 13-round DES for 10 trials

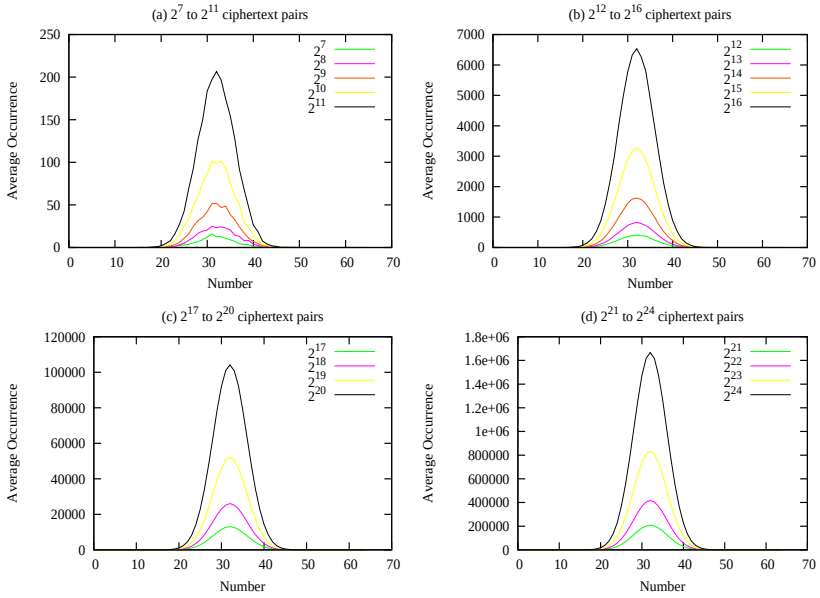

Figure B.101: Average Hamming Weight Distribution between ciphertext pairs for 13-round DES for 10 trials (Autofeeding)

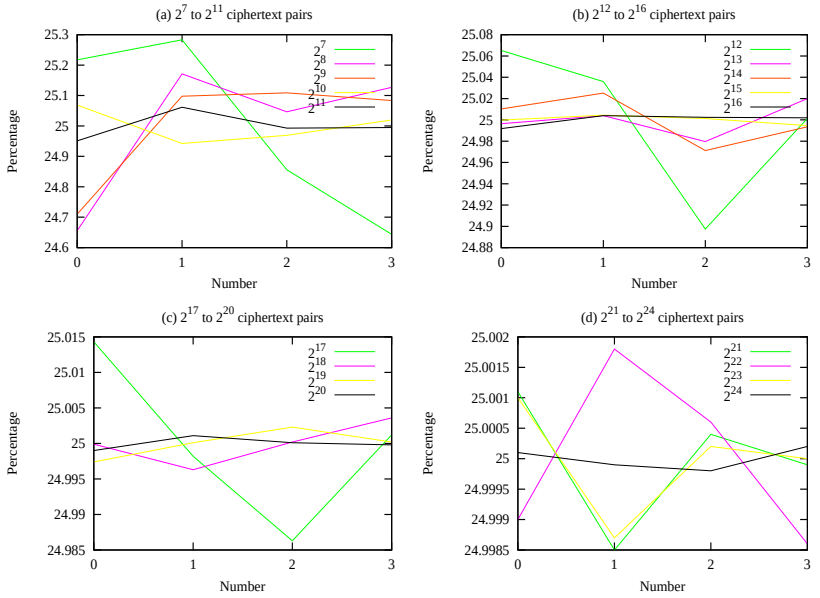

Figure B.102: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 13-round DES for 10 trials (Autofeeding)

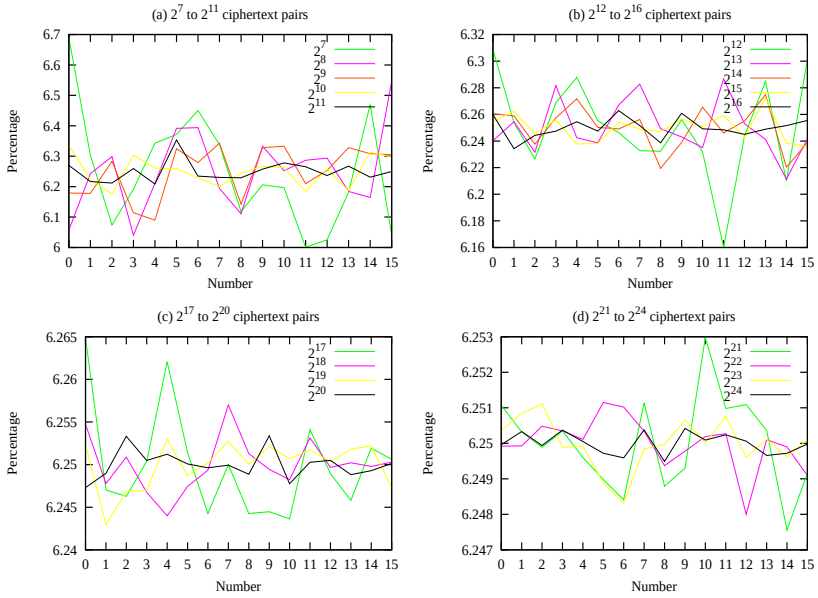

Figure B.103: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 13-round DES for 10 trials (Autofeeding)

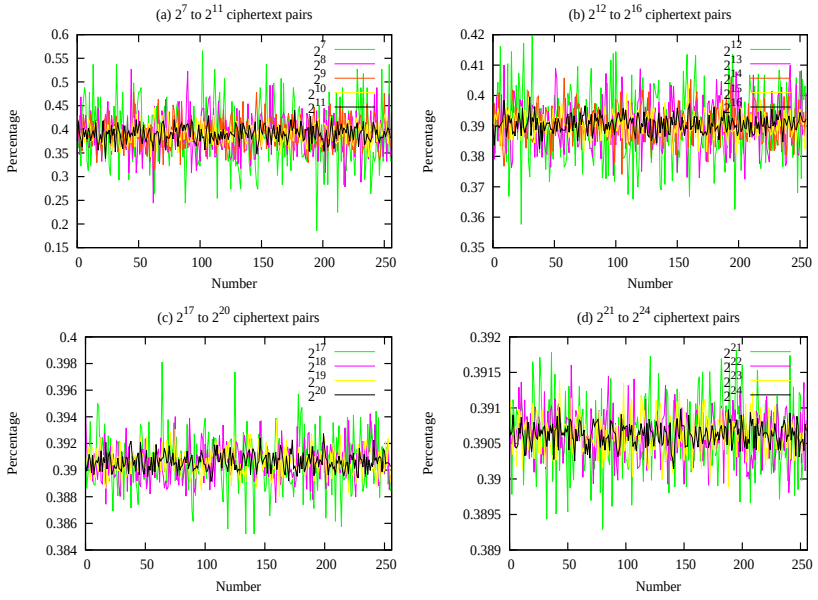

Figure B.104: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 13-round DES for 10 trials (Autofeeding)

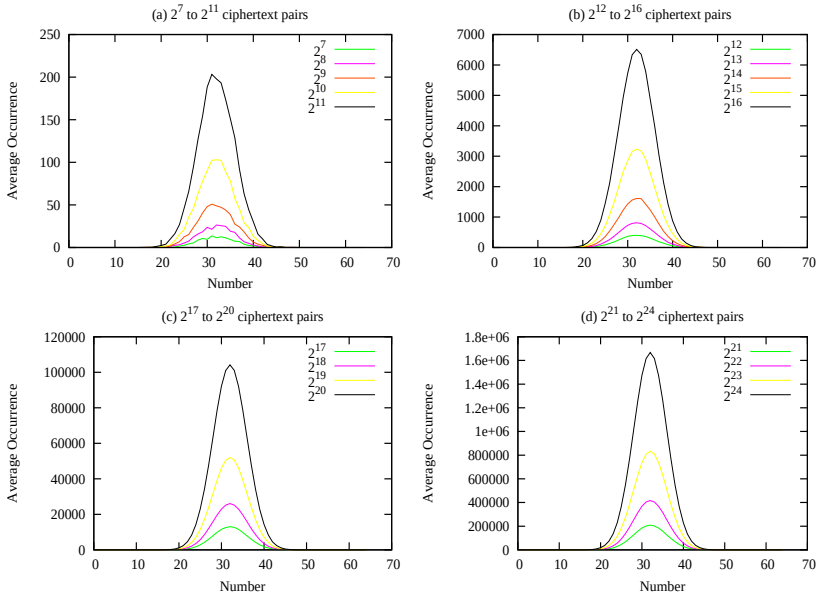

Figure B.105: Average Hamming Weight Distribution between ciphertext pairs for 14-round DES for 10 trials

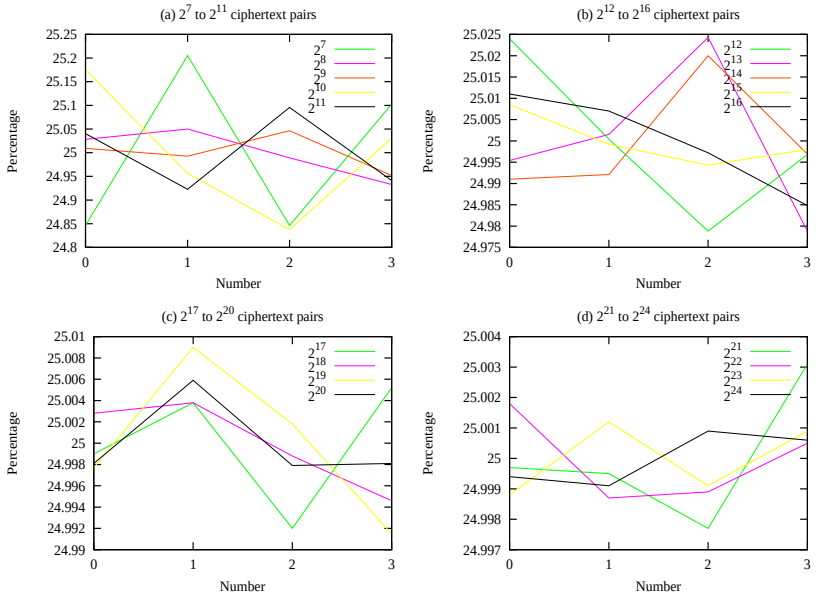

Figure B.106: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 14-round DES for 10 trials

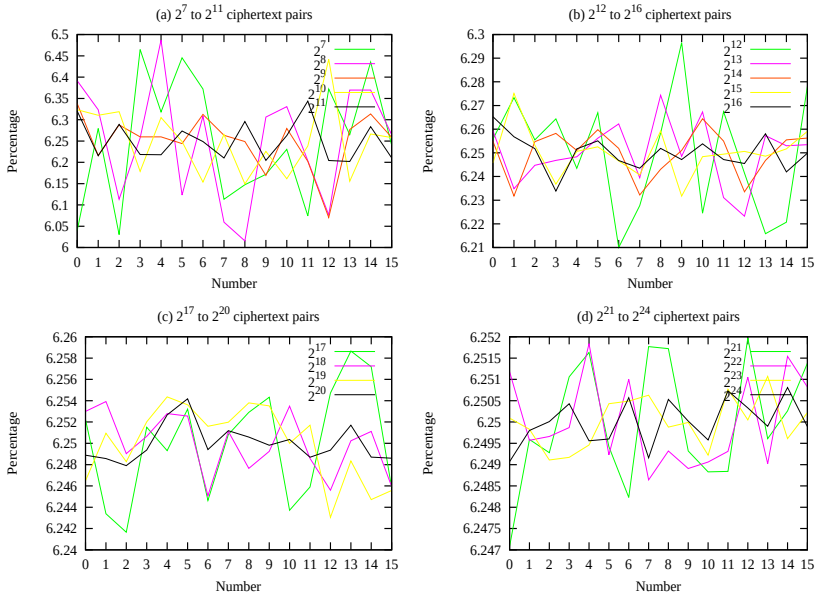

Figure B.107: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 14-round DES for 10 trials

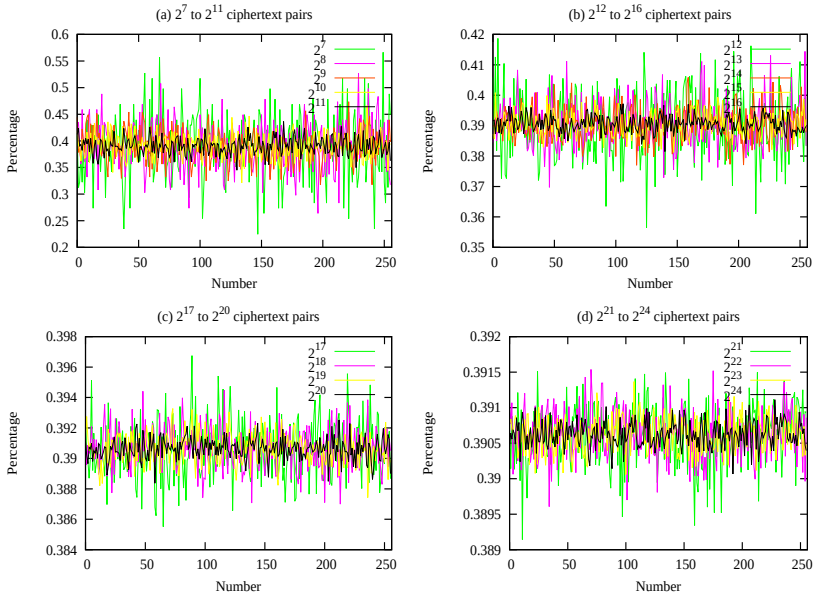

Figure B.108: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 14-round DES for 10 trials

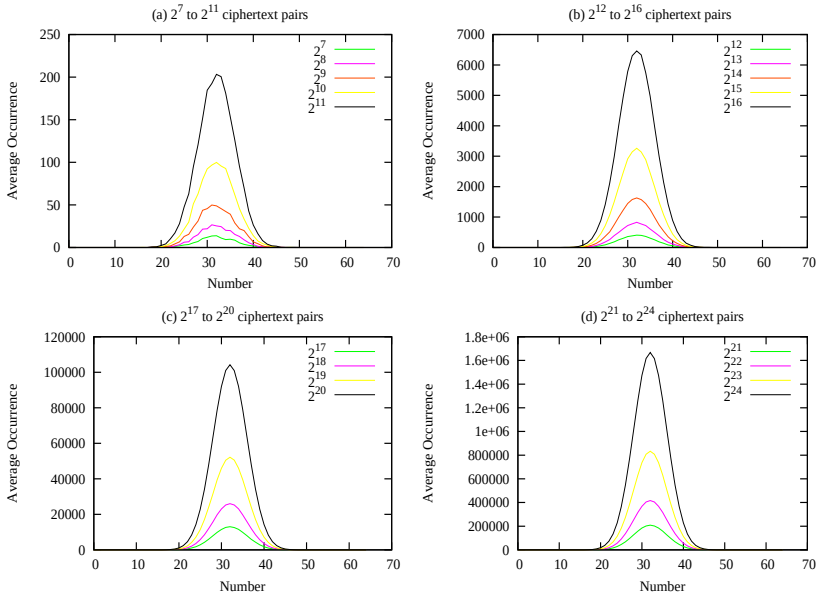

Figure B.109: Average Hamming Weight Distribution between ciphertext pairs for 14-round DES for 10 trials (Autofeeding)

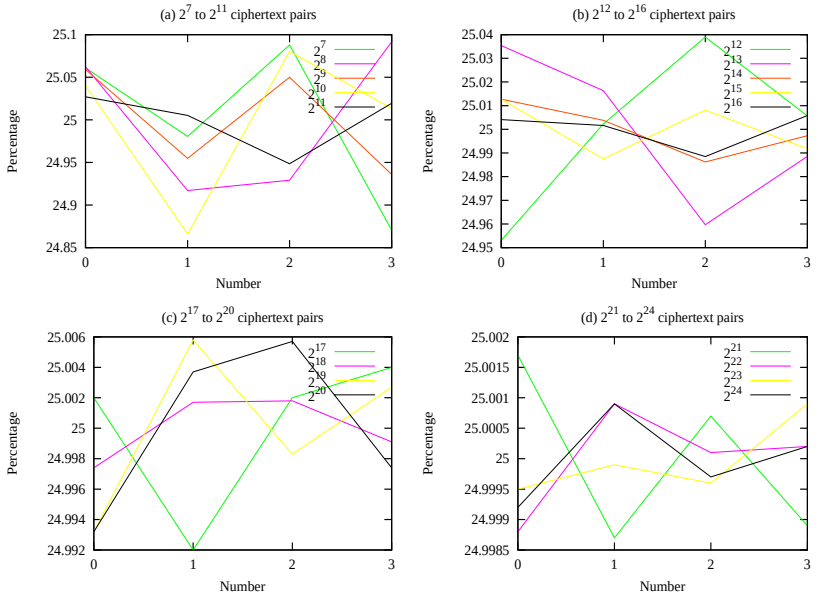

Figure B.110: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 14-round DES for 10 trials (Autofeeding)

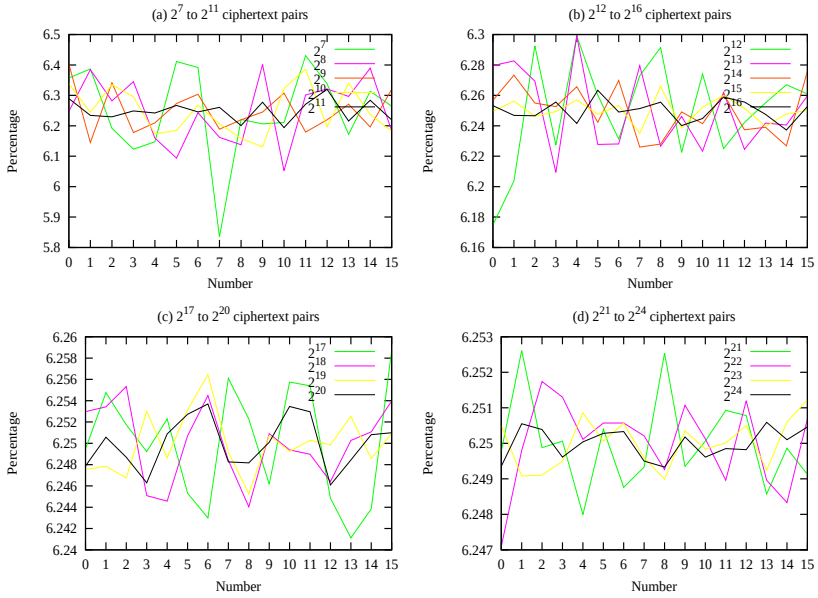

Figure B.111: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 14-round DES for 10 trials (Autofeeding)

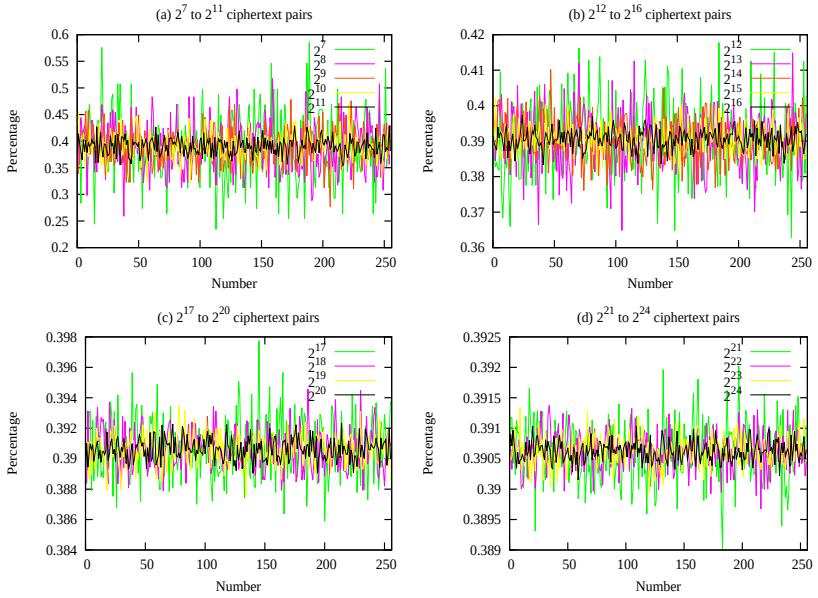

Figure B.112: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 14-round DES for 10 trials (Autofeeding)

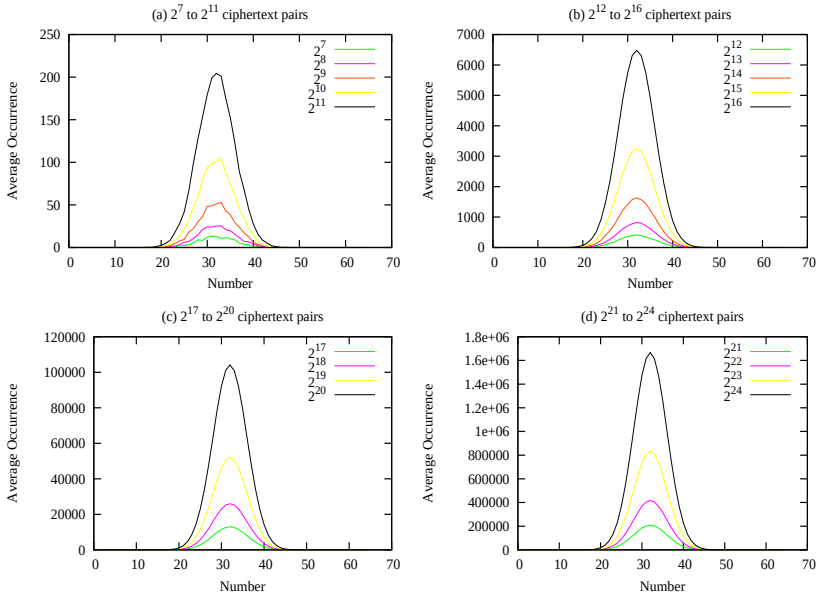

Figure B.113: Average Hamming Weight Distribution between ciphertext pairs for 15-round DES for 10 trials

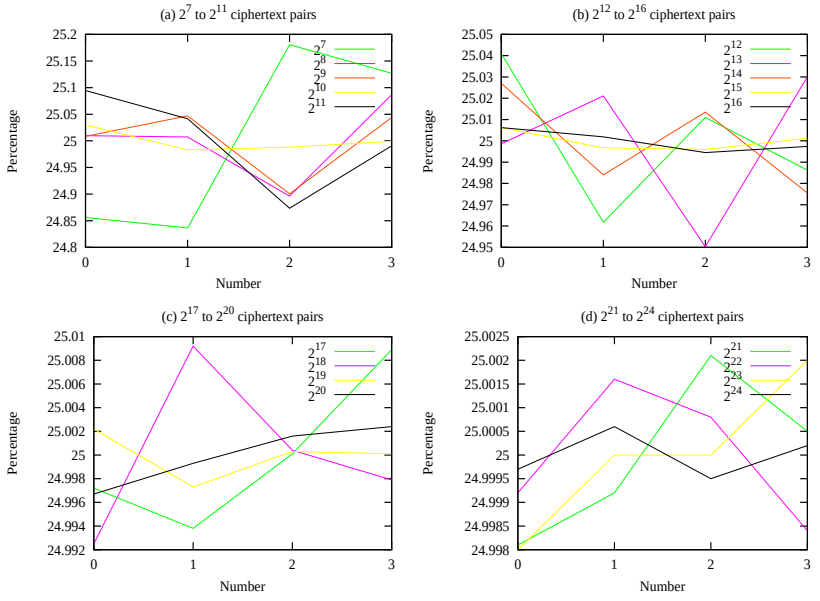

Figure B.114: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 15-round DES for 10 trials

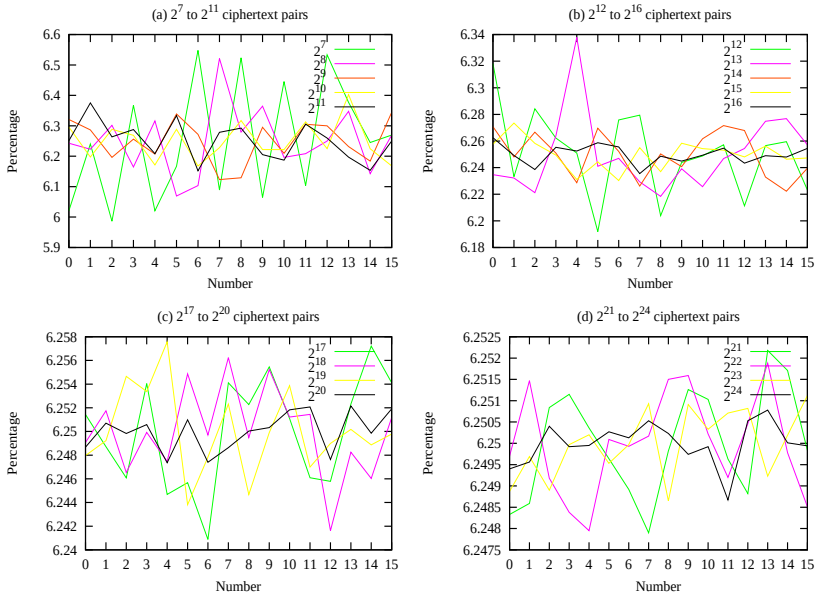

Figure B.115: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 15-round DES for 10 trials

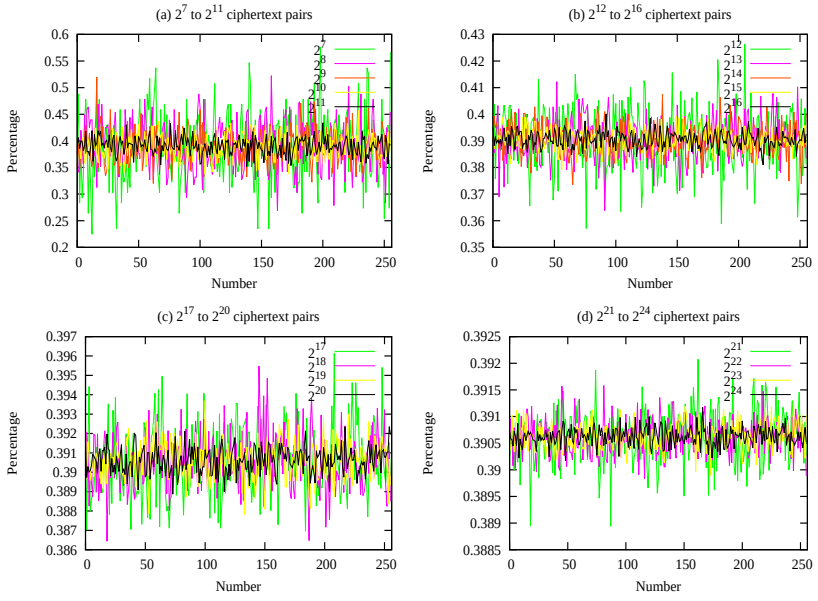

Figure B.116: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 15-round DES for 10 trials

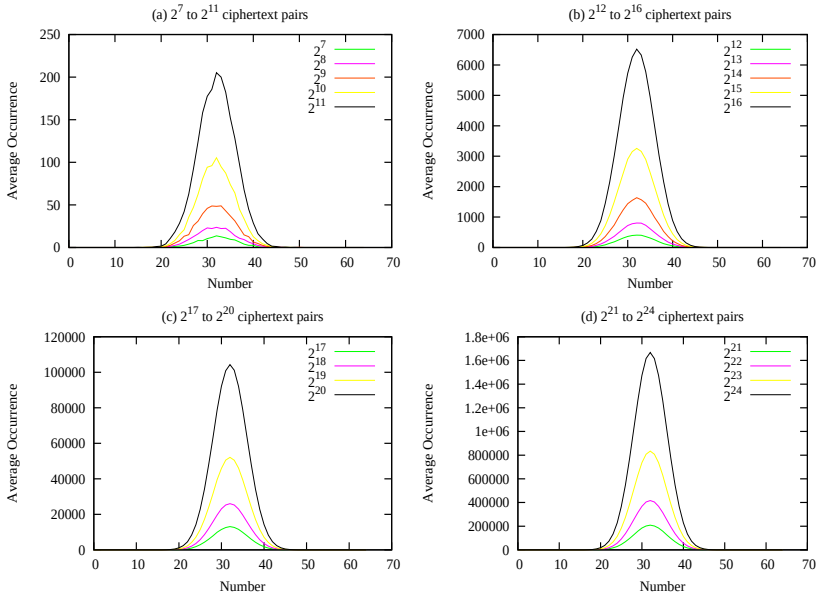

Figure B.117: Average Hamming Weight Distribution between ciphertext pairs for 15-round DES for 10 trials (Autofeeding)

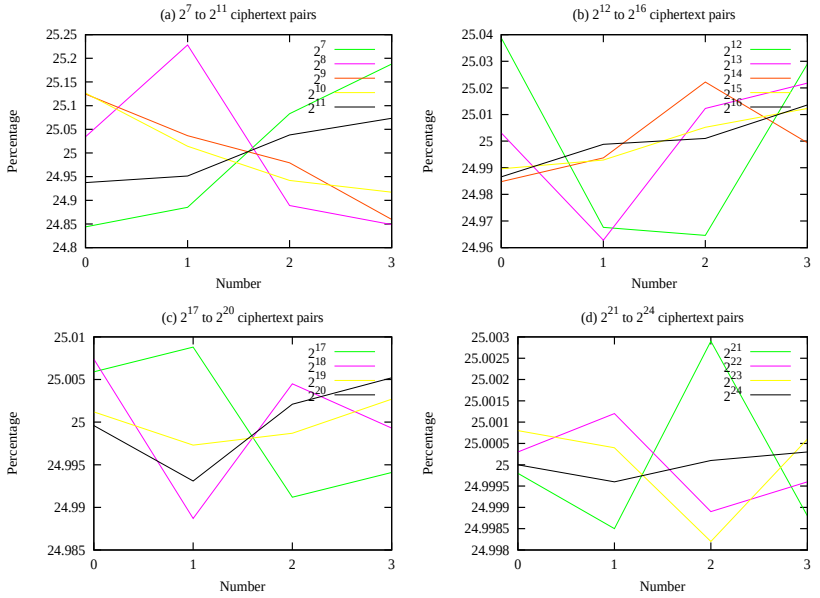

Figure B.118: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 15-round DES for 10 trials (Autofeeding)

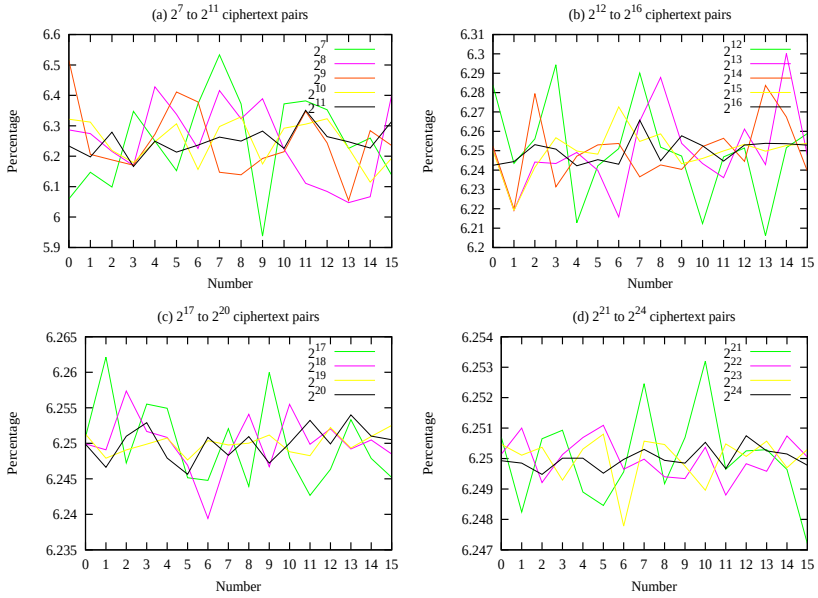

Figure B.119: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 15-round DES for 10 trials (Autofeeding)

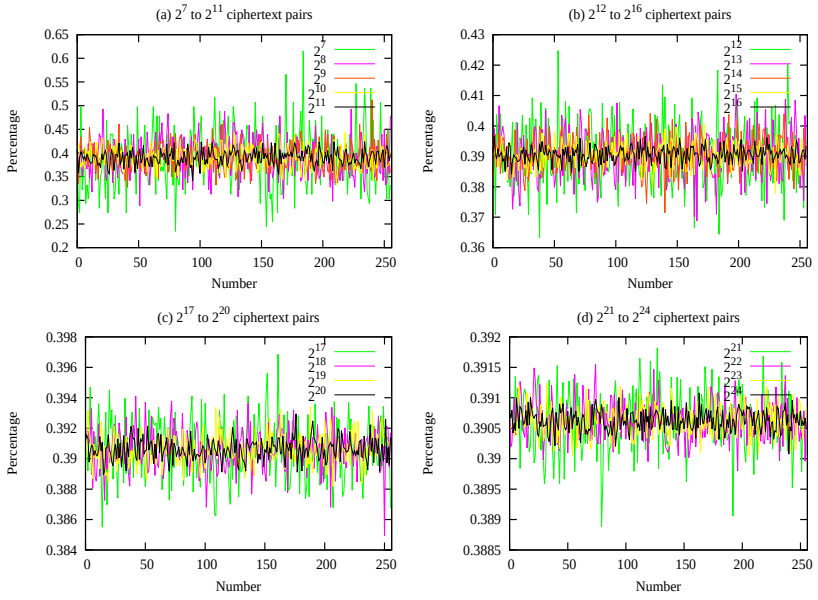

Figure B.120: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 15-round DES for 10 trials (Autofeeding)

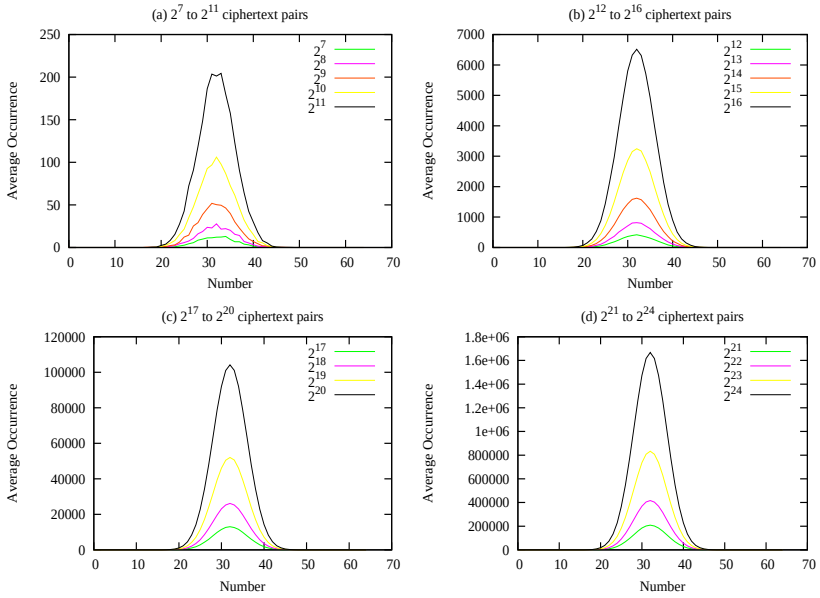

Figure B.121: Average Hamming Weight Distribution between ciphertext pairs for 16-round DES for 10 trials

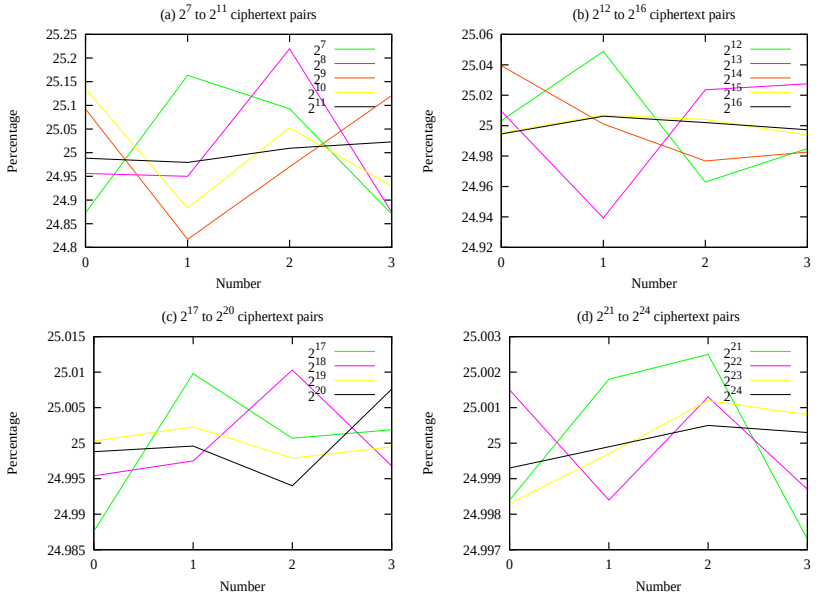

Figure B.122: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 16-round DES for 10 trials

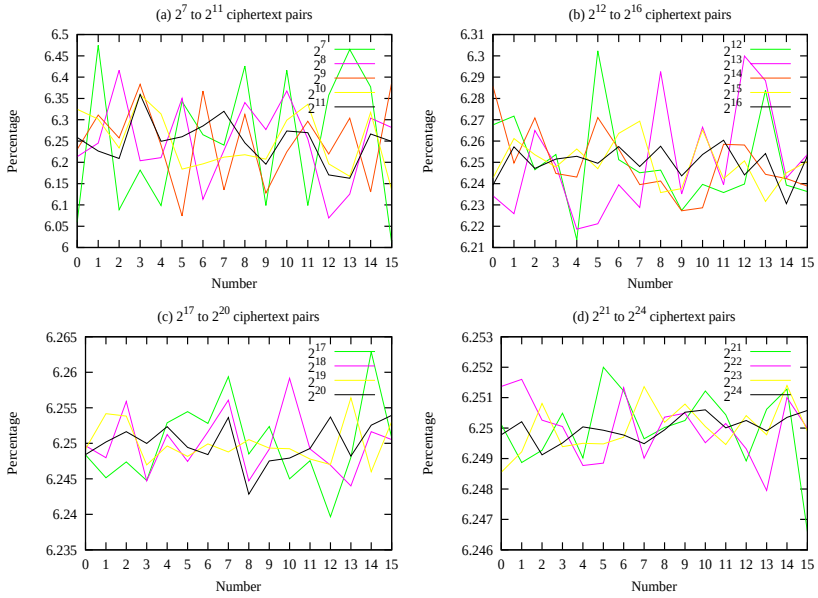

Figure B.123: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 16-round DES for 10 trials

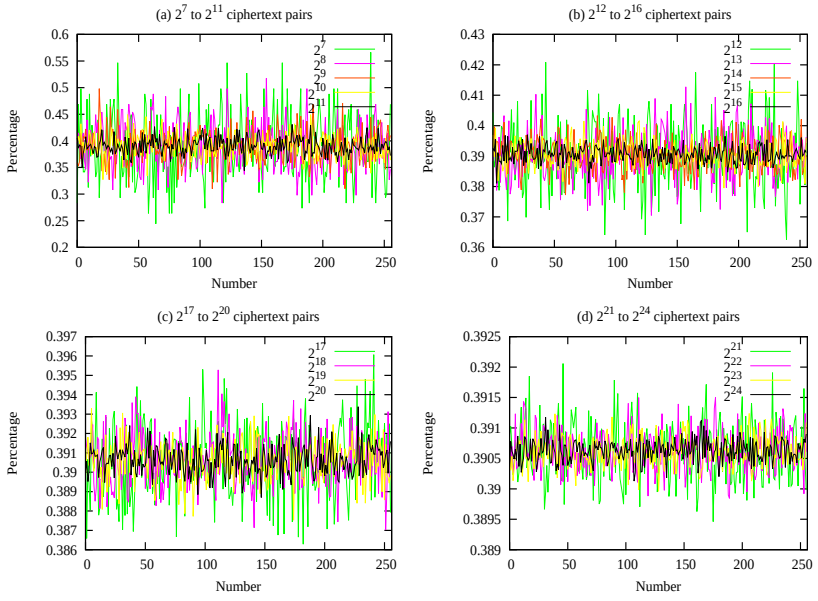

Figure B.124: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 16-round DES for 10 trials

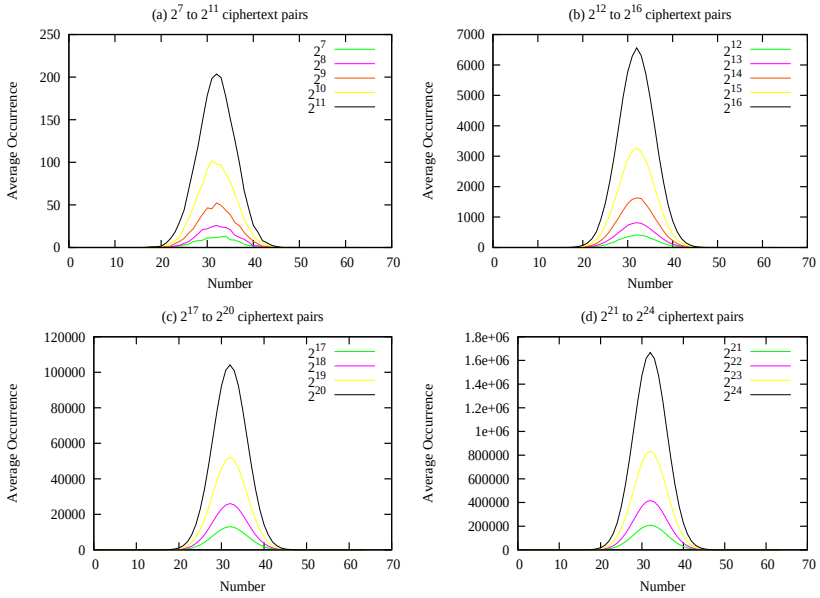

Figure B.125: Average Hamming Weight Distribution between ciphertext pairs for 16-round DES for 10 trials (Autofeeding)

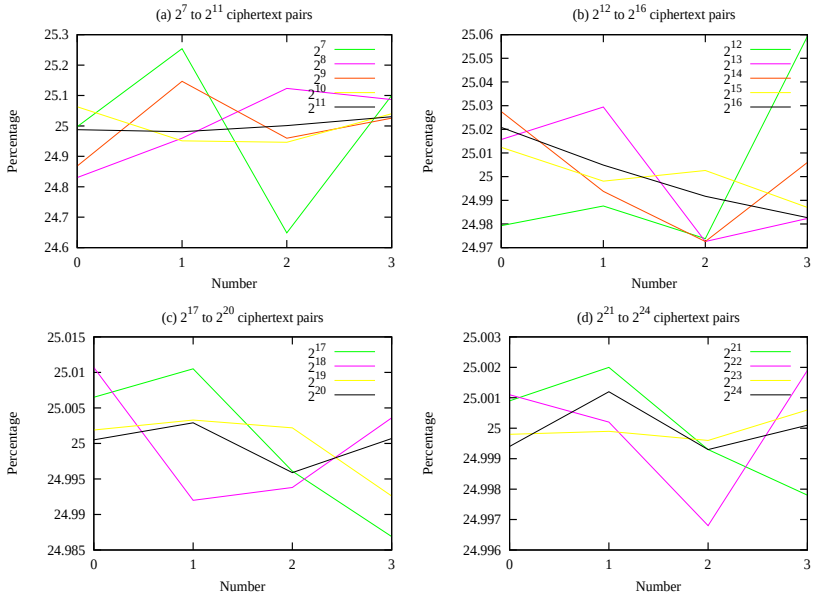

Figure B.126: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 16-round DES for 10 trials (Autofeeding)

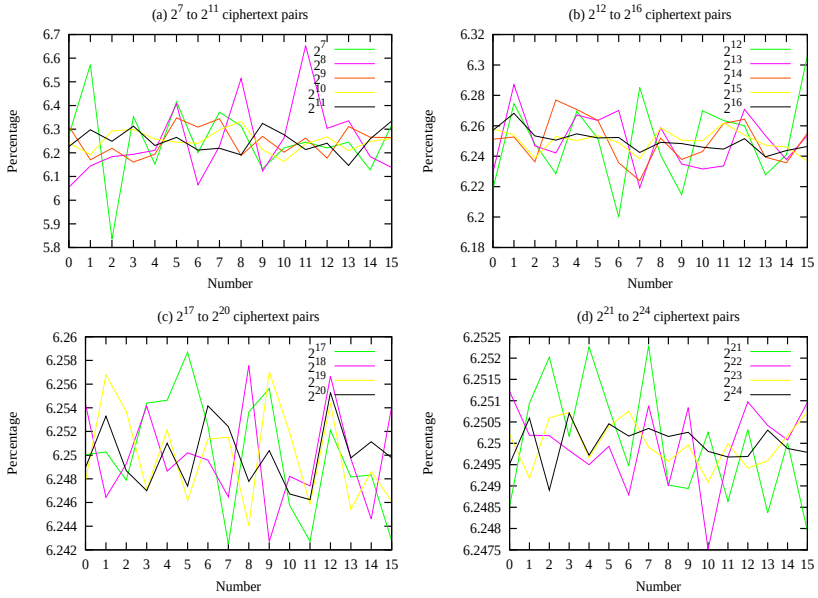

Figure B.127: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 16-round DES for 10 trials (Autofeeding)

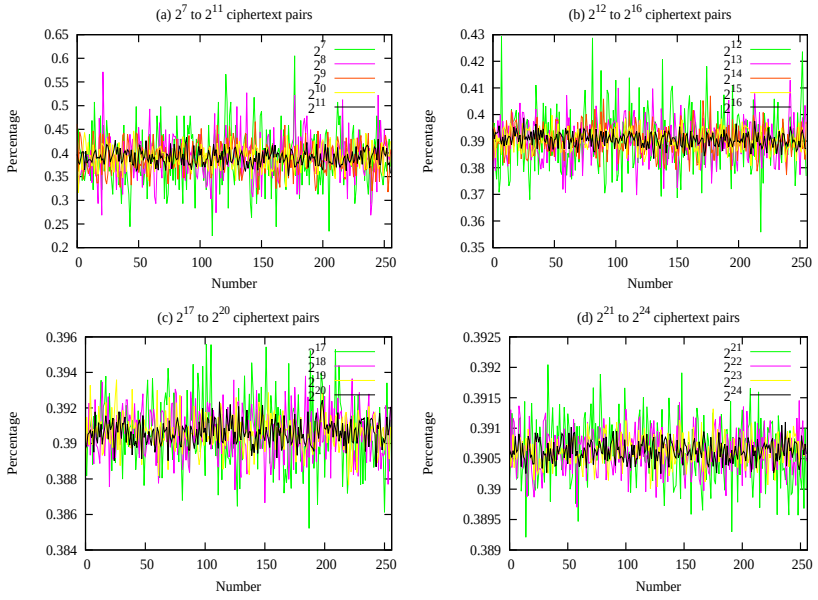

Figure B.128: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 16-round DES for 10 trials (Autofeeding)

C Experimental Test Results for RC5

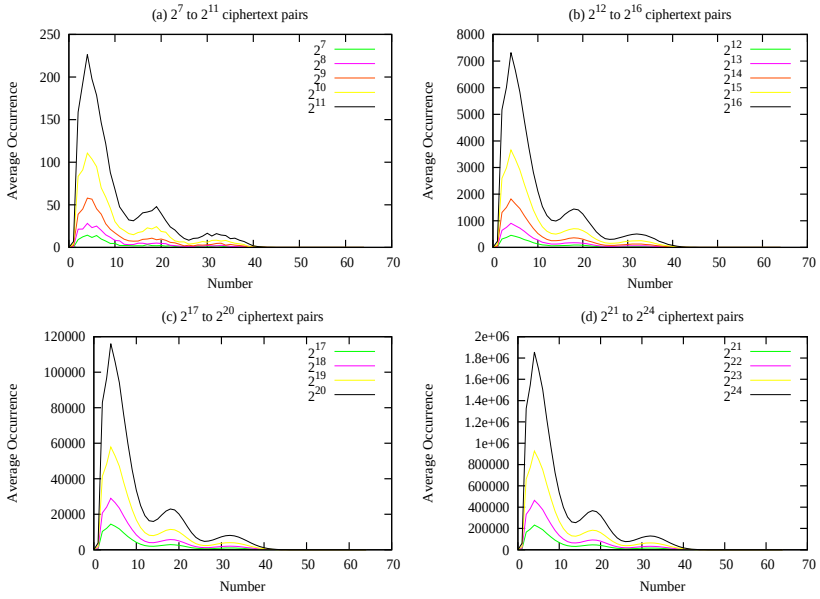

Figure C.1: Average Hamming Weight Distribution between ciphertext pairs for 1-round RC5 for 10 trials

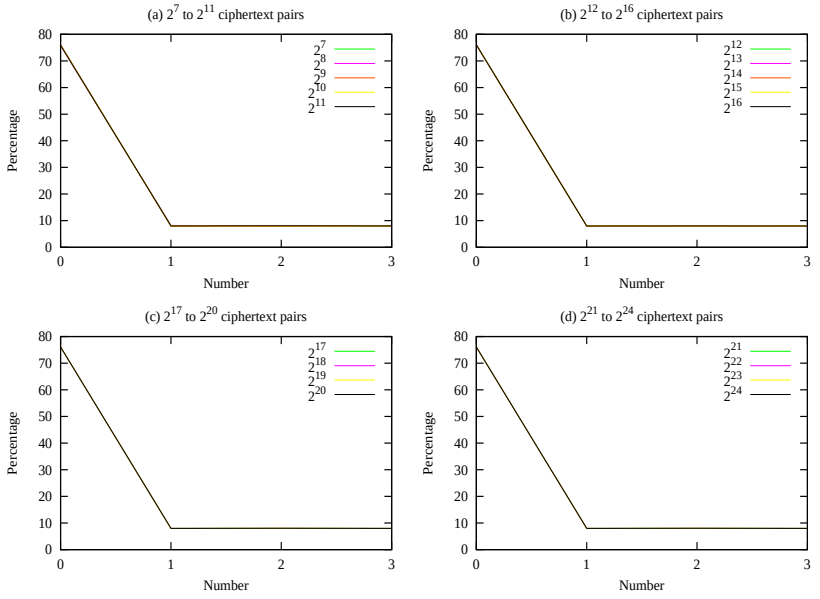

Figure C.2: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 1-round RC5 for 10 trials

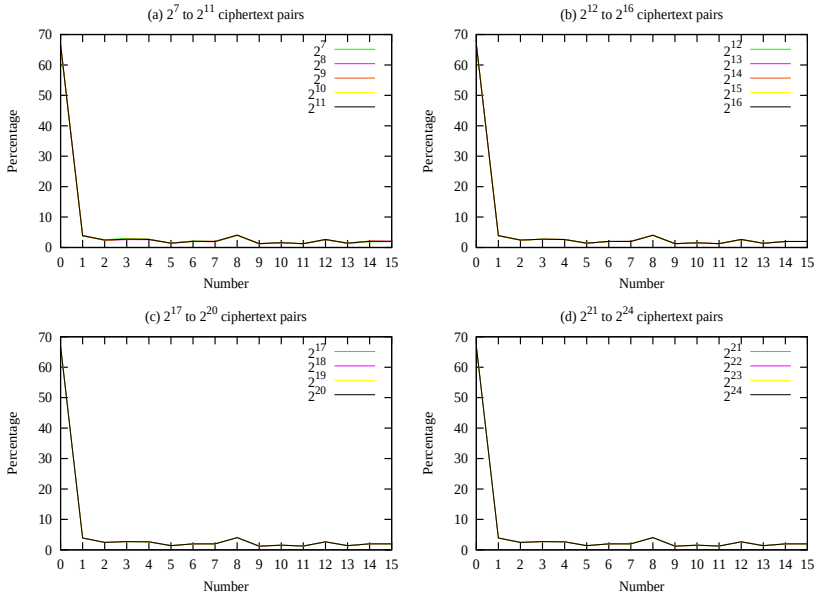

Figure C.3: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 1-round RC5 for 10 trials

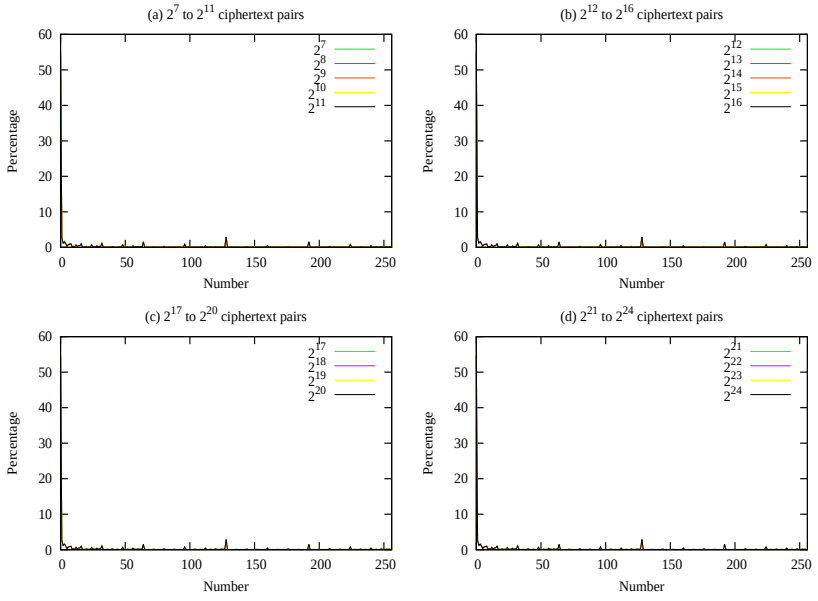

Figure C.4: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 1-round RC5 for 10 trials

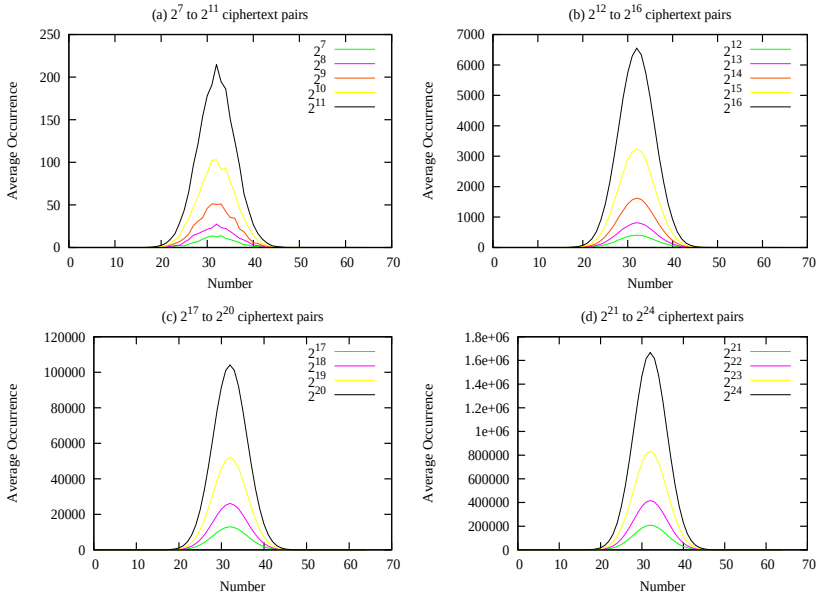

Figure C.5: Average Hamming Weight Distribution between ciphertext pairs for 1-round RC5 for 10 trials (Autofeeding)

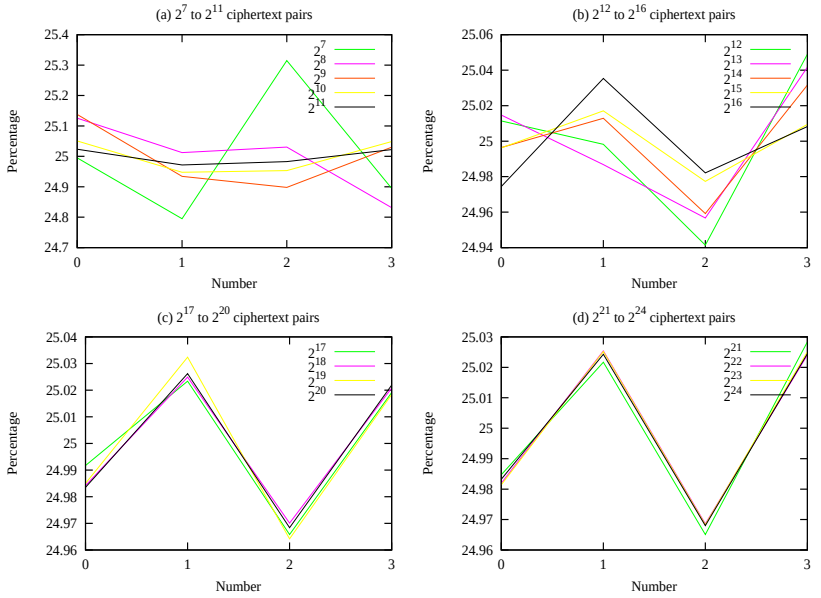

Figure C.6: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 1-round RC5 for 10 trials (Autofeeding)

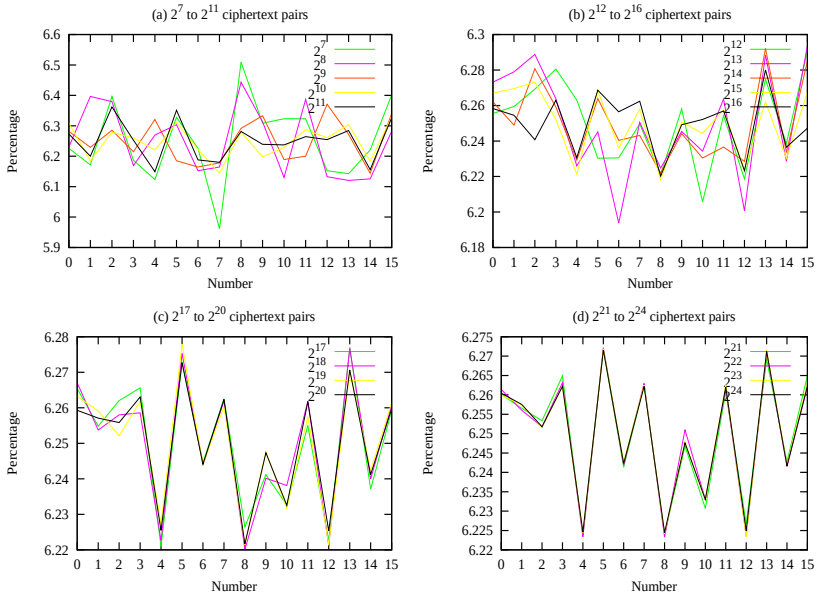

Figure C.7: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 1-round RC5 for 10 trials (Autofeeding)

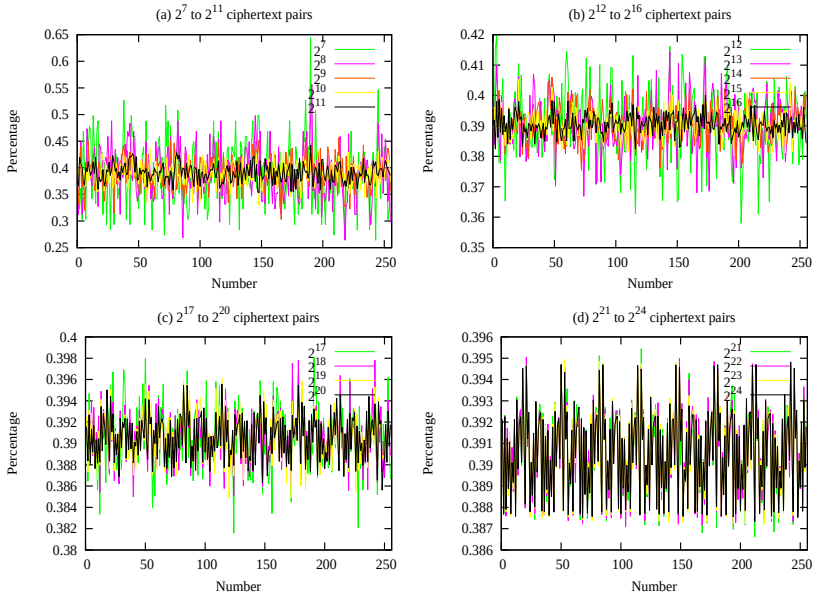

Figure C.8: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 1-round RC5 for 10 trials (Autofeeding)

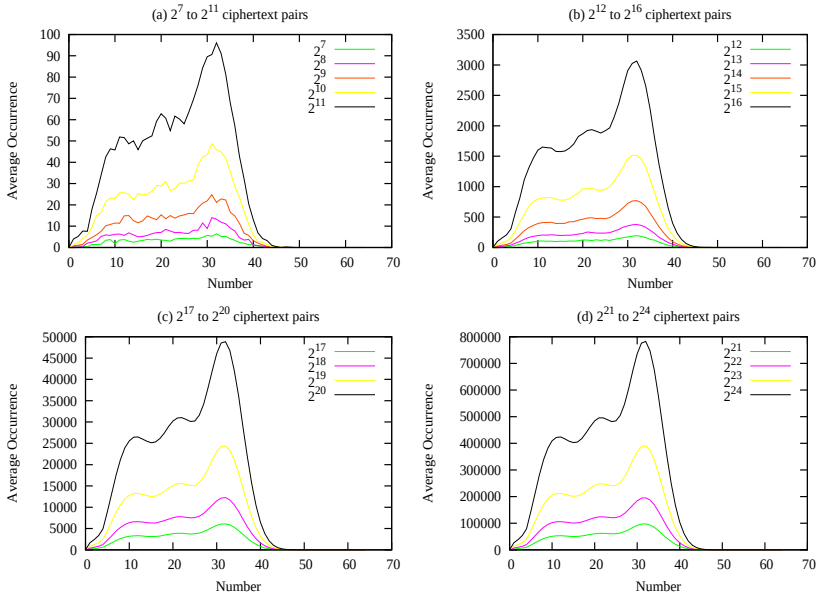

Figure C.9: Average Hamming Weight Distribution between ciphertext pairs for 2-round RC5 for 10 trials

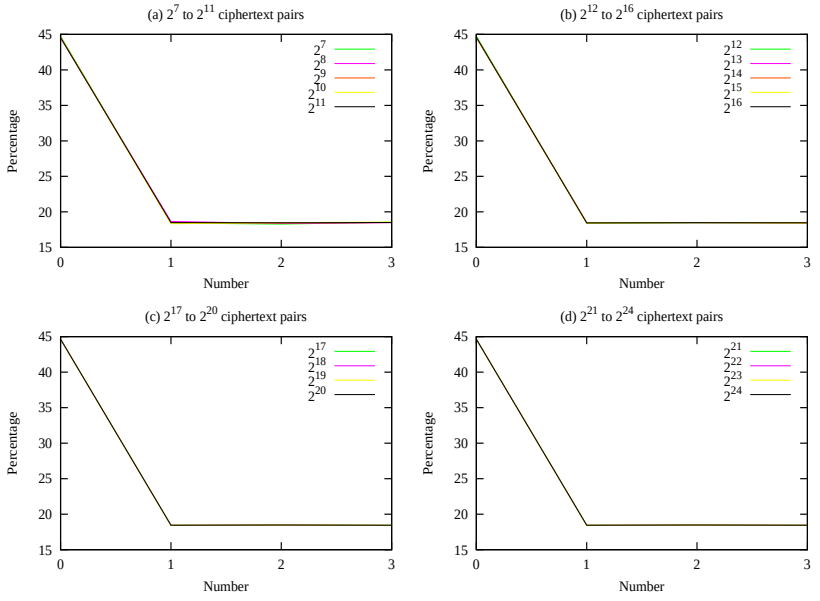

Figure C.10: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 2-round RC5 for 10 trials

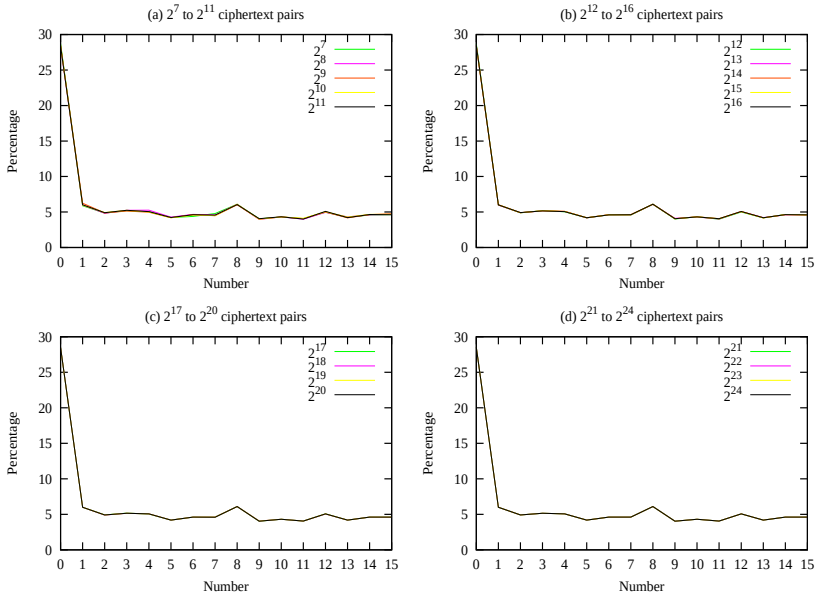

Figure C.11: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 2-round RC5 for 10 trials

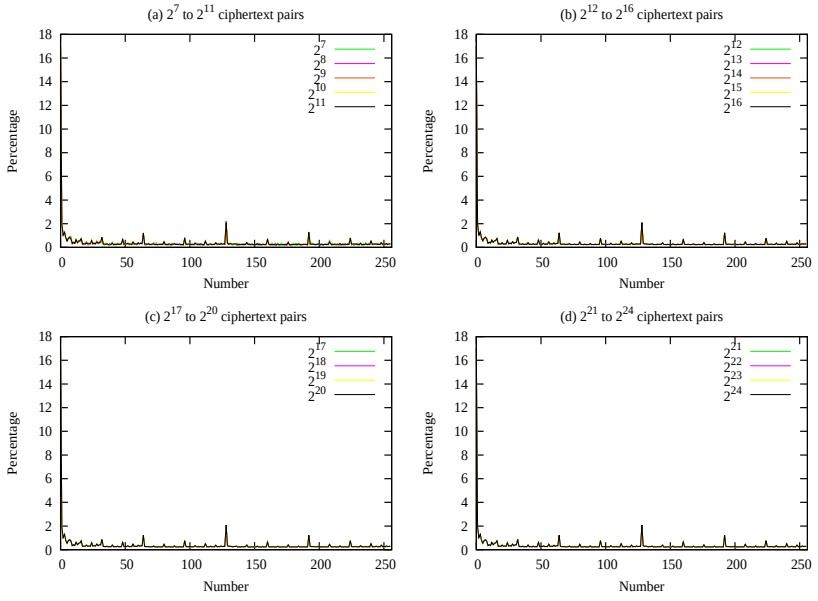

Figure C.12: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 2-round RC5 for 10 trials

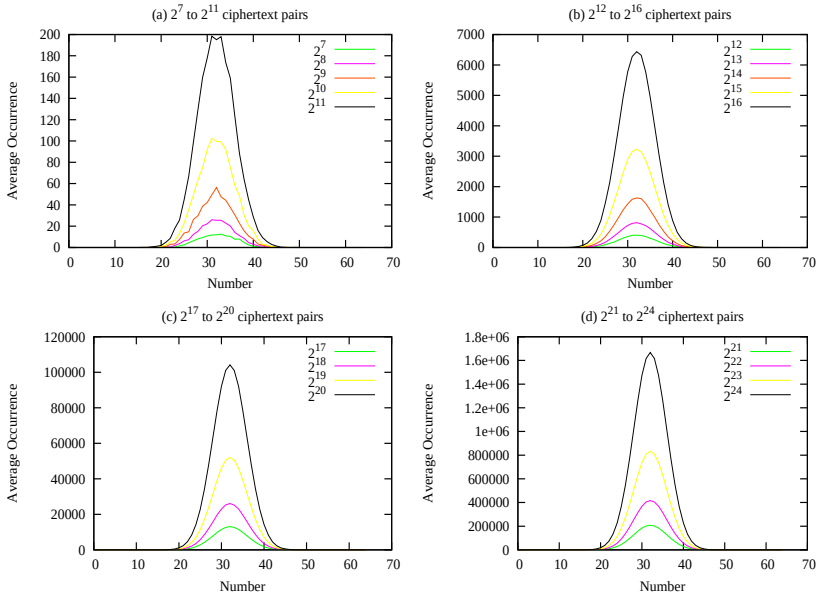

Figure C.13: Average Hamming Weight Distribution between ciphertext pairs for 2-round RC5 for 10 trials (Autofeeding)

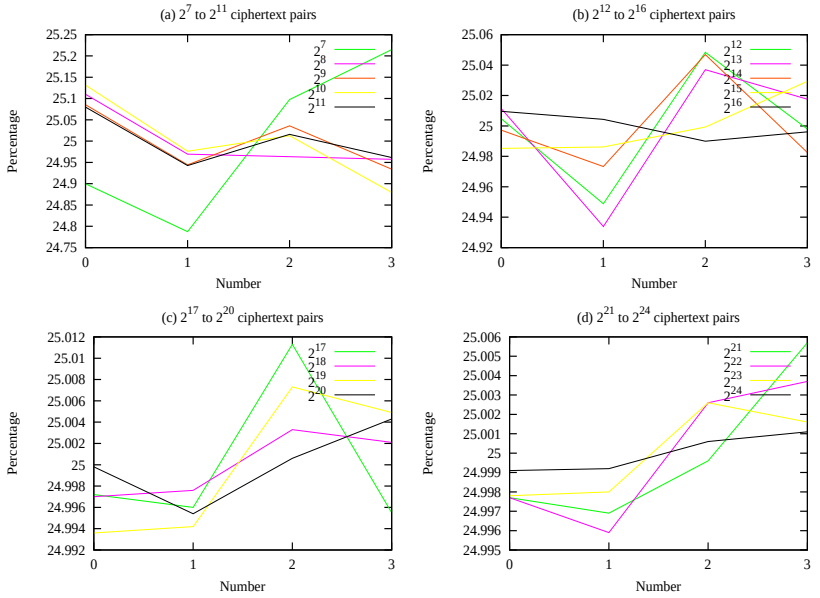

Figure C.14: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 2-round RC5 for 10 trials (Autofeeding)

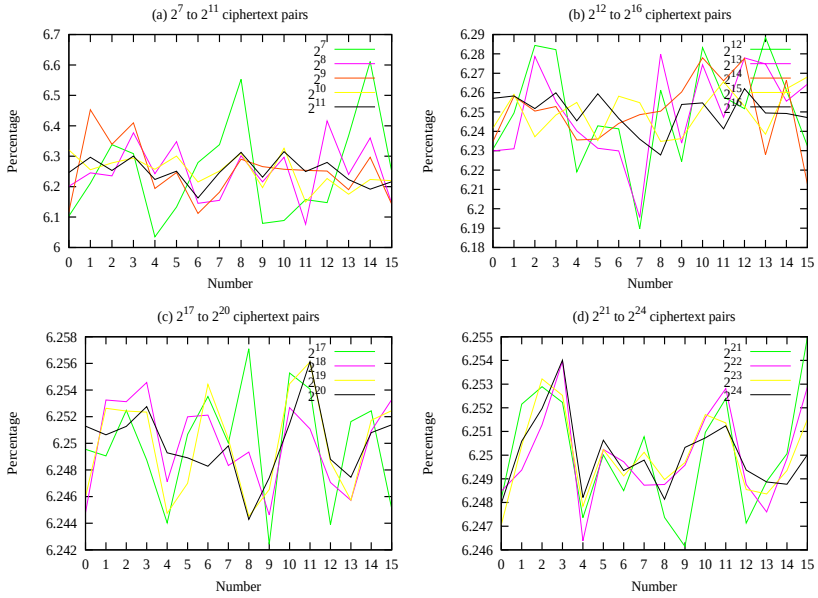

Figure C.15: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 2-round RC5 for 10 trials (Autofeeding)

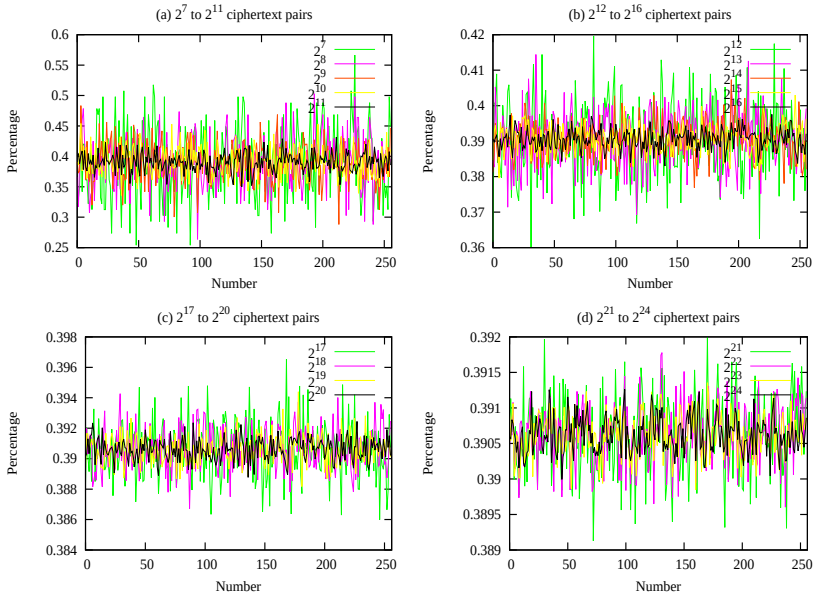

Figure C.16: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 2-round RC5 for 10 trials (Autofeeding)

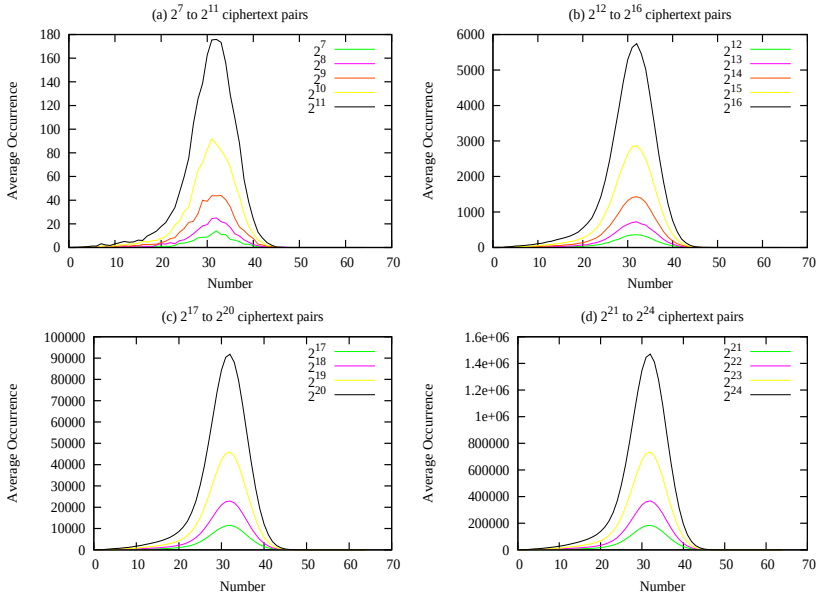

Figure C.17: Average Hamming Weight Distribution between ciphertext pairs for 3-round RC5 for 10 trials

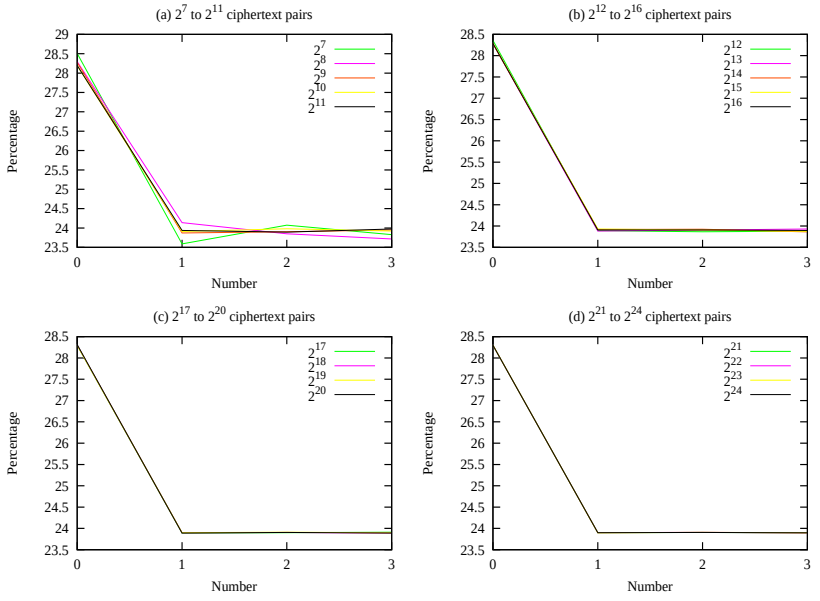

Figure C.18: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 3-round RC5 for 10 trials

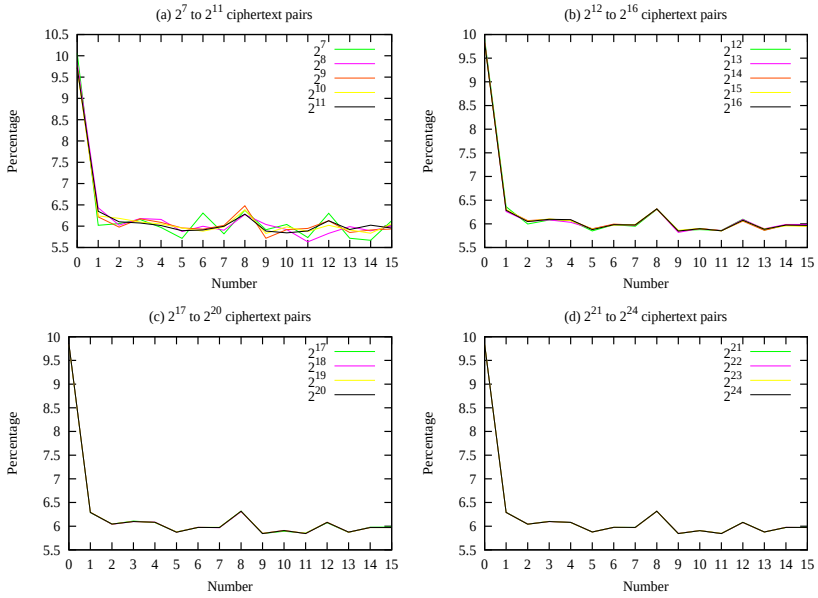

Figure C.19: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 3-round RC5 for 10 trials

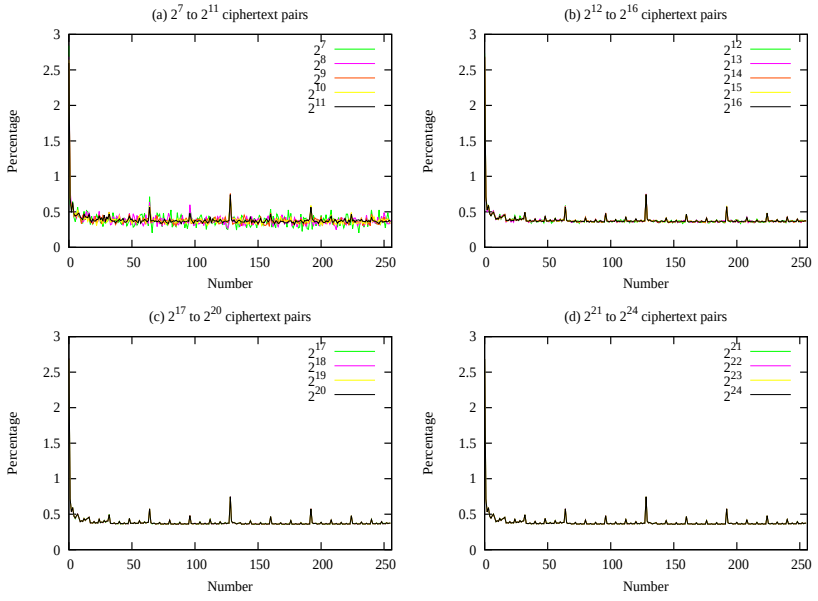

Figure C.20: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 3-round RC5 for 10 trials

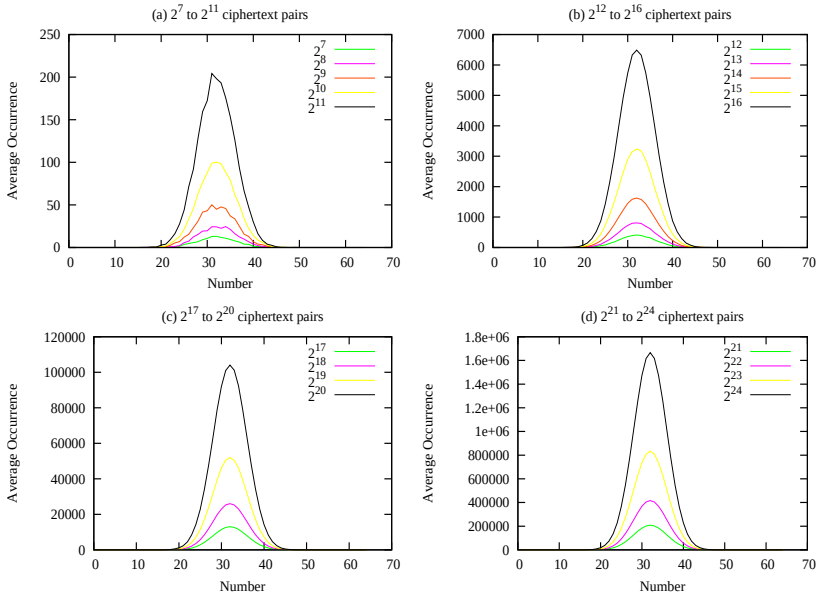

Figure C.21: Average Hamming Weight Distribution between ciphertext pairs for 3-round RC5 for 10 trials (Autofeeding)

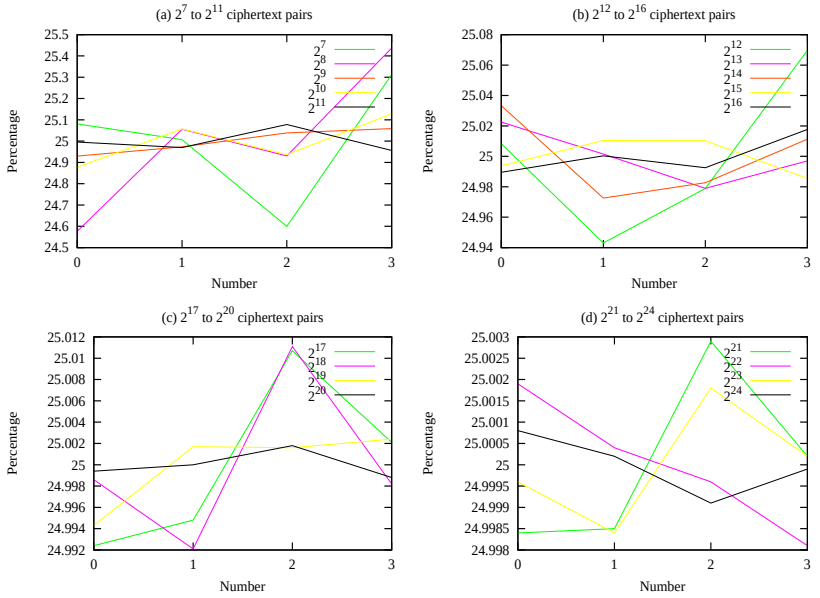

Figure C.22: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 3-round RC5 for 10 trials (Autofeeding)

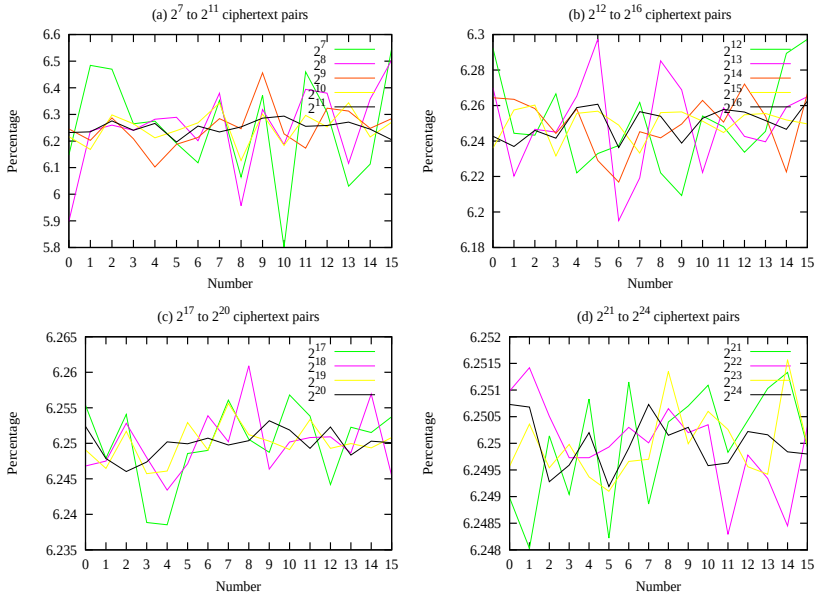

Figure C.23: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 3-round RC5 for 10 trials (Autofeeding)

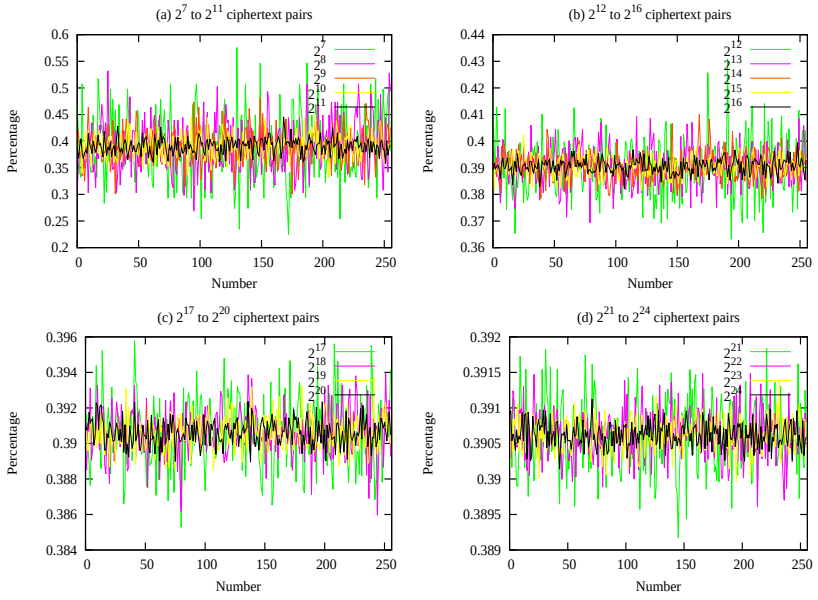

Figure C.24: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 3-round RC5 for 10 trials (Autofeeding)

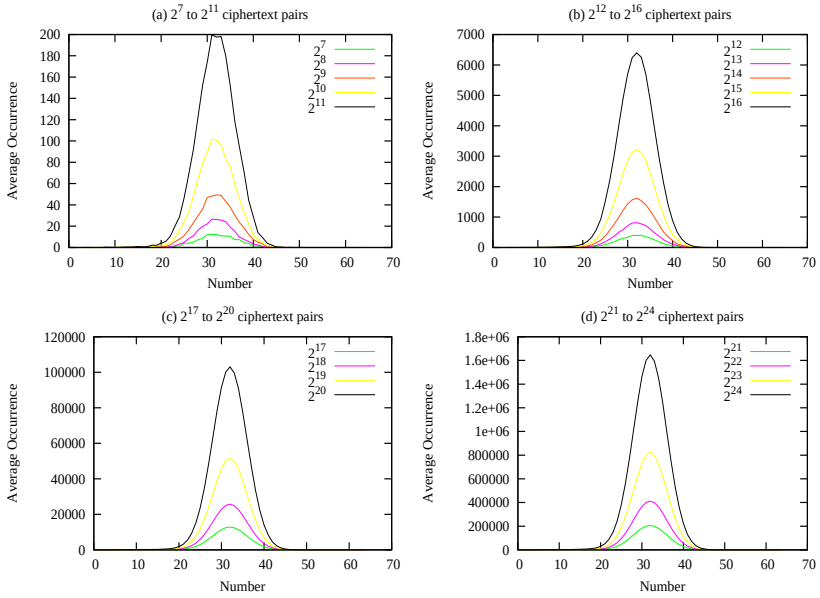

Figure C.25: Average Hamming Weight Distribution between ciphertext pairs for 4-round RC5 for 10 trials

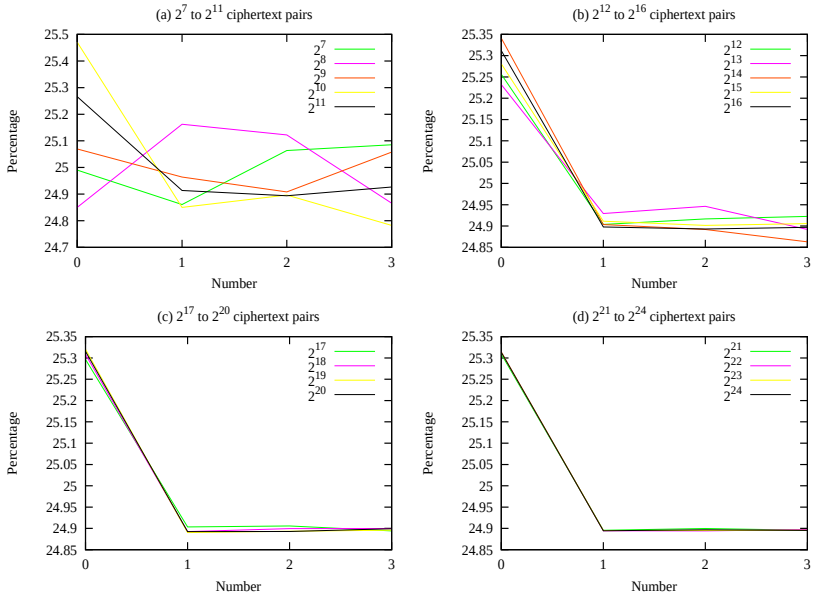

Figure C.26: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 4-round RC5 for 10 trials

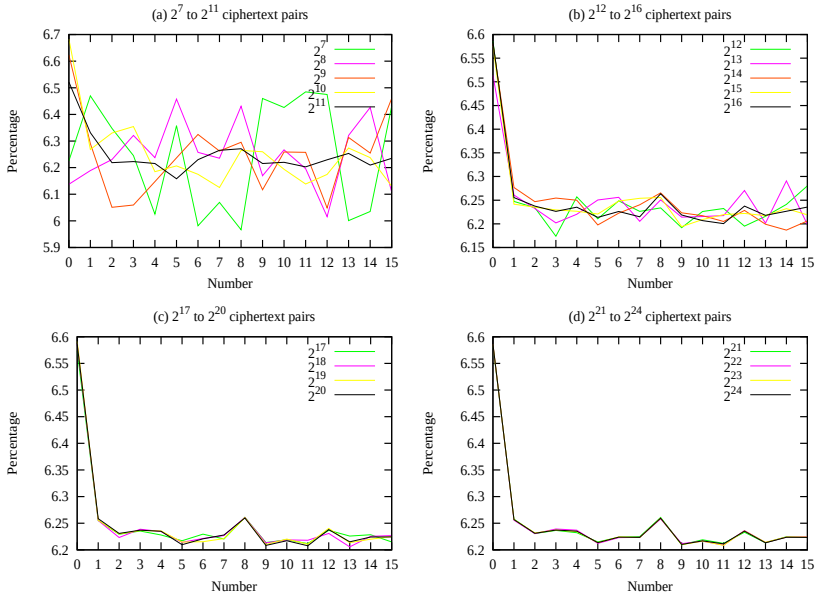

Figure C.27: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 4-round RC5 for 40 trials

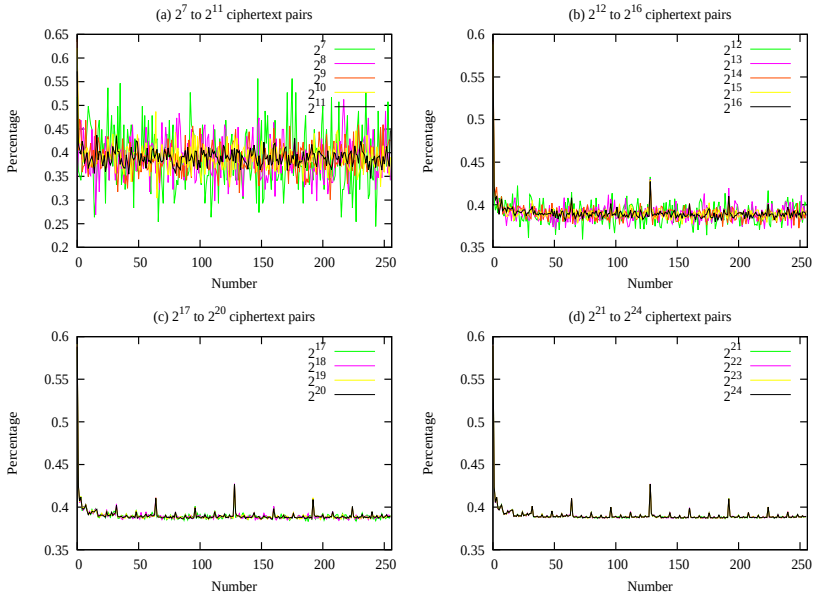

Figure C.28: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 4-round RC5 for 10 trials

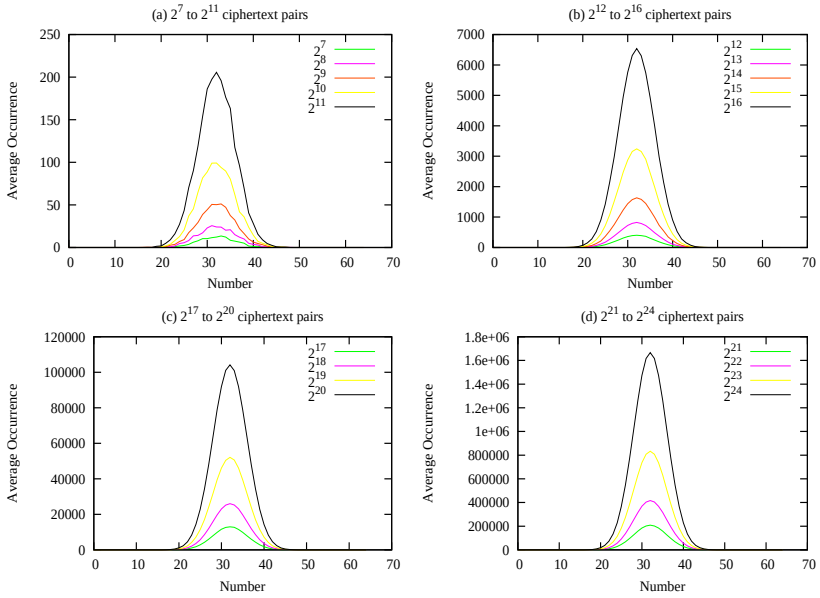

Figure C.29: Average Hamming Weight Distribution between ciphertext pairs for 4-round RC5 for 10 trials (Autofeeding)

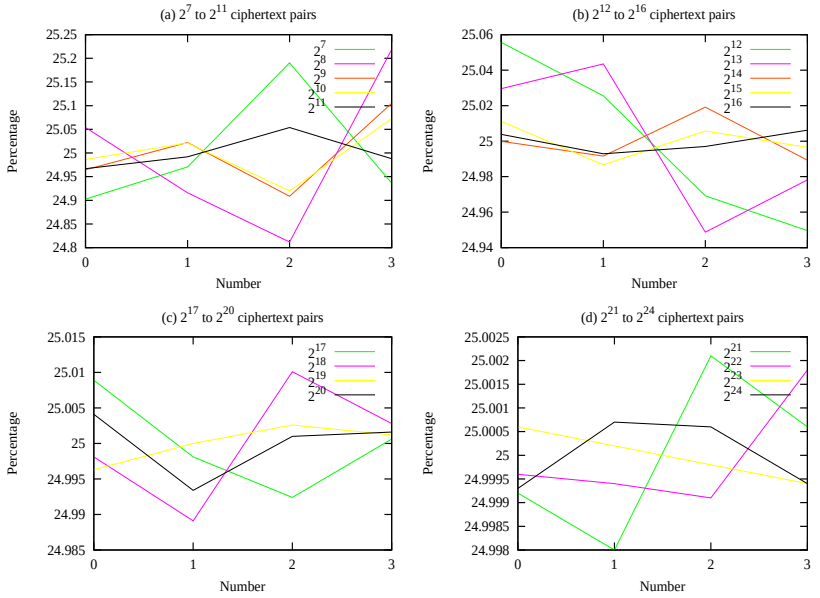

Figure C.30: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 4-round RC5 for 10 trials (Autofeeding)

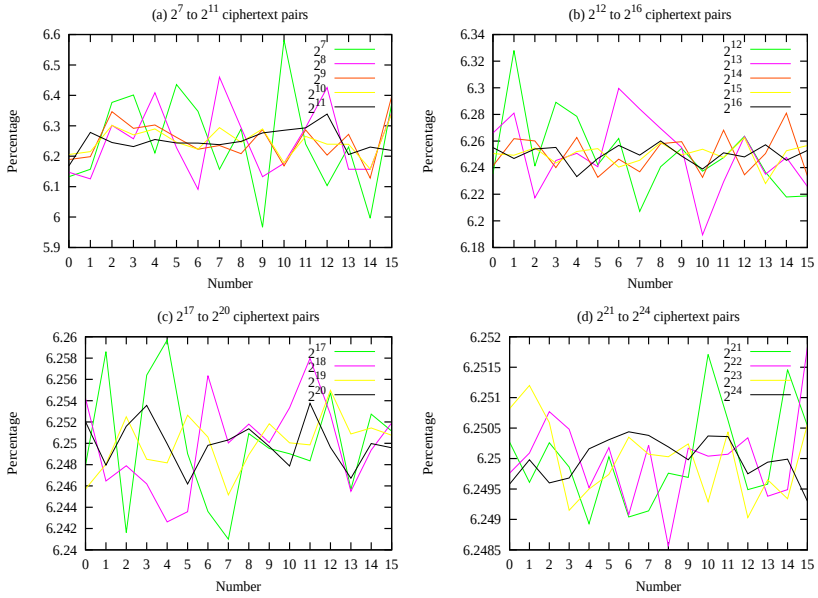

Figure C.31: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 4-round RC5 for 10 trials (Autofeeding)

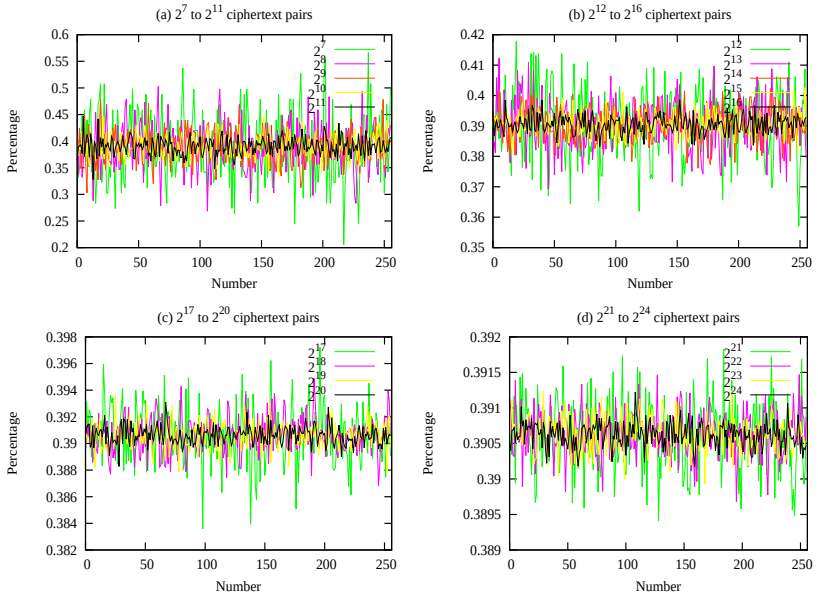

Figure C.32: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 4-round RC5 for 10 trials (Autofeeding)

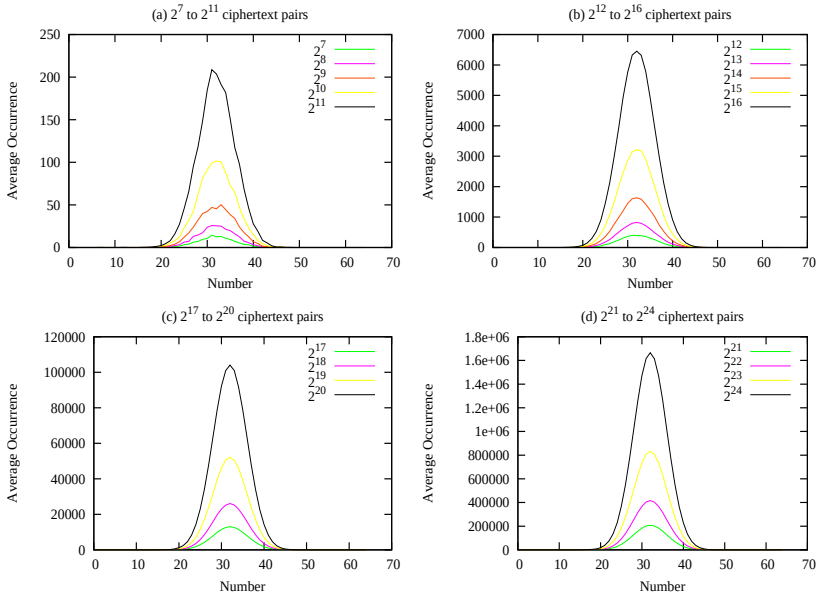

Figure C.33: Average Hamming Weight Distribution between ciphertext pairs for 5-round RC5 for 10 trials

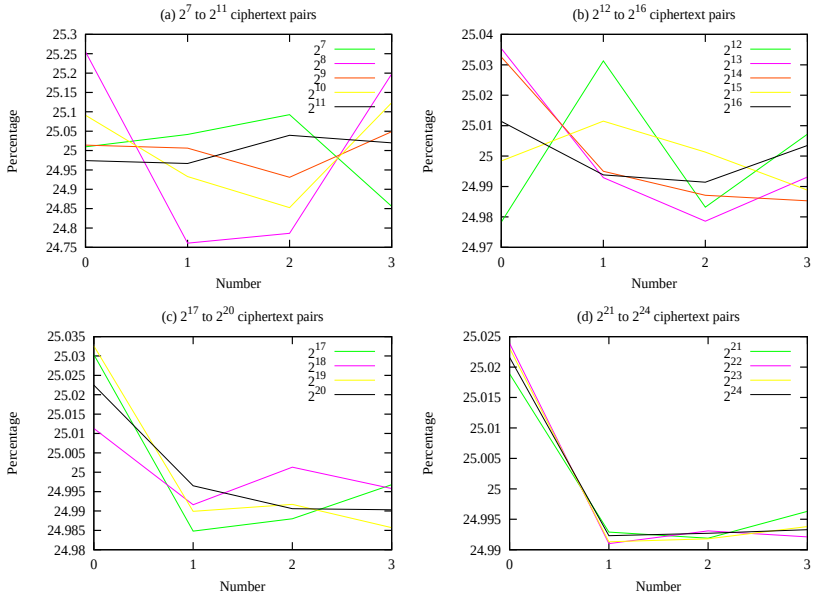

Figure C.34: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 5-round RC5 for 10 trials

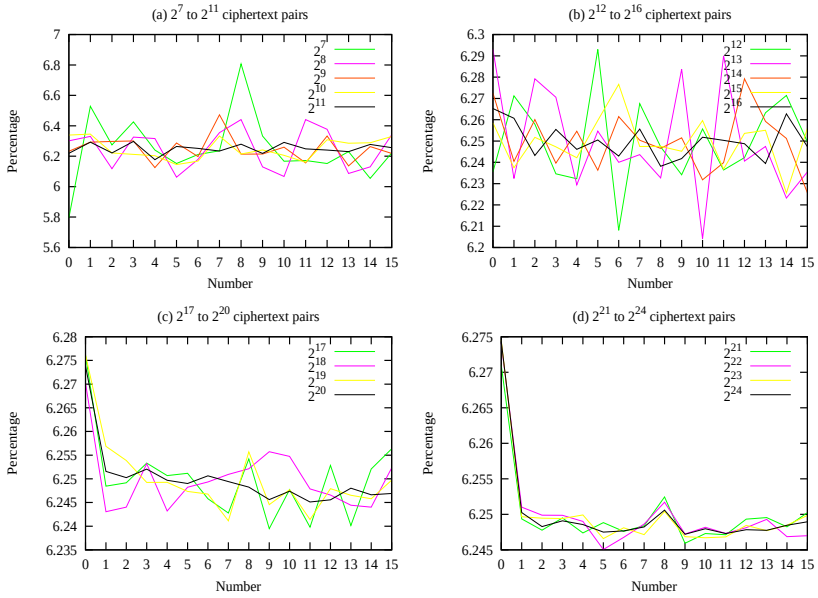

Figure C.35: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 5-round RC5 for 10 trials

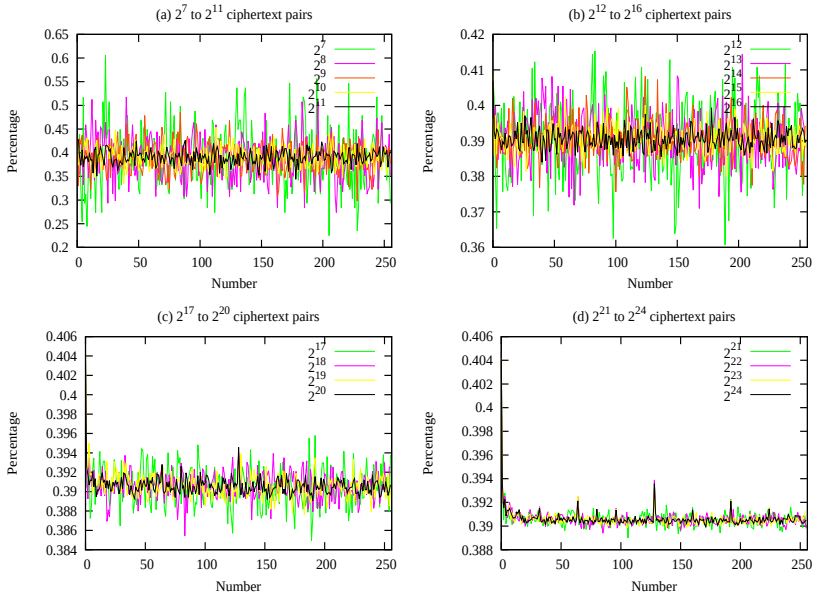

Figure C.36: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 5-round RC5 for 10 trials

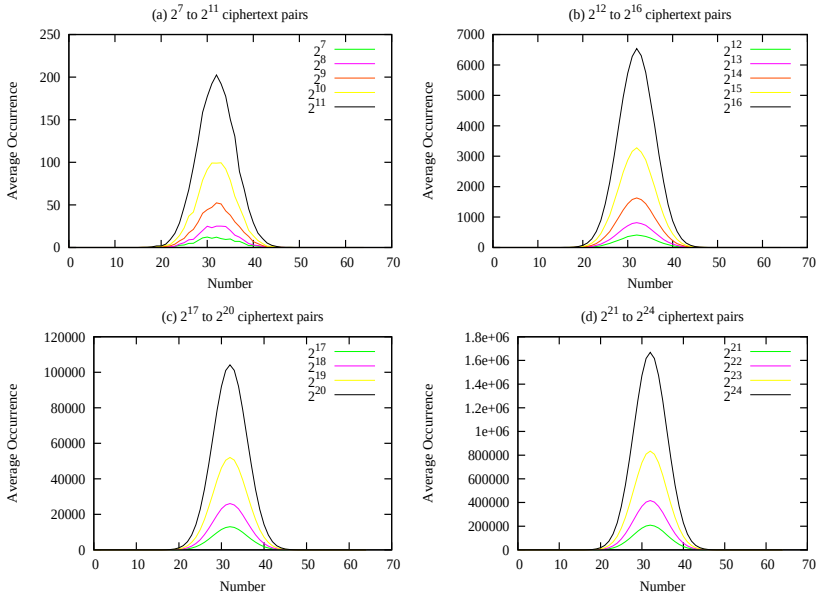

Figure C.37: Average Hamming Weight Distribution between ciphertext pairs for 5-round RC5 for 10 trials (Autofeeding)

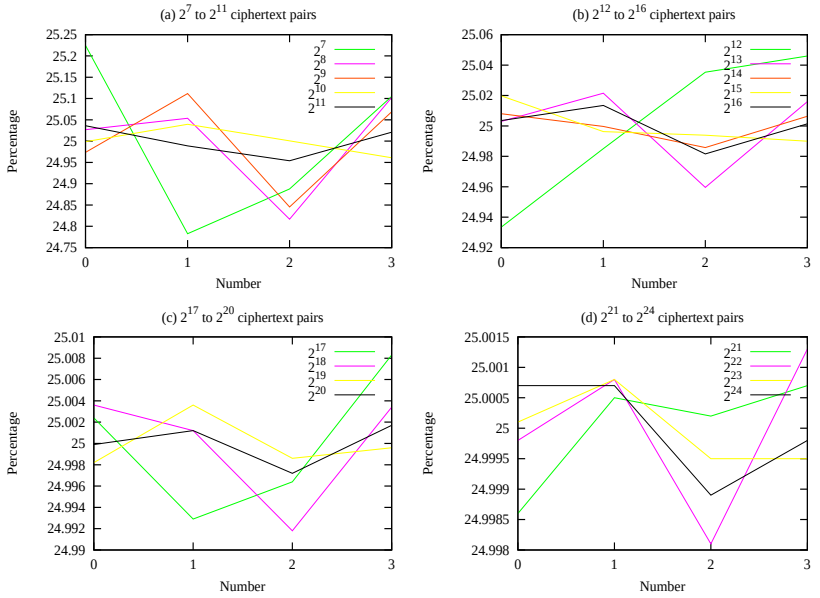

Figure C.38: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 5-round RC5 for 10 trials (Autofeeding)

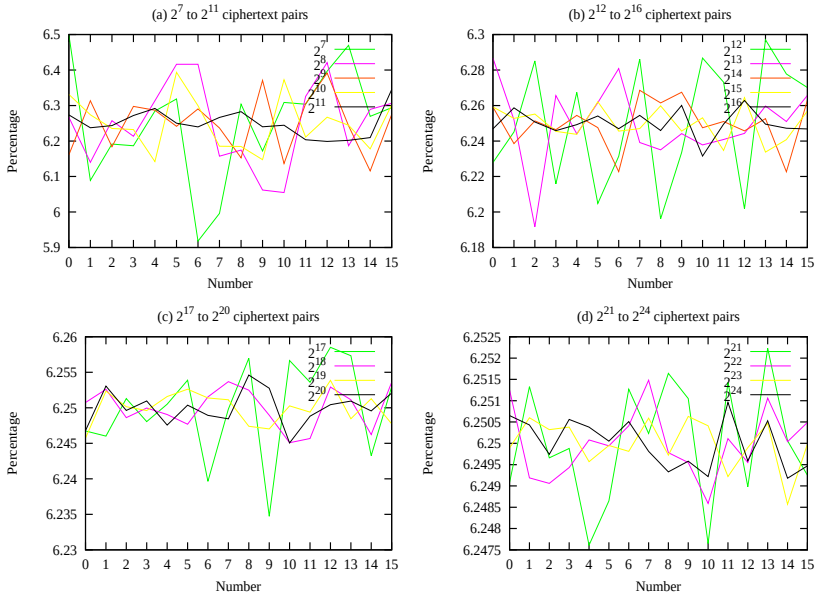

Figure C.39: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 5-round RC5 for 10 trials (Autofeeding)

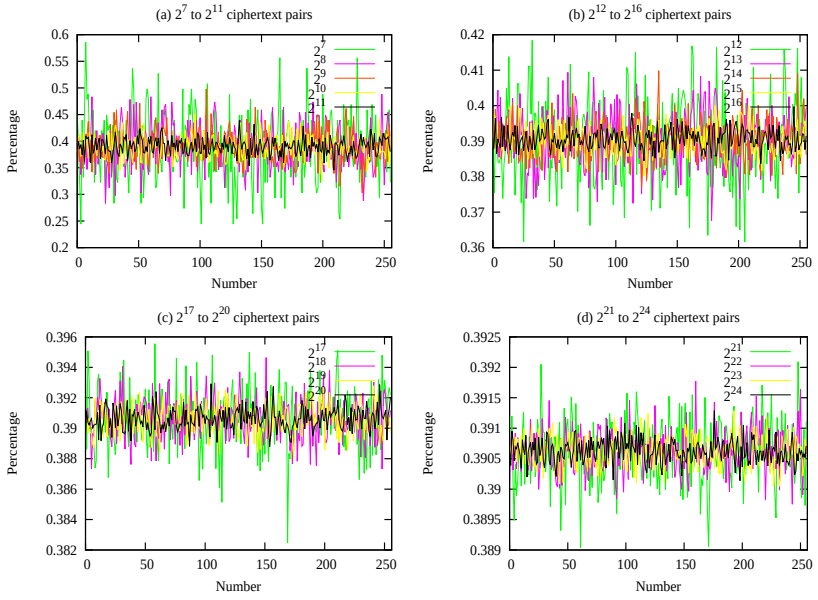

Figure C.40: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 5-round RC5 for 10 trials (Autofeeding)

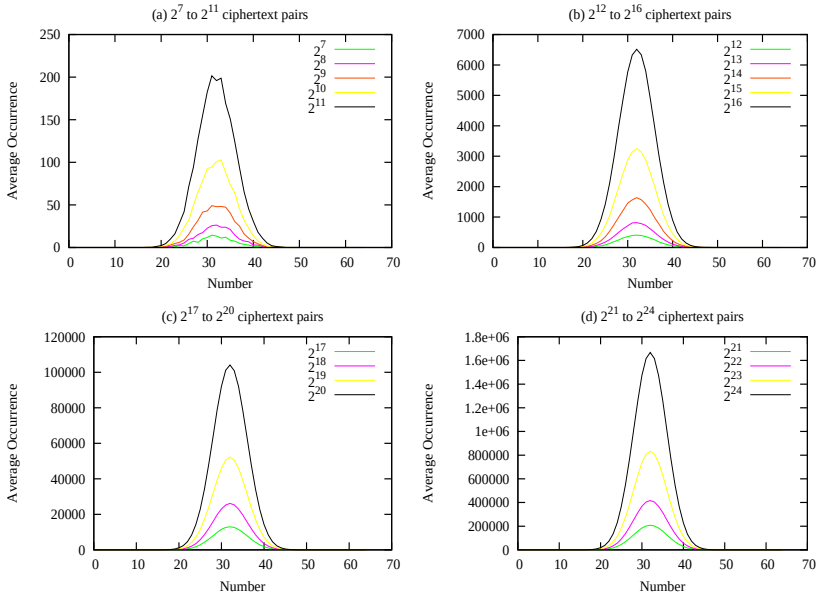

Figure C.41: Average Hamming Weight Distribution between ciphertext pairs for 6-round RC5 for 10 trials

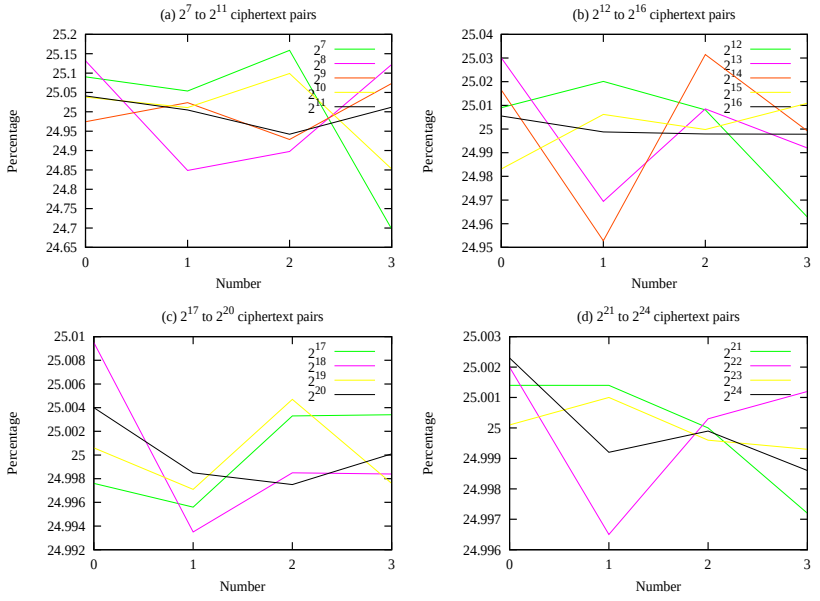

Figure C.42: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 6-round RC5 for 10 trials

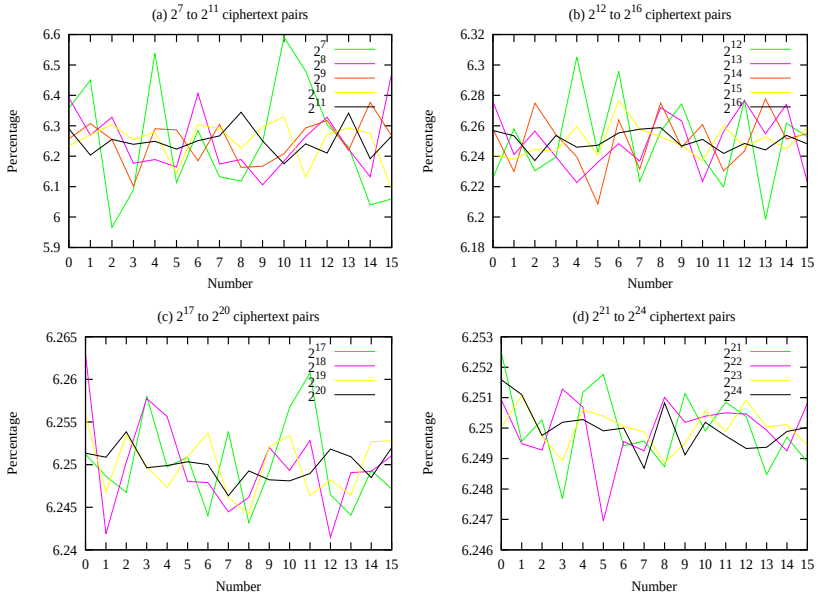

Figure C.43: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 6-round RC5 for 10 trials

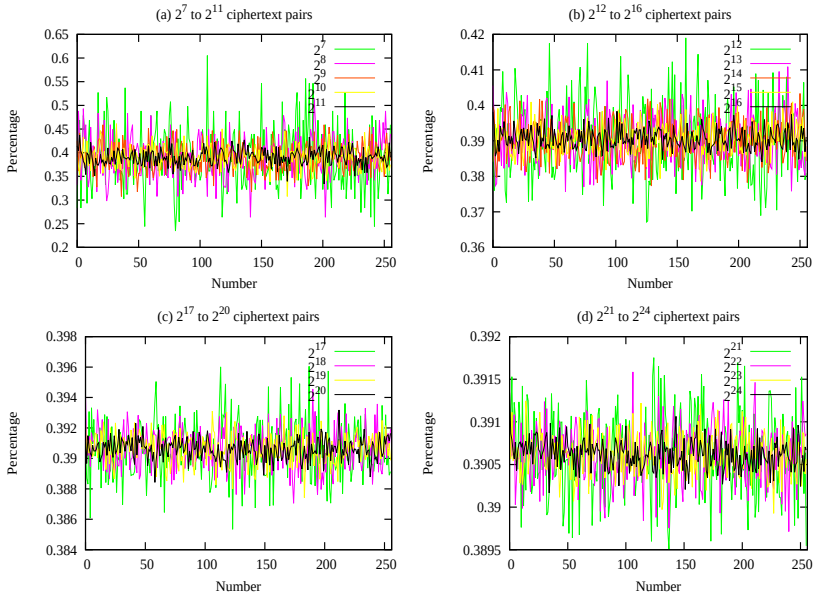

Figure C.44: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 6-round RC5 for 10 trials

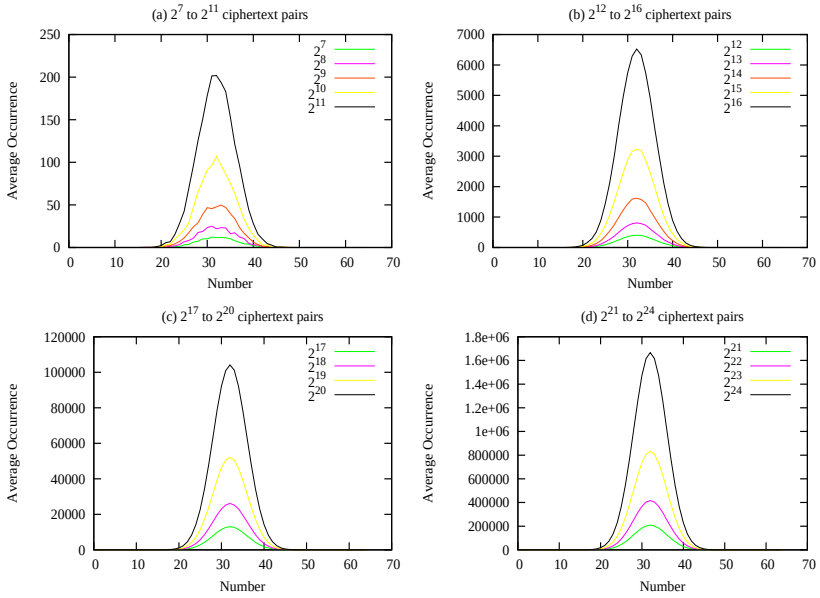

Figure C.45: Average Hamming Weight Distribution between ciphertext pairs for 6-round RC5 for 10 trials (Autofeeding)

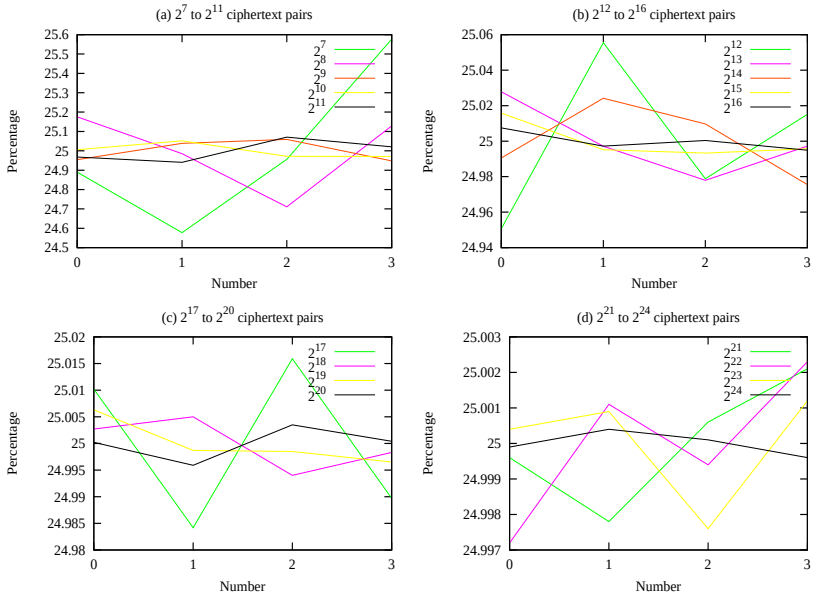

Figure C.46: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 6-round RC5 for 10 trials (Autofeeding)

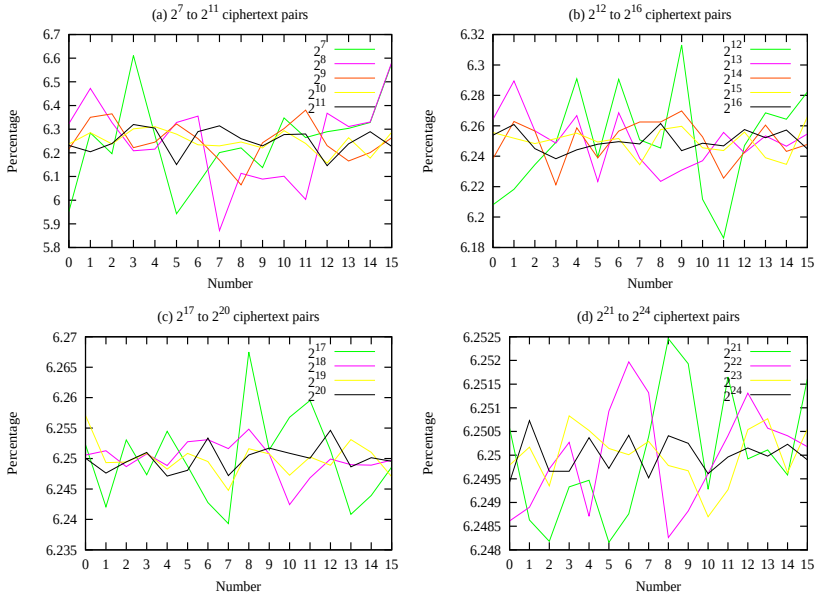

Figure C.47: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 6-round RC5 for 10 trials (Autofeeding)

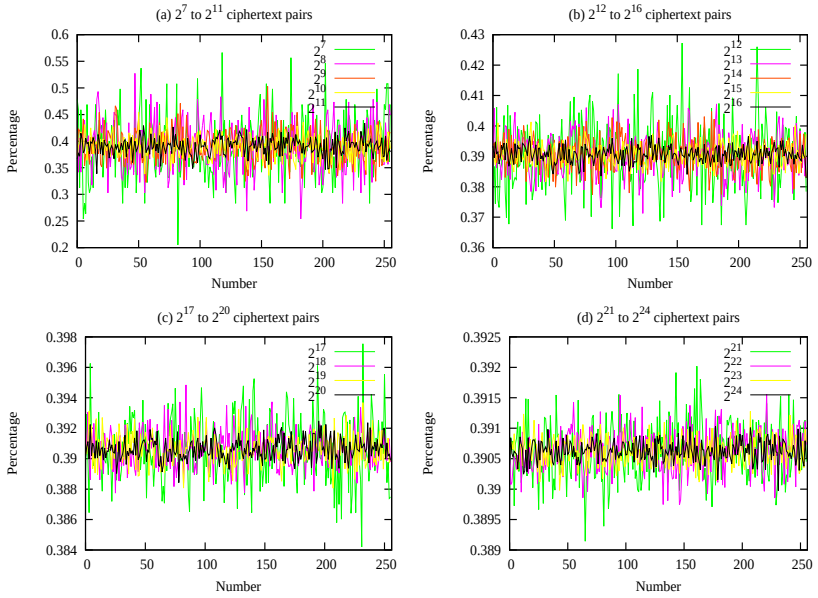

Figure C.48: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 6-round RC5 for 10 trials (Autofeeding)

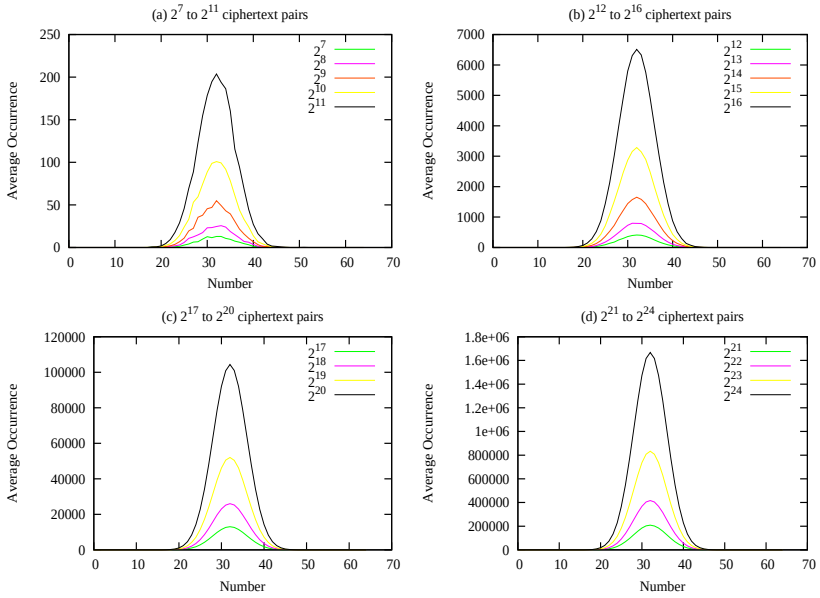

Figure C.49: Average Hamming Weight Distribution between ciphertext pairs for 7-round RC5 for 10 trials

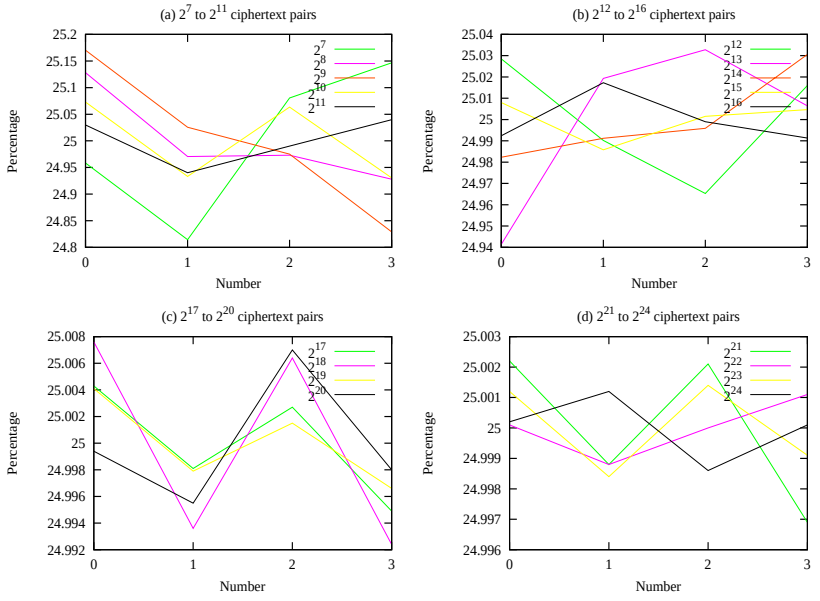

Figure C.50: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 7-round RC5 for 10 trials

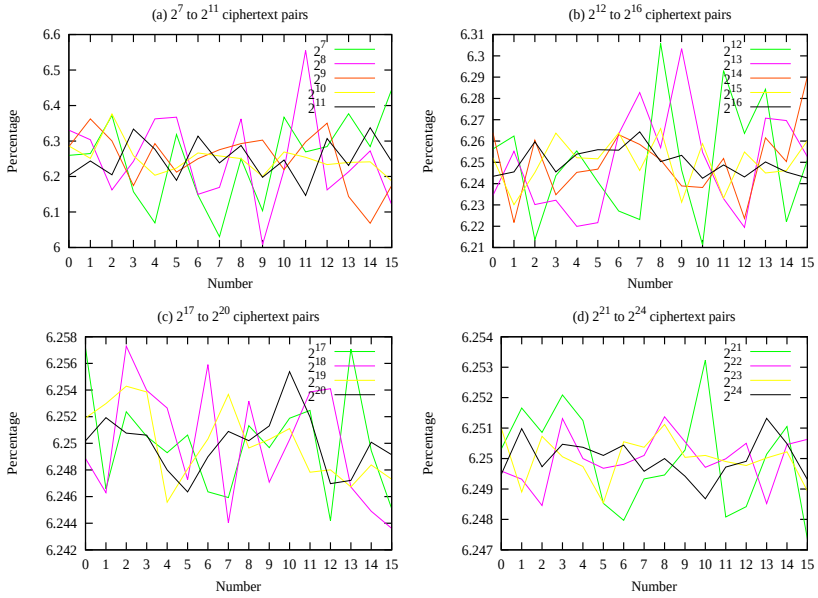

Figure C.51: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 7-round RC5 for 10 trials

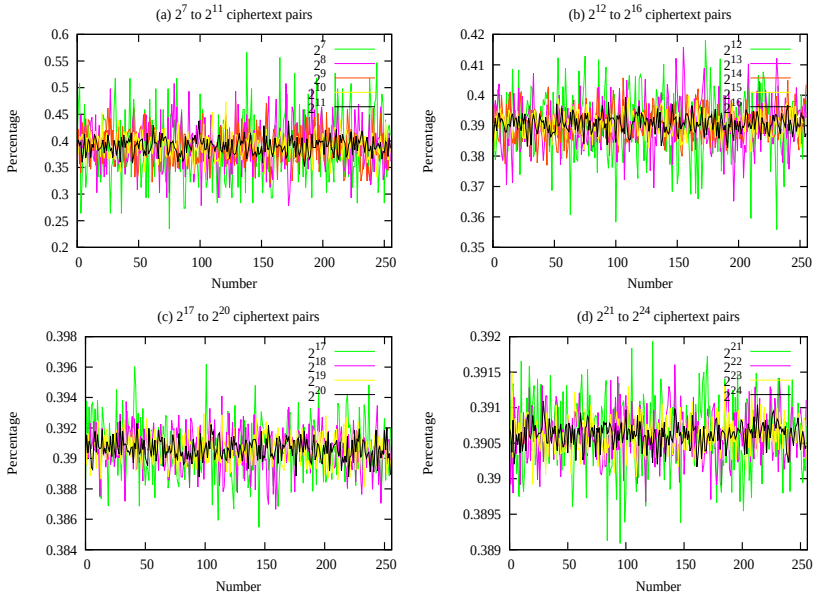

Figure C.52: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 7-round RC5 for 10 trials

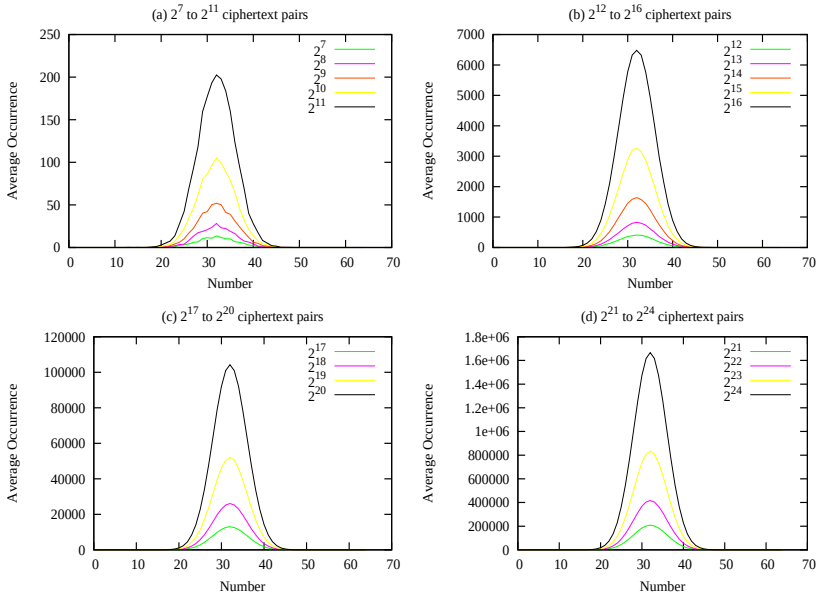

Figure C.53: Average Hamming Weight Distribution between ciphertext pairs for 7-round RC5 for 10 trials (Autofeeding)

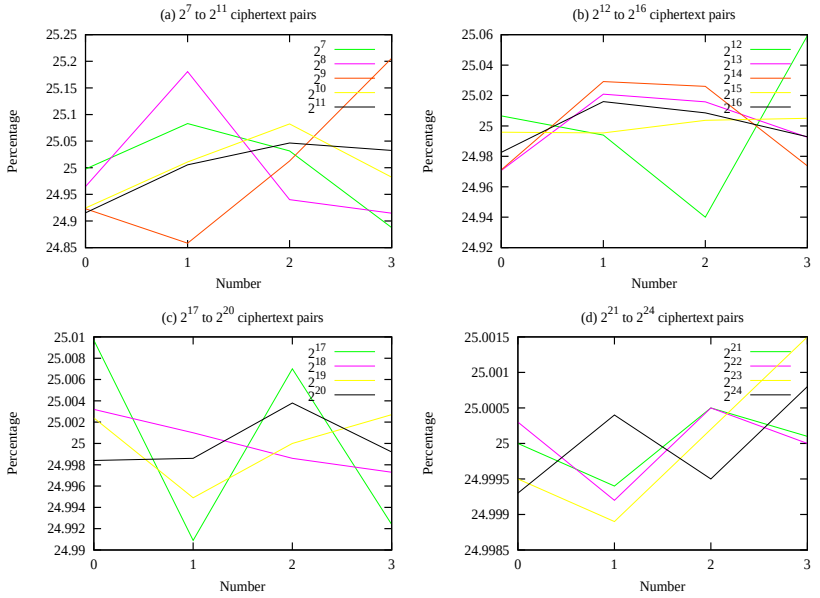

Figure C.54: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 7-round RC5 for 10 trials (Autofeeding)

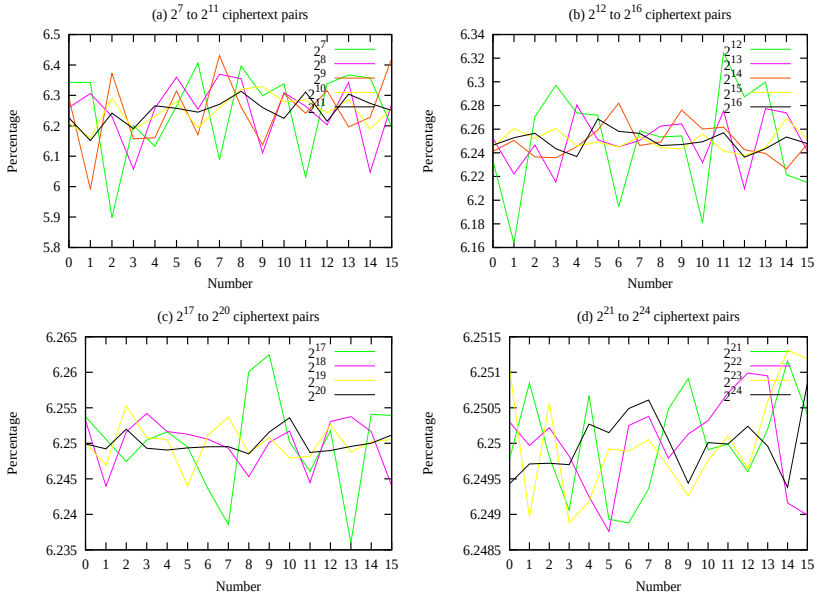

Figure C.55: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 7-round RC5 for 10 trials (Autofeeding)

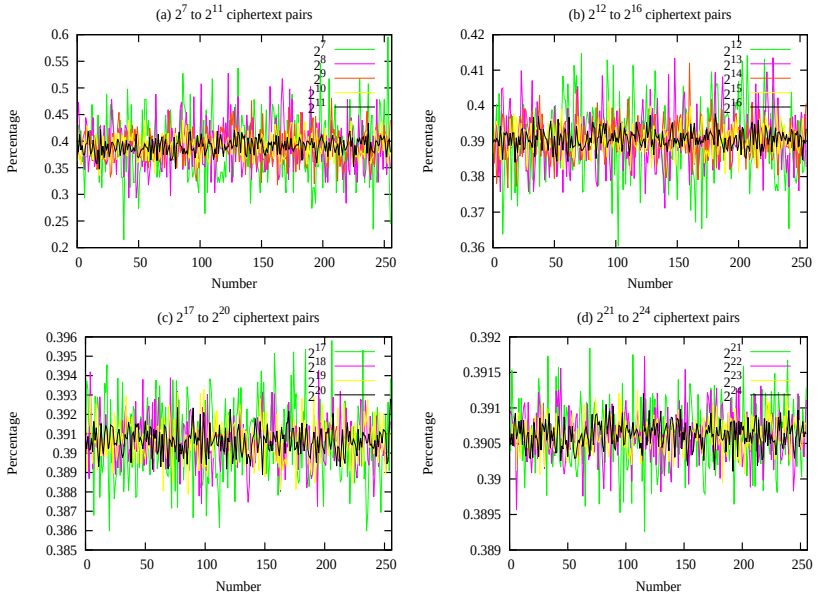

Figure C.56: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 7-round RC5 for 10 trials (Autofeeding)

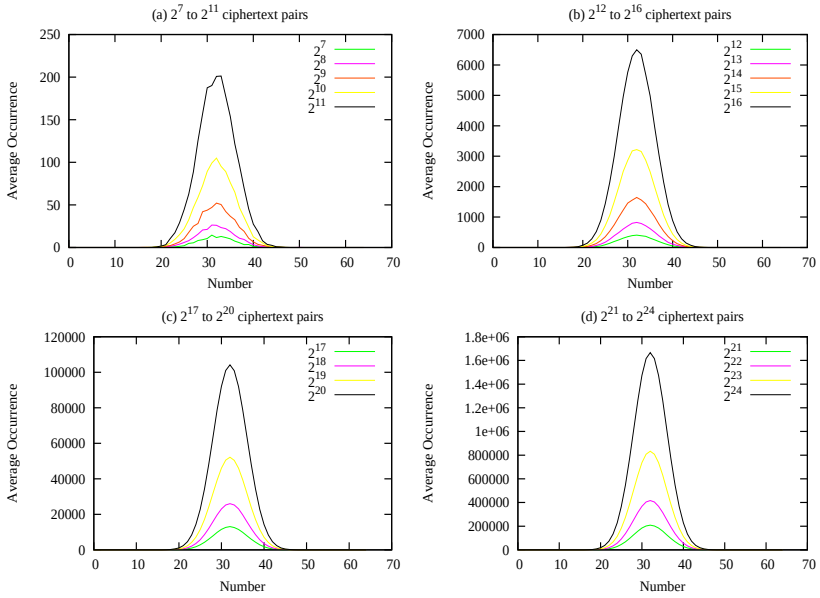

Figure C.57: Average Hamming Weight Distribution between ciphertext pairs for 8-round RC5 for 10 trials

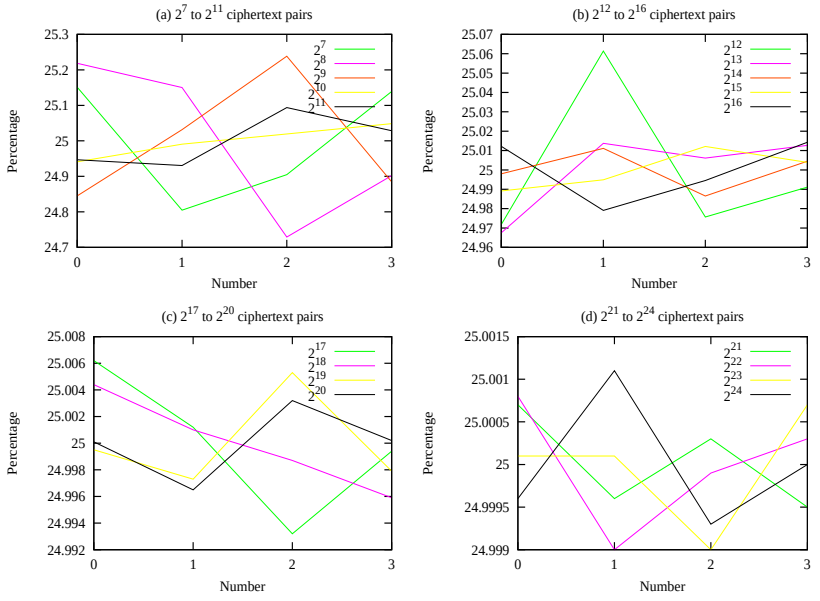

Figure C.58: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 8-round RC5 for 10 trials

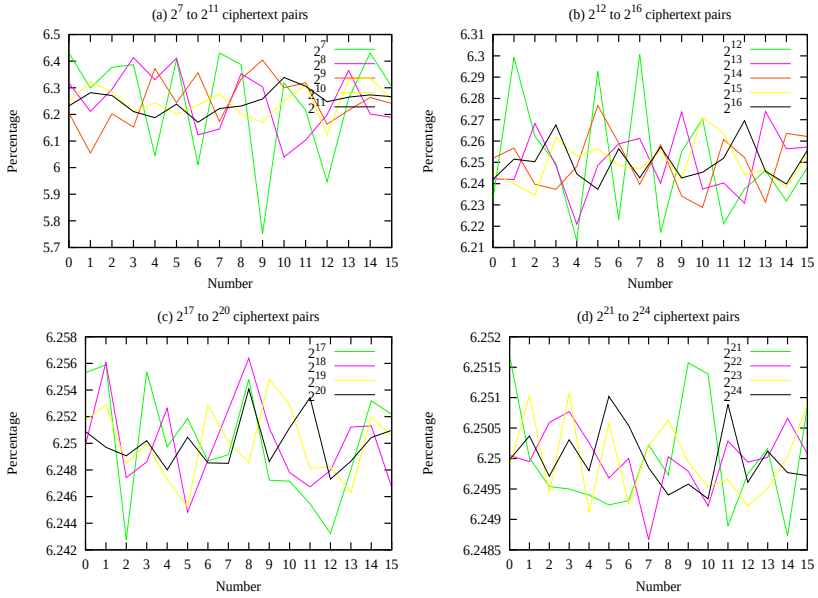

Figure C.59: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 8-round RC5 for 10 trials

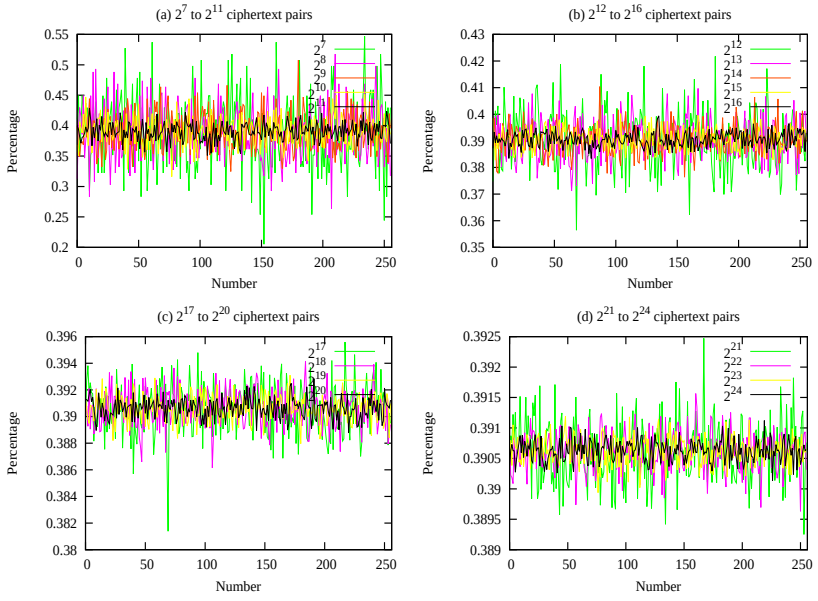

Figure C.60: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 8-round RC5 for 10 trials

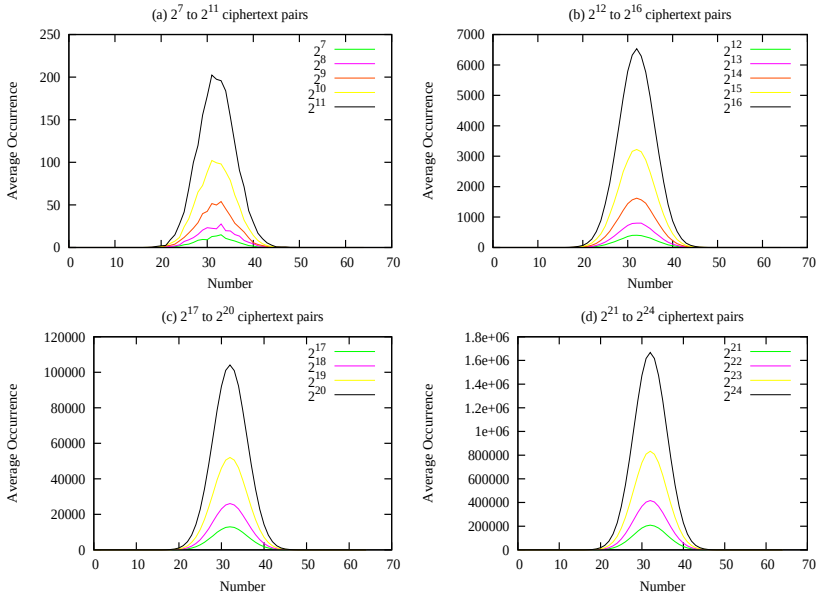

Figure C.61: Average Hamming Weight Distribution between ciphertext pairs for 8-round RC5 for 10 trials (Autofeeding)

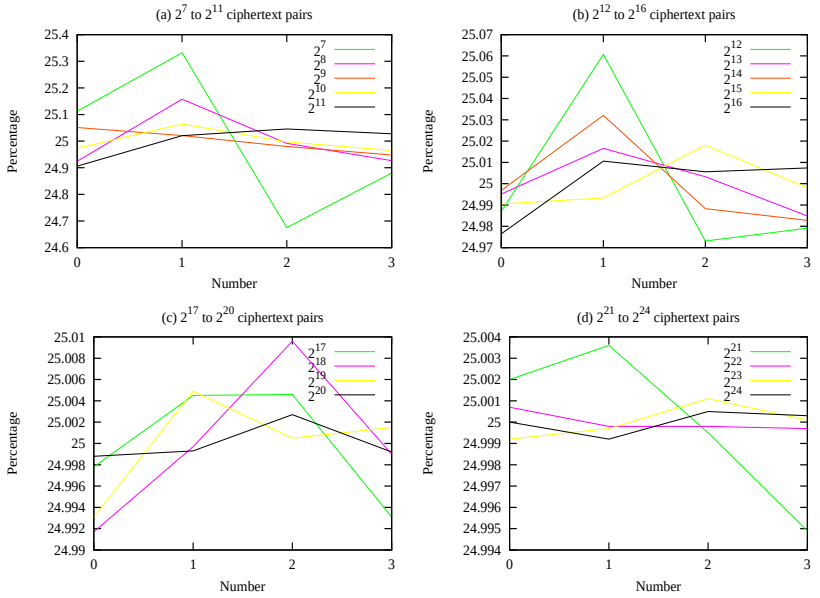

Figure C.62: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 8-round RC5 for 10 trials (Autofeeding)

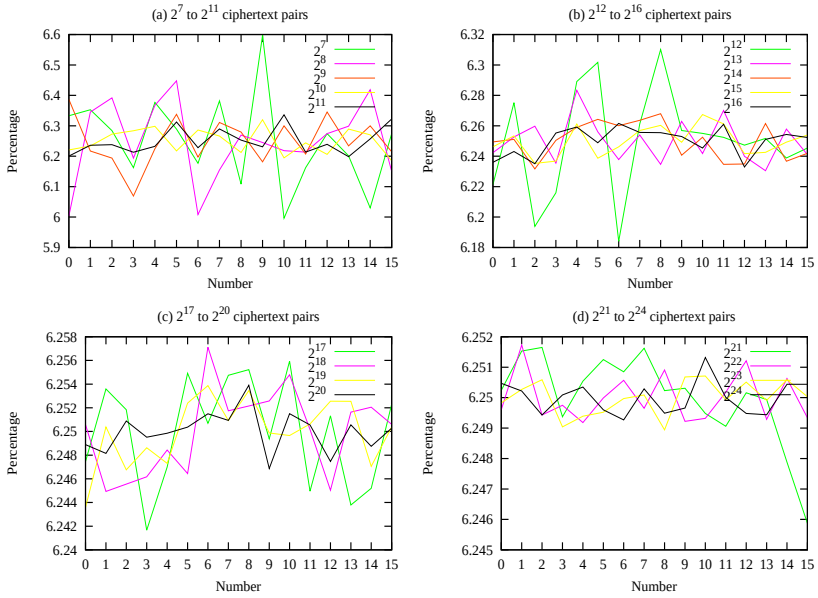

Figure C.63: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 8-round RC5 for 10 trials (Autofeeding)

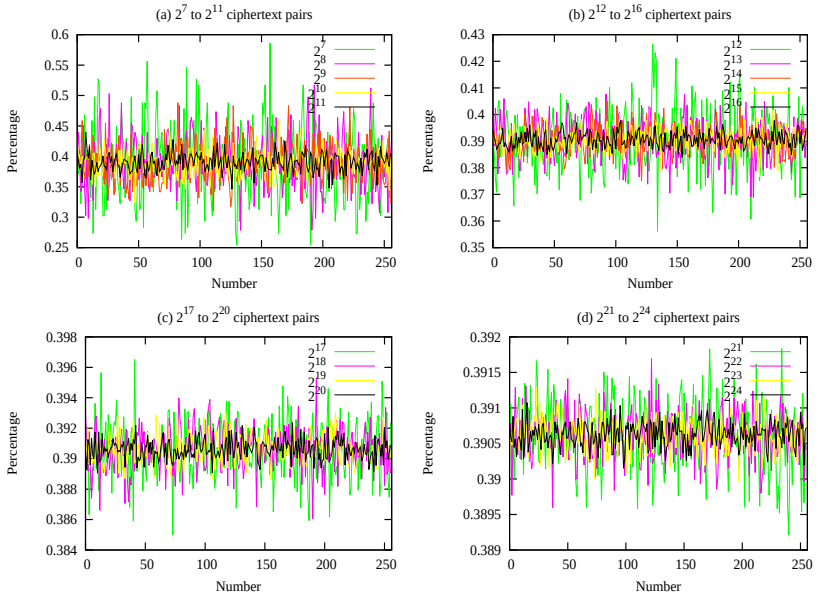

Figure C.64: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 8-round RC5 for 10 trials (Autofeeding)

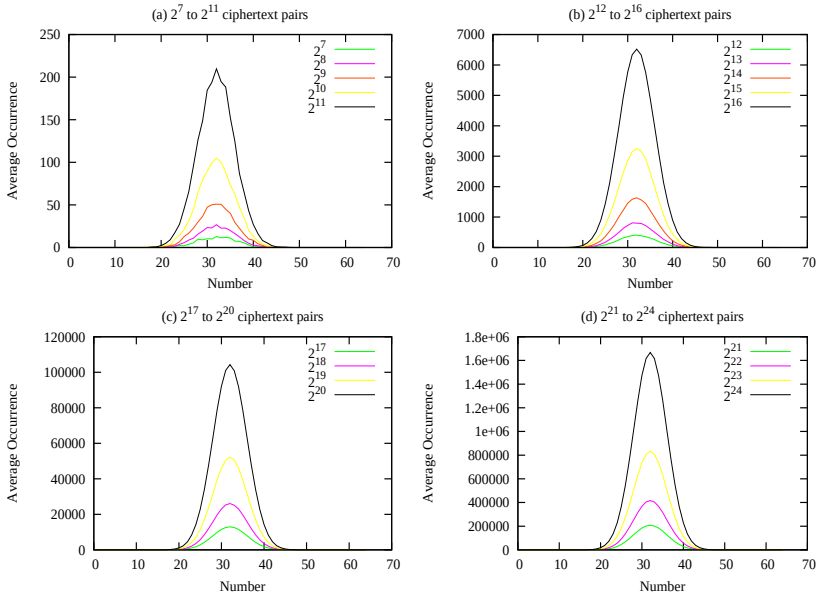

Figure C.65: Average Hamming Weight Distribution between ciphertext pairs for 9-round RC5 for 10 trials

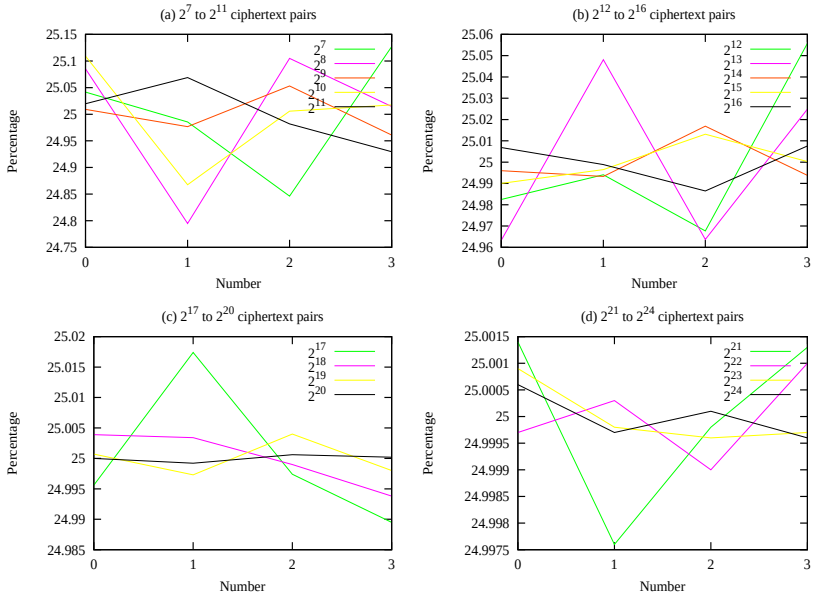

Figure C.66: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 9-round RC5 for 10 trials

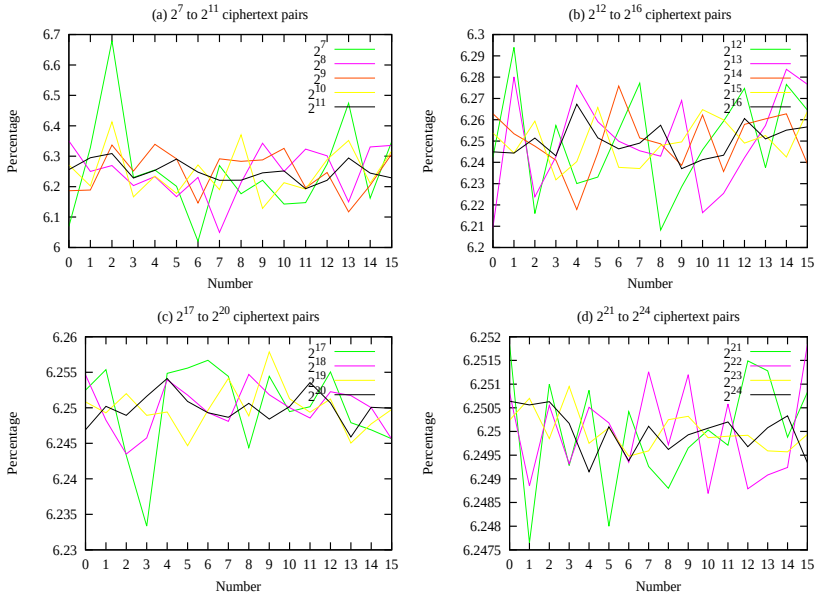

Figure C.67: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 9-round RC5 for 10 trials

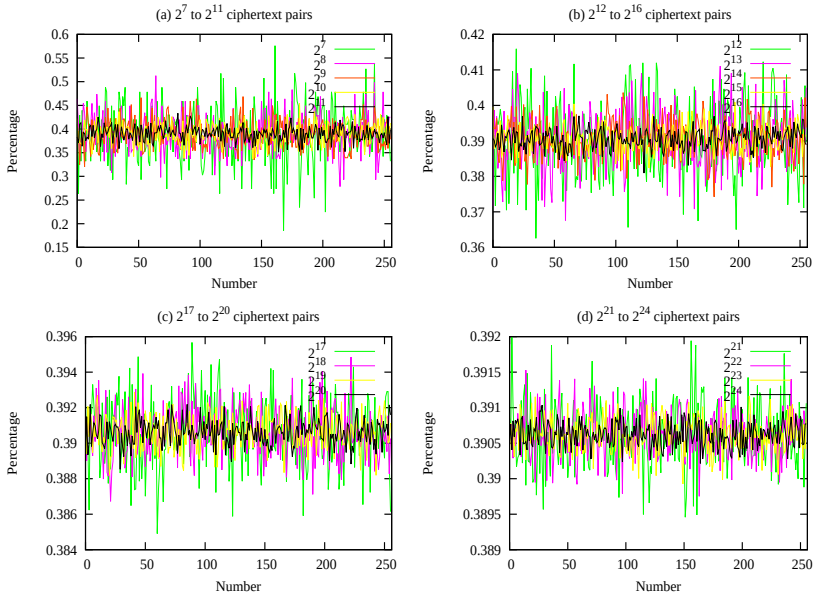

Figure C.68: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 9-round RC5 for 10 trials

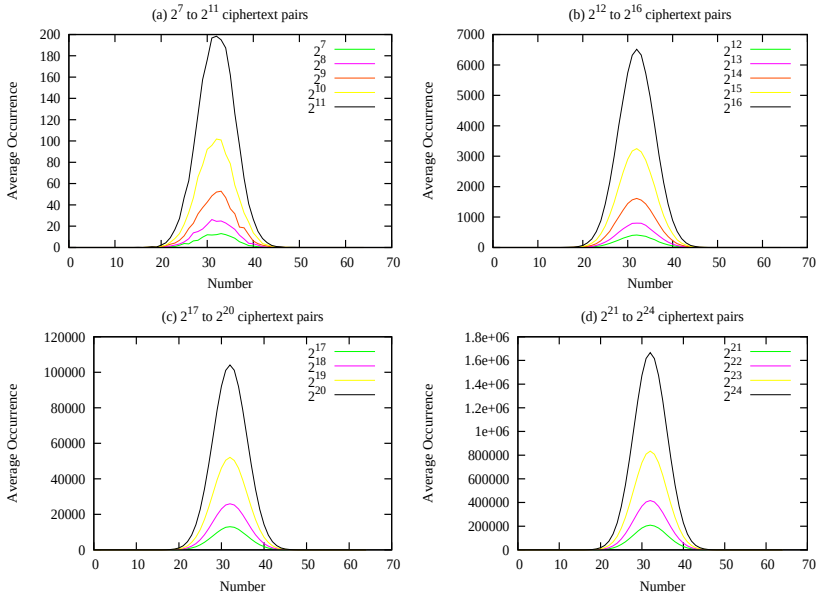

Figure C.69: Average Hamming Weight Distribution between ciphertext pairs for 9-round RC5 for 10 trials (Autofeeding)

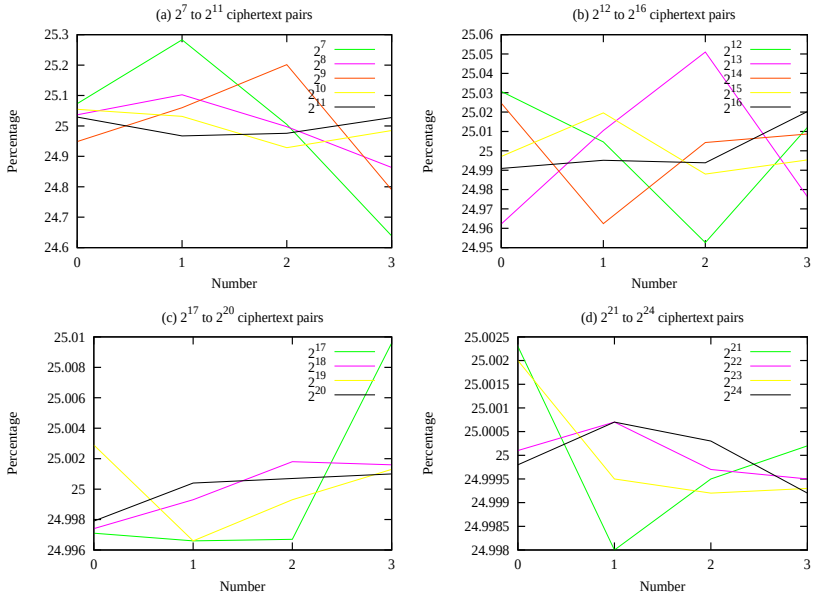

Figure C.70: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 9-round RC5 for 10 trials (Autofeeding)

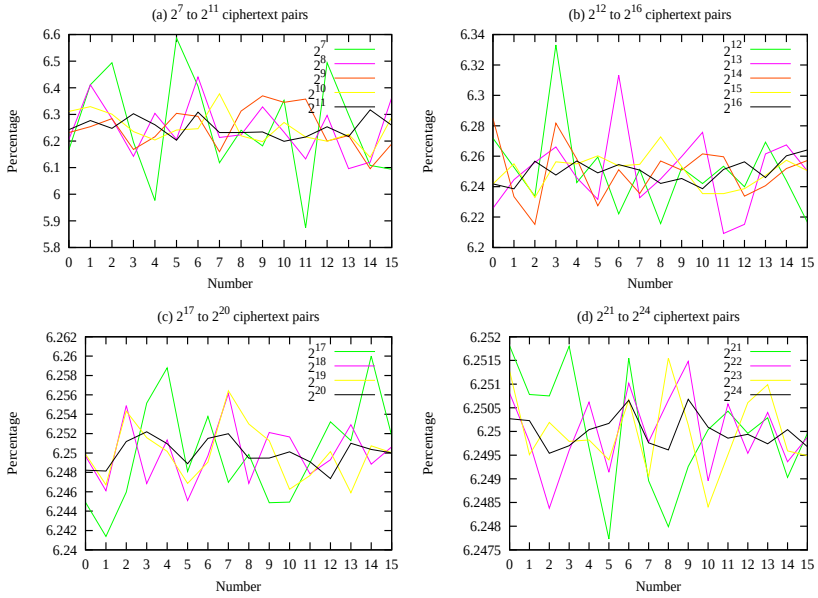

Figure C.71: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 9-round RC5 for 10 trials (Autofeeding)

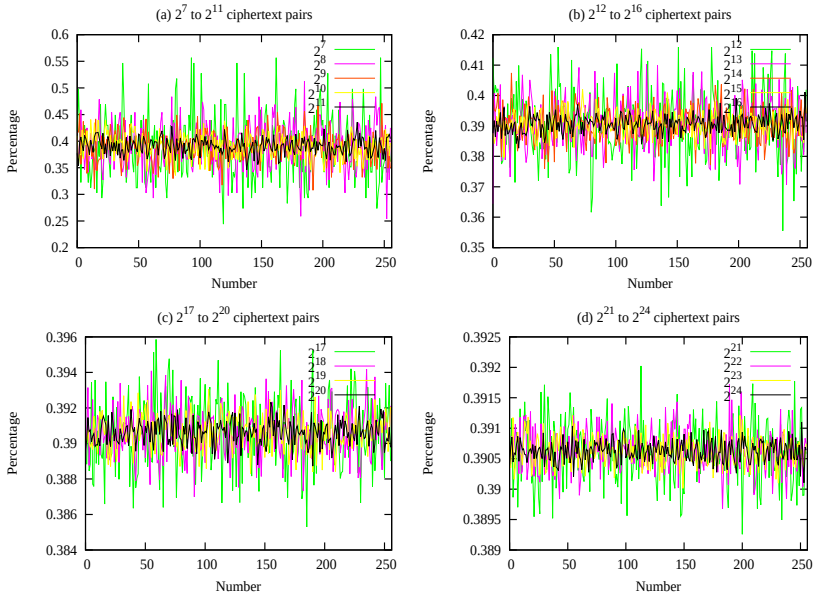

Figure C.72: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 9-round RC5 for 10 trials (Autofeeding)

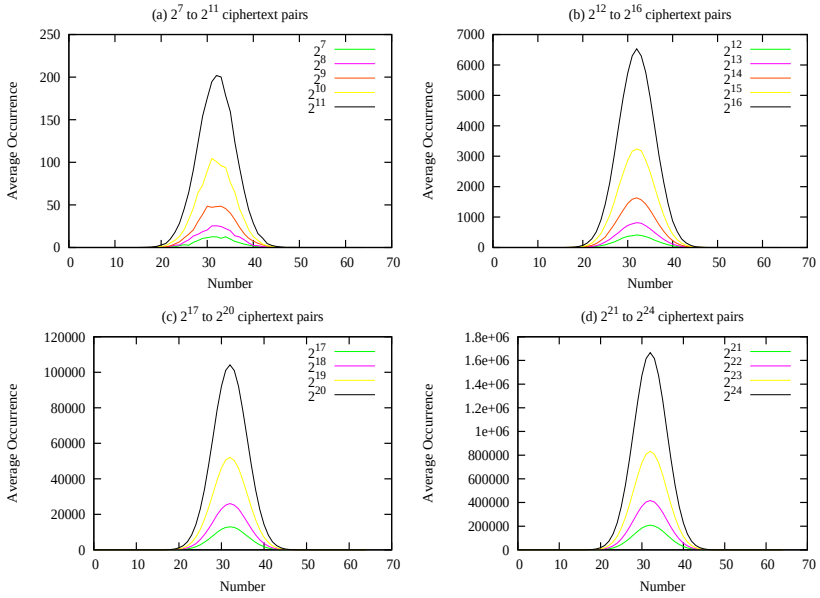

Figure C.73: Average Hamming Weight Distribution between ciphertext pairs for 10-round RC5 for 10 trials

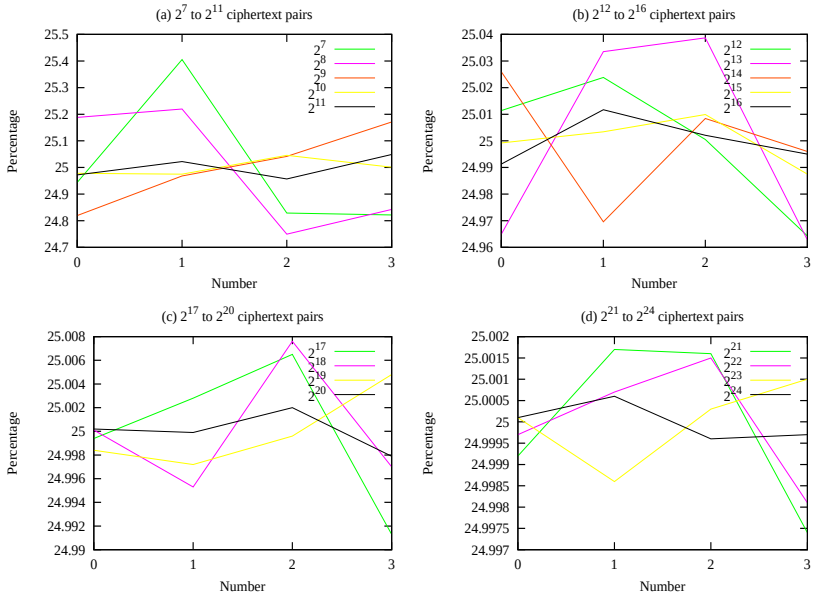

Figure C.74: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 10-round RC5 for 10 trials

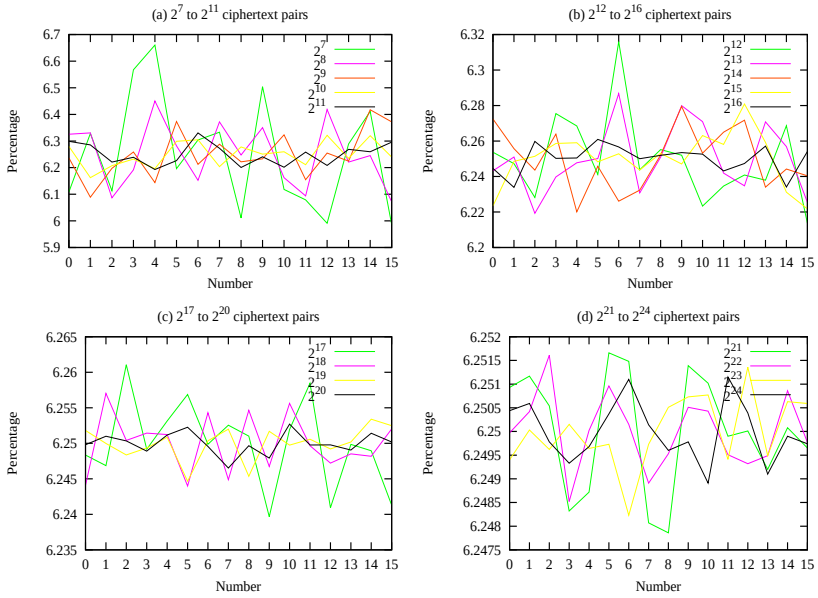

Figure C.75: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 10-round RC5 for 10 trials

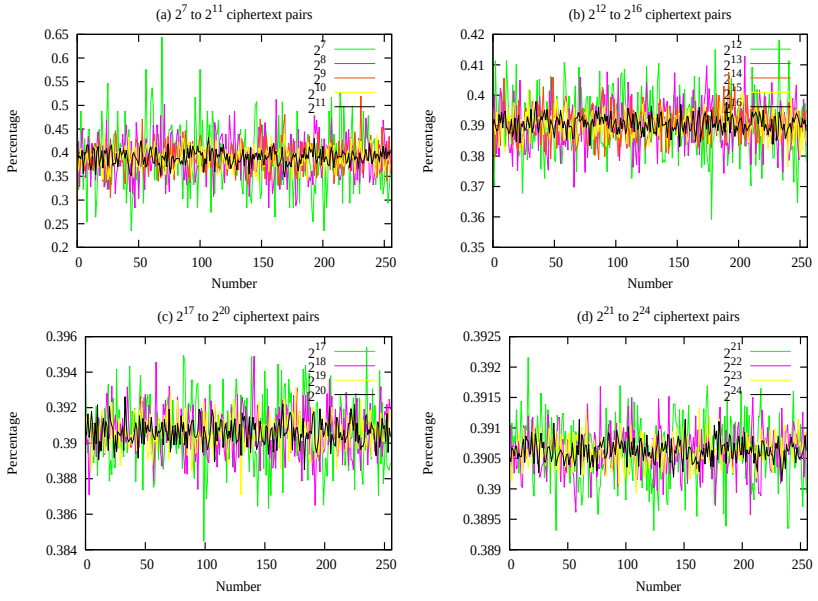

Figure C.76: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 10-round RC5 for 10 trials

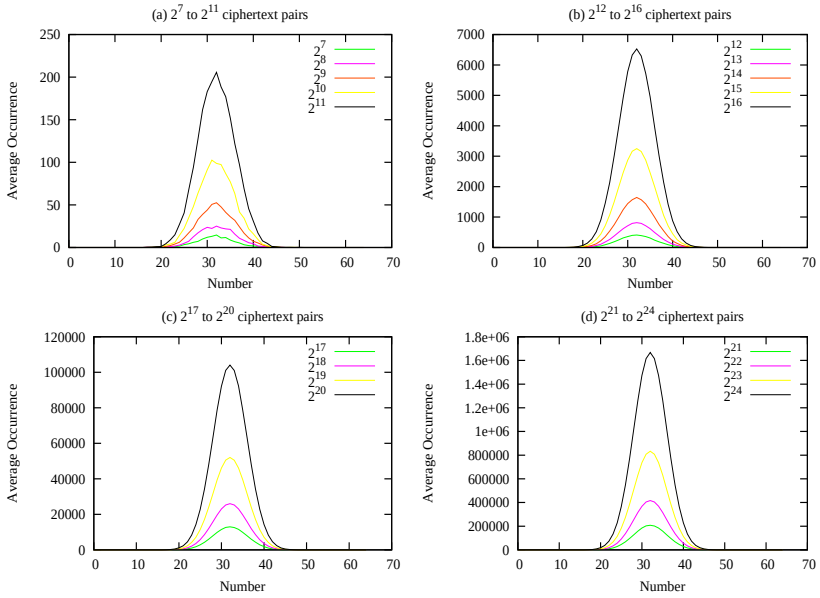

Figure C.77: Average Hamming Weight Distribution between ciphertext pairs for 10-round RC5 for 10 trials (Autofeeding)

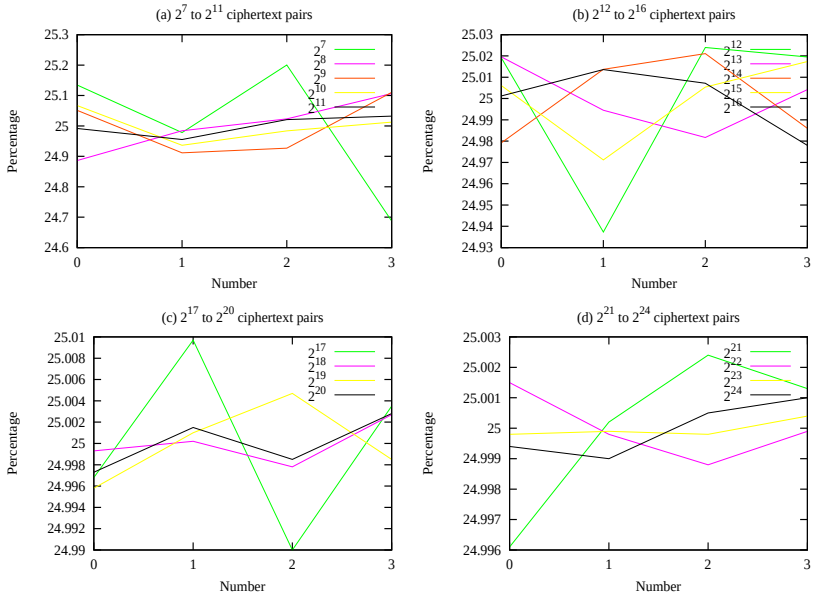

Figure C.78: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 10-round RC5 for 10 trials (Autofeeding)

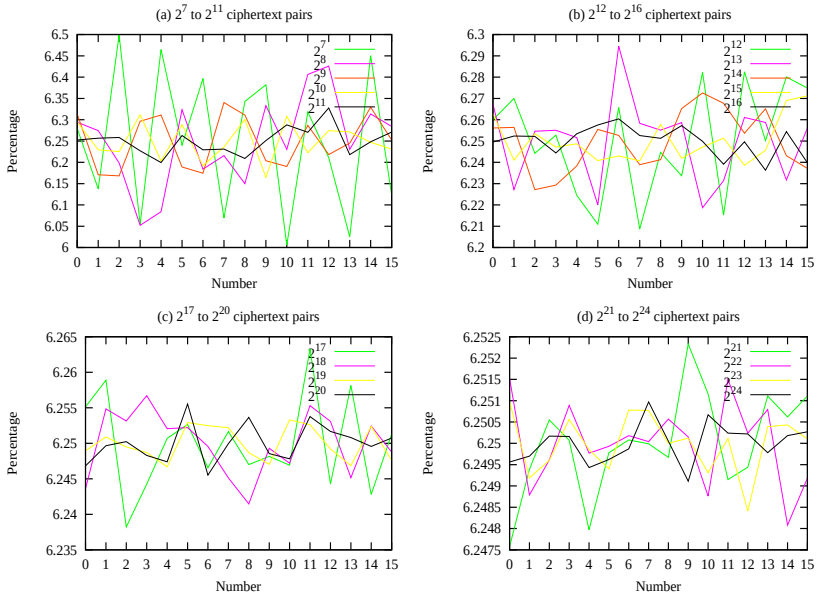

Figure C.79: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 10-round RC5 for 10 trials (Autofeeding)

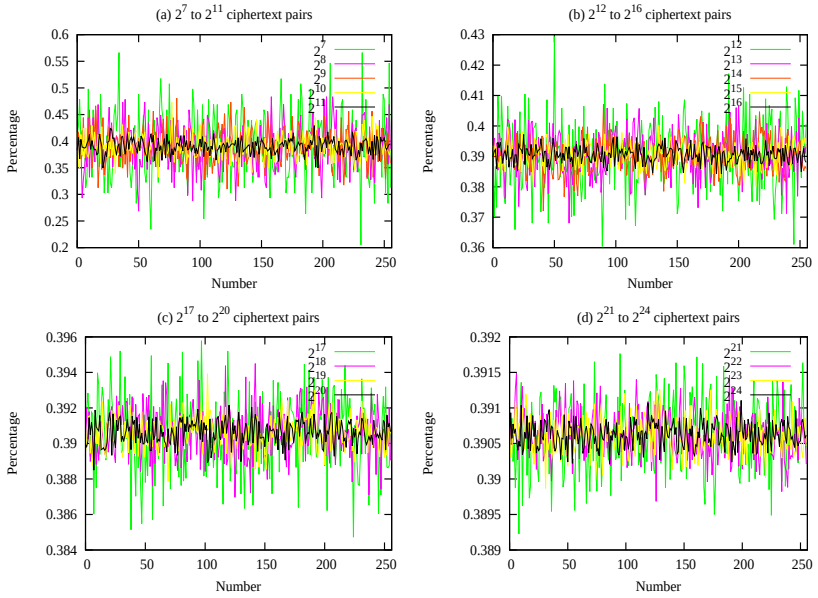

Figure C.80: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 10-round RC5 for 10 trials (Autofeeding)

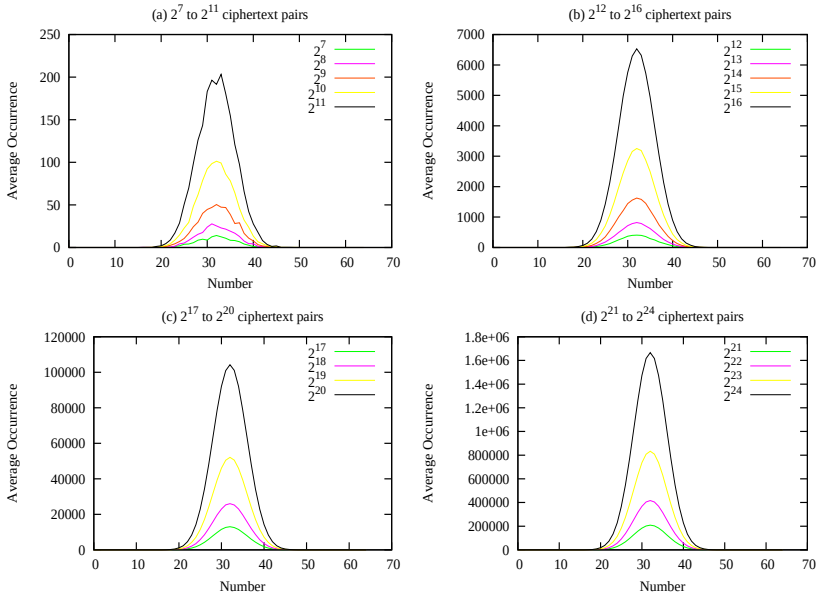

Figure C.81: Average Hamming Weight Distribution between ciphertext pairs for 11-round RC5 for 10 trials

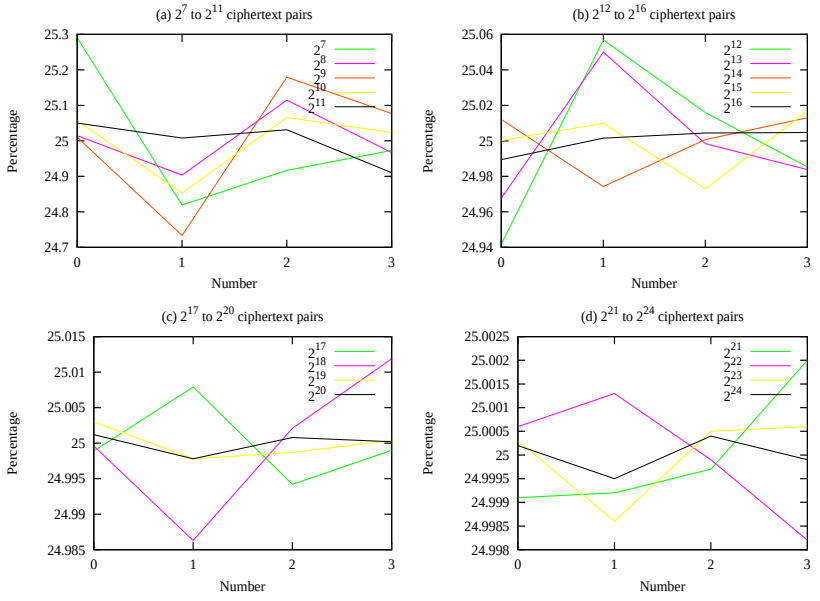

Figure C.82: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 11-round RC5 for 10 trials

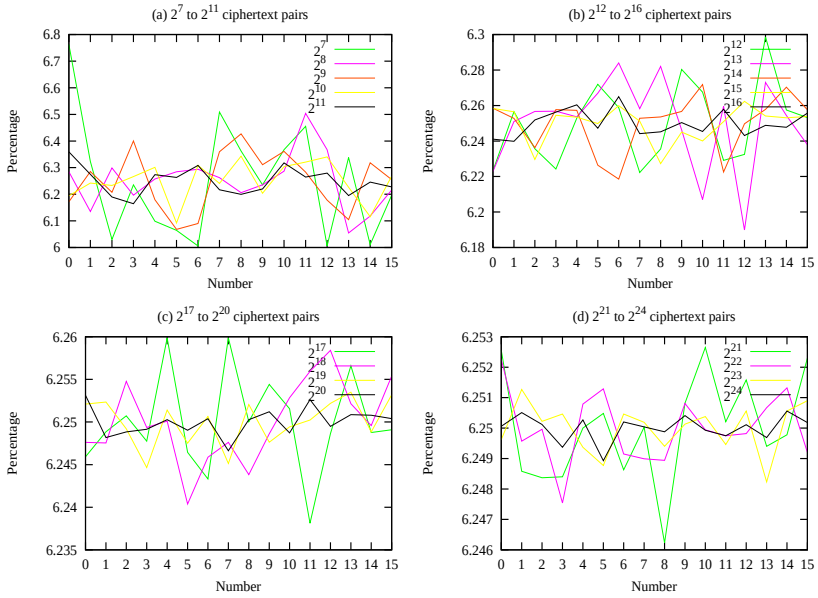

Figure C.83: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 11-round RC5 for 10 trials

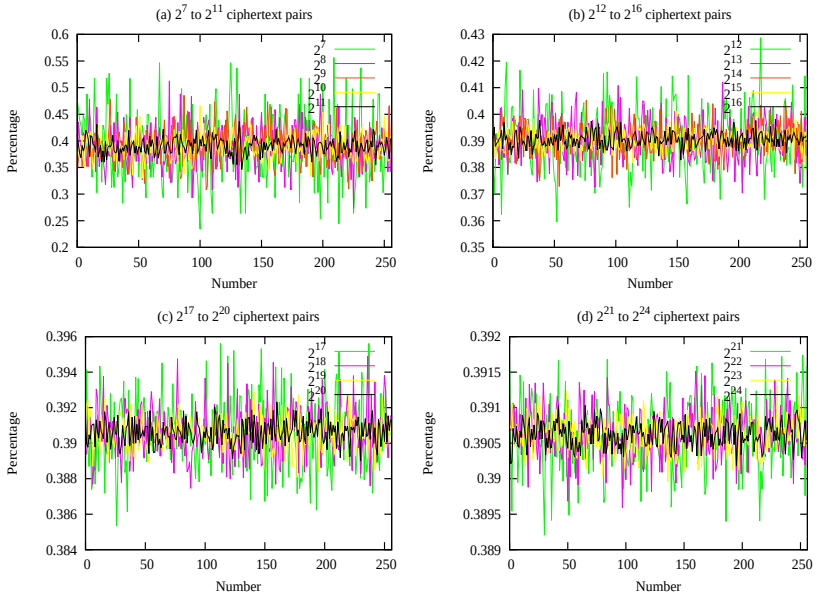

Figure C.84: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 11-round RC5 for 10 trials

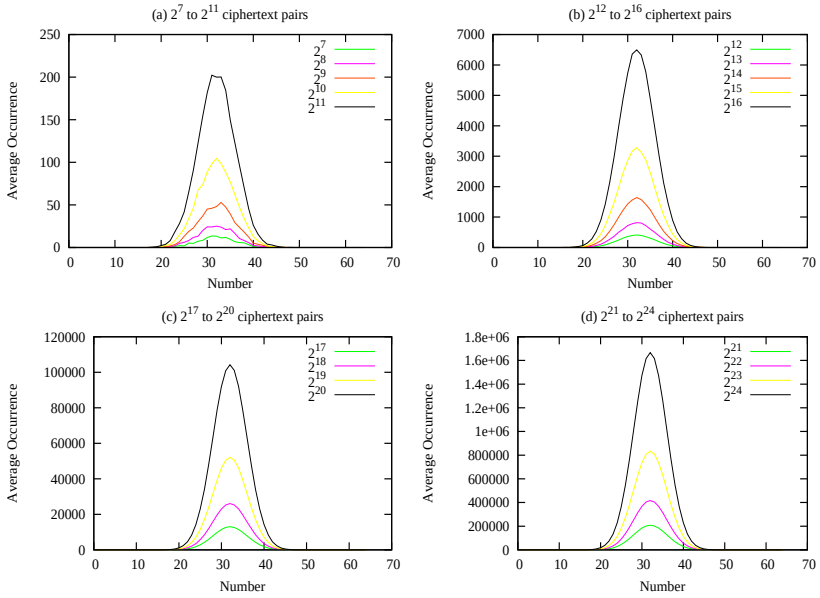

Figure C.85: Average Hamming Weight Distribution between ciphertext pairs for 11-round RC5 for 10 trials (Autofeeding)

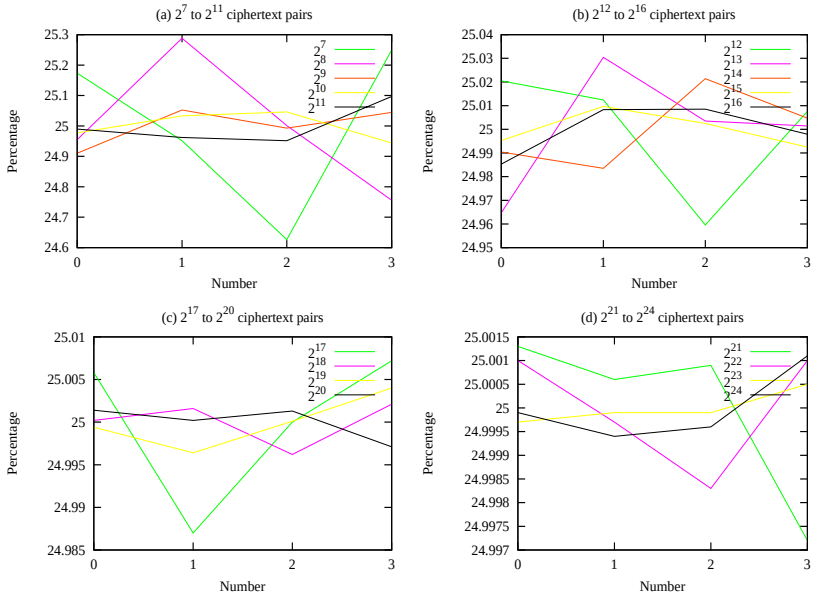

Figure C.86: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 11-round RC5 for 10 trials (Autofeeding)

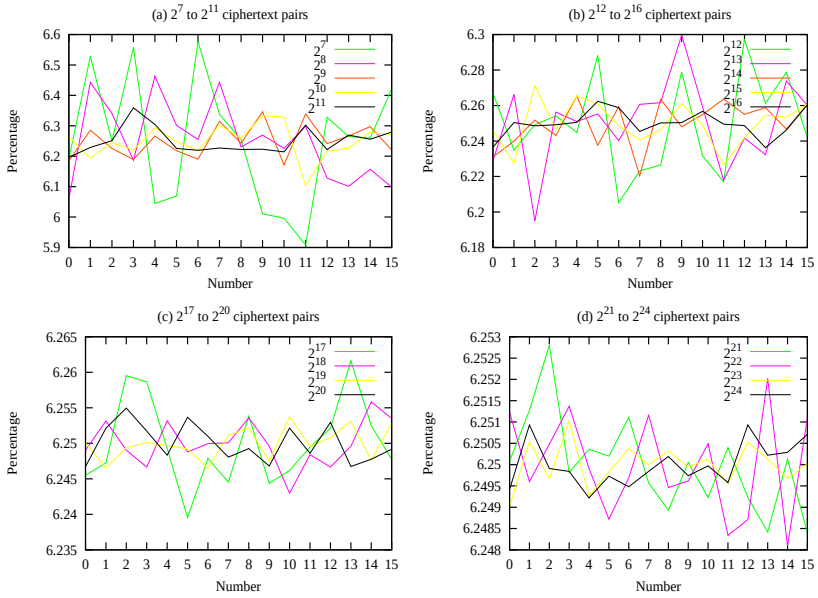

Figure C.87: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 11-round RC5 for 10 trials (Autofeeding)

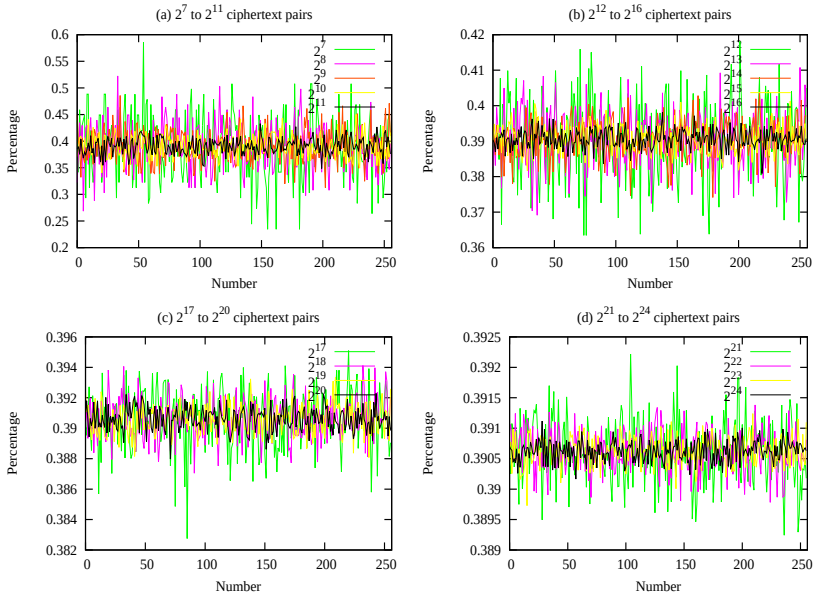

Figure C.88: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 11-round RC5 for 10 trials (Autofeeding)

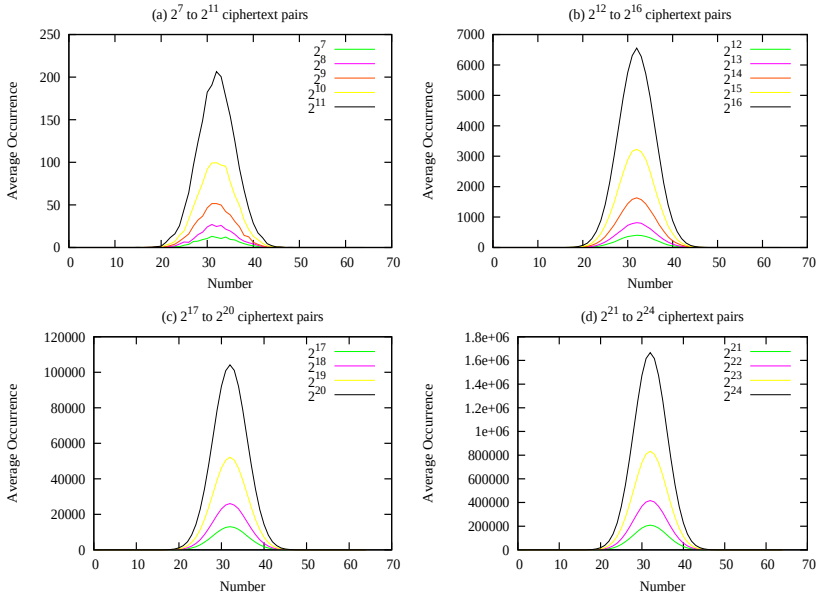

Figure C.89: Average Hamming Weight Distribution between ciphertext pairs for 12-round RC5 for 10 trials

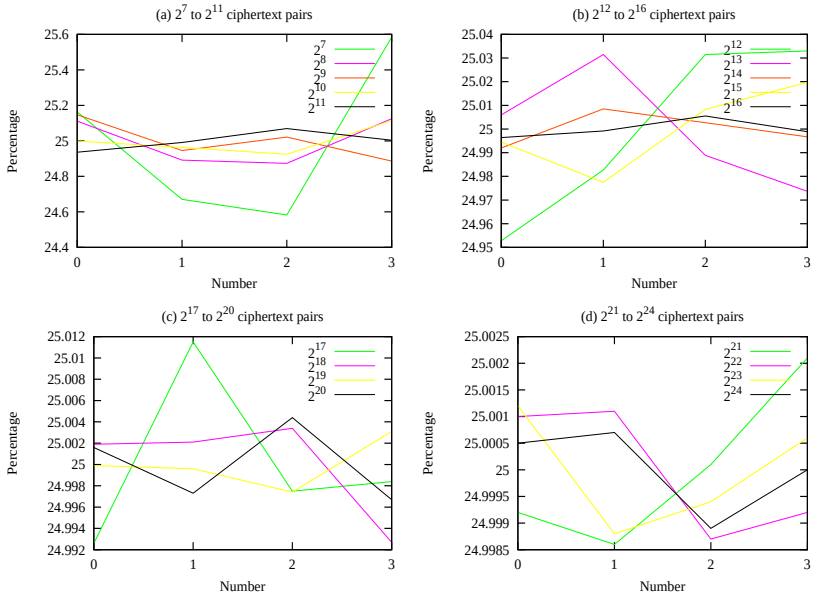

Figure C.90: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 12-round RC5 for 10 trials

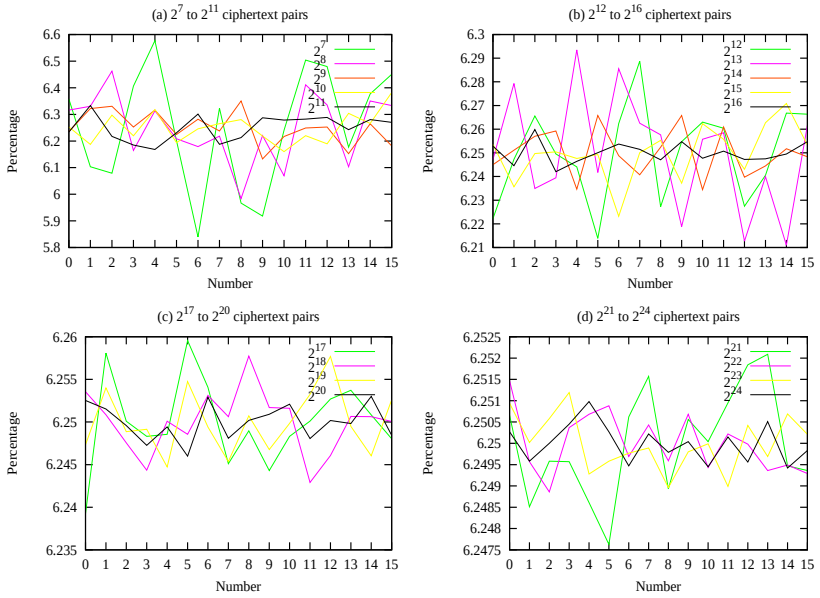

Figure C.91: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 12-round RC5 for 10 trials

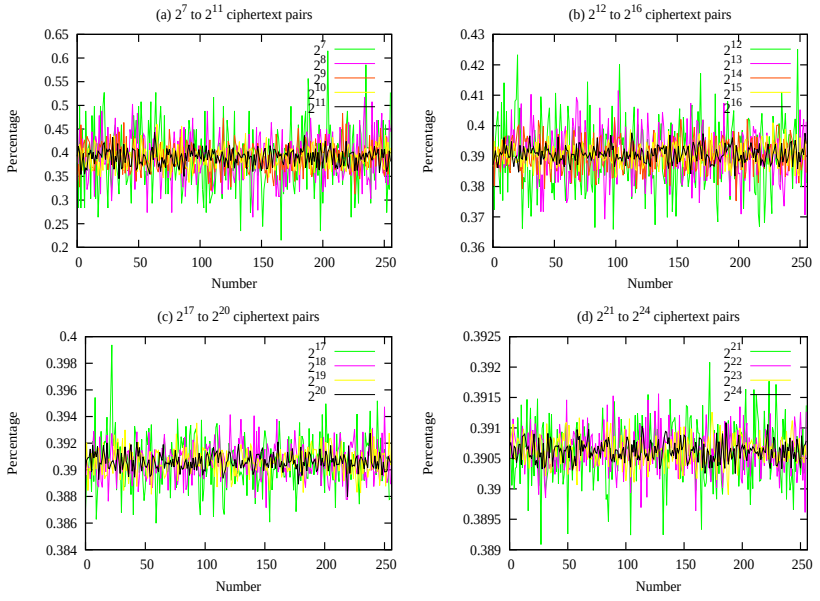

Figure C.92: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 12-round RC5 for 10 trials

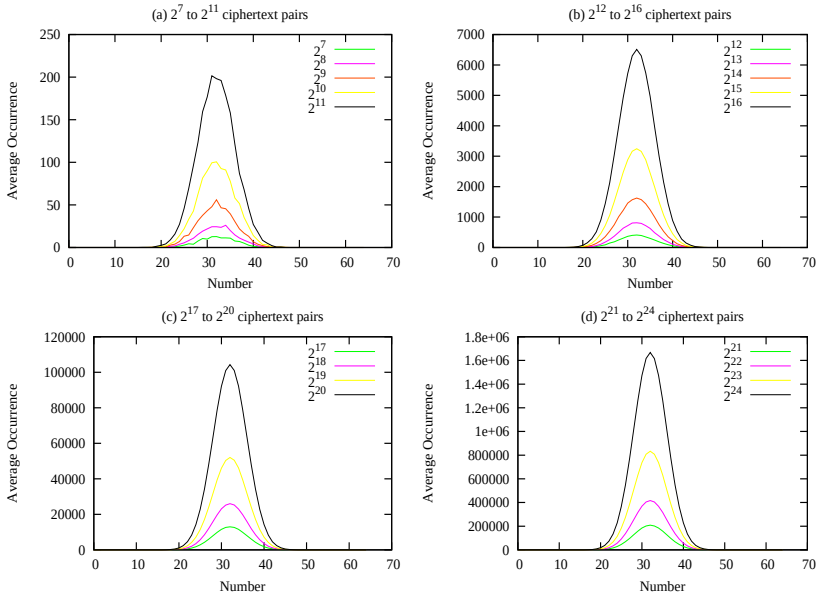

Figure C.93: Average Hamming Weight Distribution between ciphertext pairs for 12-round RC5 for 10 trials (Autofeeding)

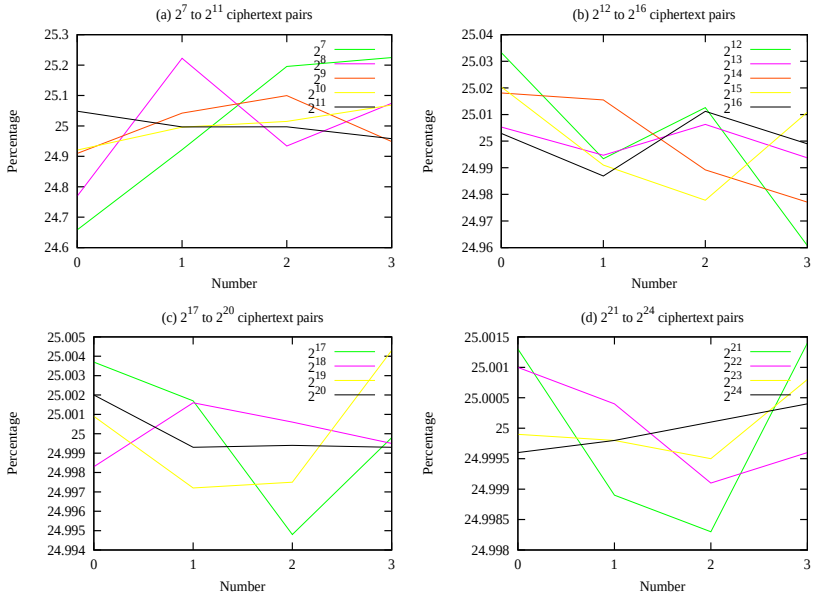

Figure C.94: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 12-round RC5 for 10 trials (Autofeeding)

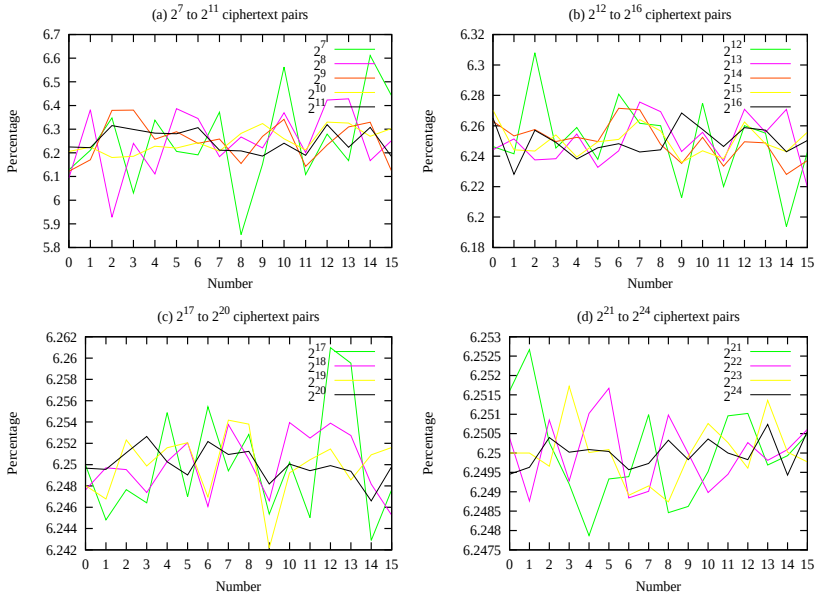

Figure C.95: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 12-round RC5 for 10 trials (Autofeeding)

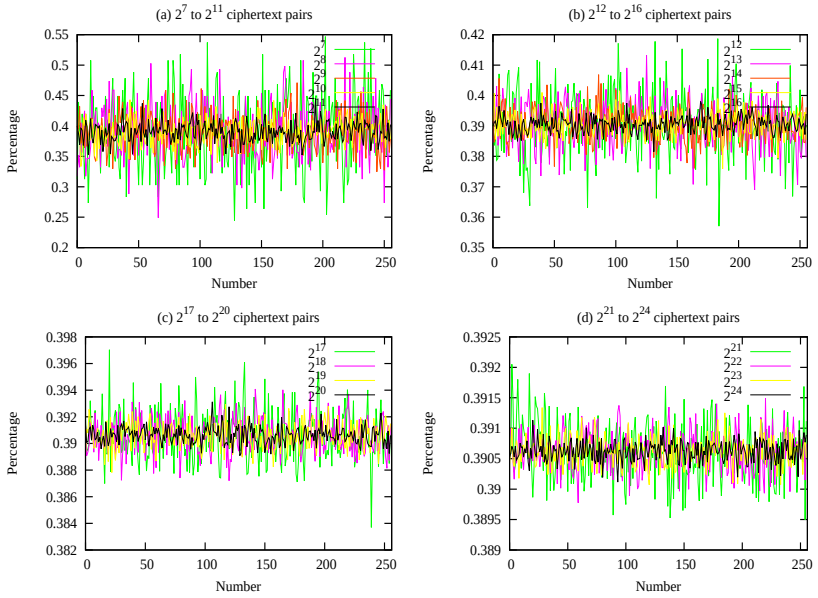

Figure C.96: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 12-round RC5 for 10 trials (Autofeeding)

D Experimental Test Results for AES-128

Table D.1: Experimental Results of SAC Tests on Various Rounds of AES-128

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	127	0.005	171.796	232.271	256($i.e.$ 2^8)	Fail	Fig.D.1
2	127	0.005	171.796	232.271	256($i.e.$ 2^8)	Fail	Fig.D.9
3	127	0.005	171.796	191.456	65536($i.e.$ 2^{16})	Fail	Fig.D.17
4	127	0.005	171.796	61.7608	16777216($i.e.$ 2^{24})	Pass	Fig.D.25
5	127	0.005	171.796	57.6577	16777216($i.e.$ 2^{24})	Pass	Fig.D.33
6	127	0.005	171.796	56.7642	16777216($i.e.$ 2^{24})	Pass	Fig.D.41
7	127	0.005	171.796	51.9667	16777216($i.e.$ 2^{24})	Pass	Fig.D.49
8	127	0.005	171.796	44.7875	16777216($i.e.$ 2^{24})	Pass	Fig.D.57
9	127	0.005	171.796	54.6889	16777216($i.e.$ 2^{24})	Pass	Fig.D.65
10	127	0.005	171.796	52.5992	16777216($i.e.$ 2^{24})	Pass	Fig.D.73

Table D.2: Experimental Results of modSAC Tests for 2-bit substring on Various Rounds of AES-128

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	3	0.005	12.838	21494.5	128($i.e.$ 2^7)	Fail	Fig.D.2
2	3	0.005	12.838	13795.3	128($i.e.$ 2^7)	Fail	Fig.D.10
3	3	0.005	12.838	14.5775	4096($i.e.$ 2^{12})	Fail	Fig.D.18
4	3	0.005	12.838	3.95398	16777216($i.e.$ 2^{24})	Pass	Fig.D.26
5	3	0.005	12.838	3.48624	16777216($i.e.$ 2^{24})	Pass	Fig.D.34
6	3	0.005	12.838	2.8363	16777216($i.e.$ 2^{24})	Pass	Fig.D.42
7	3	0.005	12.838	2.93421	16777216($i.e.$ 2^{24})	Pass	Fig.D.50
8	3	0.005	12.838	2.49624	16777216($i.e.$ 2^{24})	Pass	Fig.D.58
9	3	0.005	12.838	2.92834	16777216($i.e.$ 2^{24})	Pass	Fig.D.66
10	3	0.005	12.838	1.93027	16777216($i.e.$ 2^{24})	Pass	Fig.D.74

Table D.3: Experimental Results of modSAC Tests for 4-bit substring on Various Rounds of AES-128

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	15	0.005	32.801	53878.3	128($i.e.$ 2^7)	Fail	Fig.D.3
2	15	0.005	32.801	34475.9	128($i.e.$ 2^7)	Fail	Fig.D.11
3	15	0.005	32.801	42.7754	4096($i.e.$ 2^{12})	Fail	Fig.D.19
4	15	0.005	32.801	15.9943	16777216($i.e.$ 2^{24})	Pass	Fig.D.27
5	15	0.005	32.801	16.8487	16777216($i.e.$ 2^{24})	Pass	Fig.D.35
6	15	0.005	32.801	14.9346	16777216($i.e.$ 2^{24})	Pass	Fig.D.43
7	15	0.005	32.801	15.4279	16777216($i.e.$ 2^{24})	Pass	Fig.D.51
8	15	0.005	32.801	14.5539	16777216($i.e.$ 2^{24})	Pass	Fig.D.59
9	15	0.005	32.801	15.8909	16777216($i.e.$ 2^{24})	Pass	Fig.D.67
10	15	0.005	32.801	14.2374	16777216($i.e.$ 2^{24})	Pass	Fig.D.75

Table D.4: Experimental Results of modSAC Tests for 8-bit substring on Various Rounds of AES-128

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	255	0.005	316.919	458778	128($i.e.$ 2^7)	Fail	Fig.D.4
2	255	0.005	316.919	293060	128($i.e.$ 2^7)	Fail	Fig.D.12
3	255	0.005	316.919	333.528	1024($i.e.$ 2^{10})	Fail	Fig.D.20
4	255	0.005	316.919	259.646	16777216($i.e.$ 2^{24})	Pass	Fig.D.28
5	255	0.005	316.919	257.642	16777216($i.e.$ 2^{24})	Pass	Fig.D.36
6	255	0.005	316.919	258.327	16777216($i.e.$ 2^{24})	Pass	Fig.D.44
7	255	0.005	316.919	269.851	16777216($i.e.$ 2^{24})	Pass	Fig.D.52
8	255	0.005	316.919	257.594	16777216($i.e.$ 2^{24})	Pass	Fig.D.60
9	255	0.005	316.919	262.553	16777216($i.e.$ 2^{24})	Pass	Fig.D.68
10	255	0.005	316.919	256.164	16777216($i.e.$ 2^{24})	Pass	Fig.D.76

Table D.5: Experimental Results of SAC Tests on Various Rounds of AES-128 (Autofeeding)

No. of Rounds	d.o.f	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	127	0.005	171.796	232.271	256($i.e.$ 2^8)	Fail	Fig.D.5
2	127	0.005	171.796	232.271	256($i.e.$ 2^8)	Fail	Fig.D.13
3	127	0.005	171.796	174.762	65536($i.e.$ 2^{16})	Fail	Fig.D.21
4	127	0.005	171.796	54.6873	16777216($i.e.$ 2^{24})	Pass	Fig.D.29
5	127	0.005	171.796	50.8516	16777216($i.e.$ 2^{24})	Pass	Fig.D.37
6	127	0.005	171.796	60.4335	16777216($i.e.$ 2^{24})	Pass	Fig.D.45
7	127	0.005	171.796	52.7733	16777216($i.e.$ 2^{24})	Pass	Fig.D.53
8	127	0.005	171.796	55.4094	16777216($i.e.$ 2^{24})	Pass	Fig.D.61
9	127	0.005	171.796	47.6626	16777216($i.e.$ 2^{24})	Pass	Fig.D.69
10	127	0.005	171.796	64.2755	16777216($i.e.$ 2^{24})	Pass	Fig.D.77

Table D.6: Experimental Results of modSAC Tests for 2-bit substring on Various Rounds of AES-128 (Autofeeding)

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	3	0.005	12.838	21520.8	128($i.e.$ 2^7)	Fail	Fig.D.6
2	3	0.005	12.838	13815.5	128($i.e.$ 2^7)	Fail	Fig.D.14
3	3	0.005	12.838	17.3209	4096($i.e.$ 2^{12})	Fail	Fig.D.22
4	3	0.005	12.838	3.41287	16777216($i.e.$ 2^{24})	Pass	Fig.D.30
5	3	0.005	12.838	2.88539	16777216($i.e.$ 2^{24})	Pass	Fig.D.38
6	3	0.005	12.838	3.16109	16777216($i.e.$ 2^{24})	Pass	Fig.D.46
7	3	0.005	12.838	3.99462	16777216($i.e.$ 2^{24})	Pass	Fig.D.54
8	3	0.005	12.838	1.82648	16777216($i.e.$ 2^{24})	Pass	Fig.D.62
9	3	0.005	12.838	1.97849	16777216($i.e.$ 2^{24})	Pass	Fig.D.70
10	3	0.005	12.838	2.46563	16777216($i.e.$ 2^{24})	Pass	Fig.D.78

Table D.7: Experimental Results of modSAC Tests for 4-bit substring on Various Rounds of AES-128 (Autofeeding)

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	15	0.005	32.801	53896.1	128($i.e.$ 2^7)	Fail	Fig.D.7
2	15	0.005	32.801	34492.2	128($i.e.$ 2^7)	Fail	Fig.D.15
3	15	0.005	32.801	43.5414	4096($i.e.$ 2^{12})	Fail	Fig.D.23
4	15	0.005	32.801	14.9178	16777216($i.e.$ 2^{24})	Pass	Fig.D.31
5	15	0.005	32.801	13.4932	16777216($i.e.$ 2^{24})	Pass	Fig.D.39
6	15	0.005	32.801	16.3209	16777216($i.e.$ 2^{24})	Pass	Fig.D.47
7	15	0.005	32.801	16.5109	16777216($i.e.$ 2^{24})	Pass	Fig.D.55
8	15	0.005	32.801	16.2405	16777216($i.e.$ 2^{24})	Pass	Fig.D.63
9	15	0.005	32.801	14.5288	16777216($i.e.$ 2^{24})	Pass	Fig.D.71
10	15	0.005	32.801	16.112	16777216($i.e.$ 2^{24})	Pass	Fig.D.79

Table D.8: Experimental Results of modSAC Tests for 8-bit substring on Various Rounds of AES-128 (Autofeeding)

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	255	0.005	316.919	458777	128($i.e.$ 2^7)	Fail	Fig.D.8
2	255	0.005	316.919	293055	128($i.e.$ 2^7)	Fail	Fig.D.16
3	255	0.005	316.919	317.056	1024($i.e.$ 2^{10})	Fail	Fig.D.24
4	255	0.005	316.919	252.196	16777216($i.e.$ 2^{24})	Pass	Fig.D.32
5	255	0.005	316.919	263.047	16777216($i.e.$ 2^{24})	Pass	Fig.D.40
6	255	0.005	316.919	263.553	16777216($i.e.$ 2^{24})	Pass	Fig.D.48
7	255	0.005	316.919	259.71	16777216($i.e.$ 2^{24})	Pass	Fig.D.56
8	255	0.005	316.919	257.422	16777216($i.e.$ 2^{24})	Pass	Fig.D.64
9	255	0.005	316.919	257.288	16777216($i.e.$ 2^{24})	Pass	Fig.D.72
10	255	0.005	316.919	261.526	16777216($i.e.$ 2^{24})	Pass	Fig.D.80

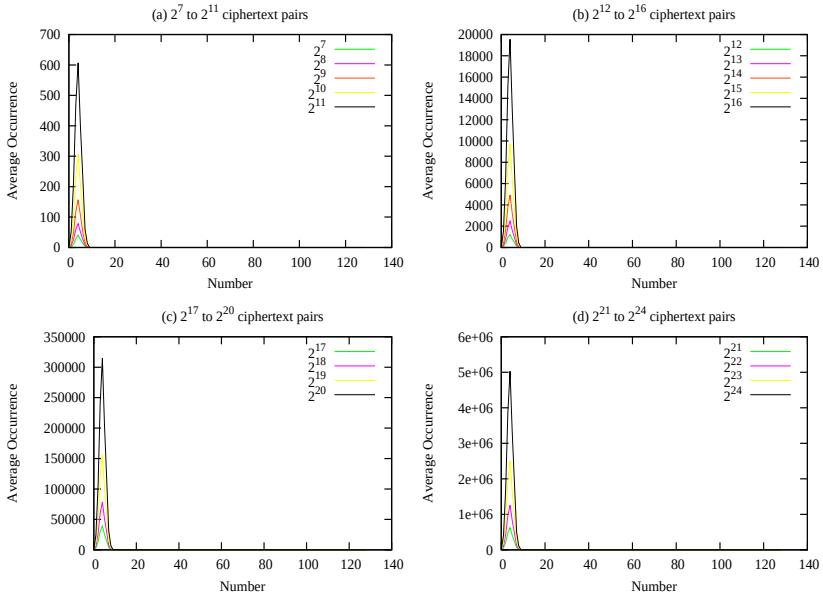

Figure D.1: Average Hamming Weight Distribution between ciphertext pairs for 1-round AES-128 for 10 trials

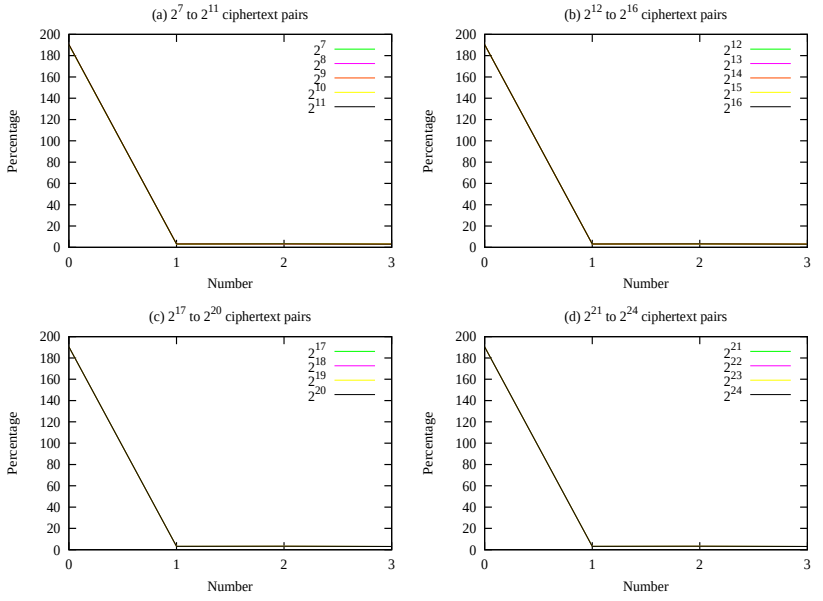

Figure D.2: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 1-round AES-128 for 10 trials

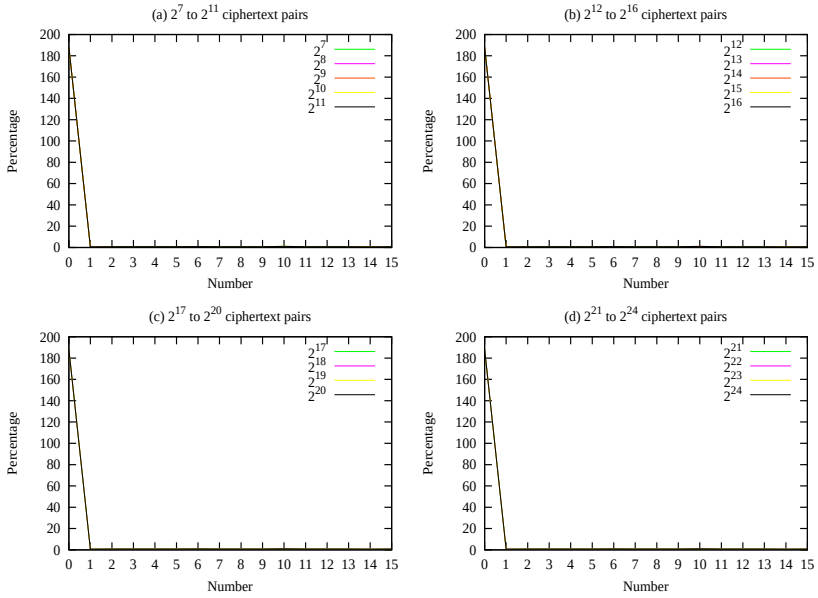

Figure D.3: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 1-round AES-128 for 10 trials

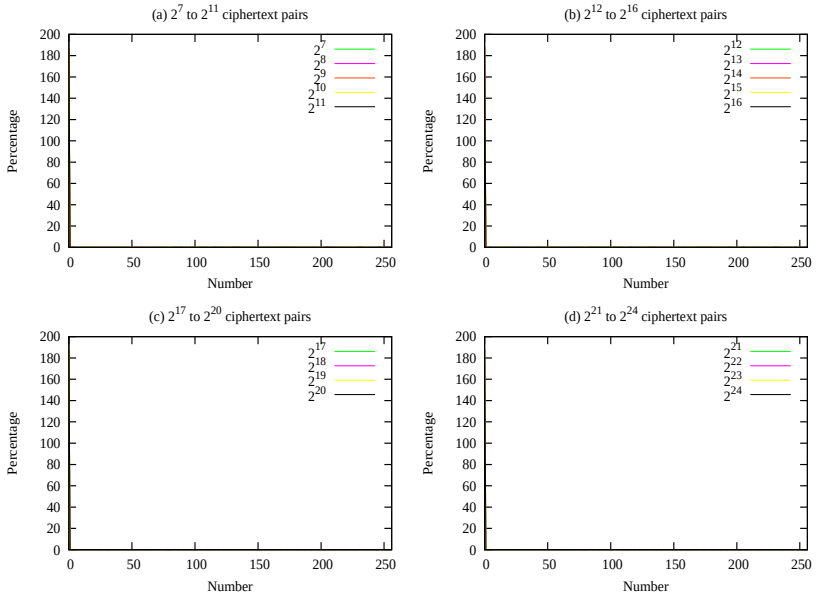

Figure D.4: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 1-round AES-128 for 10 trials

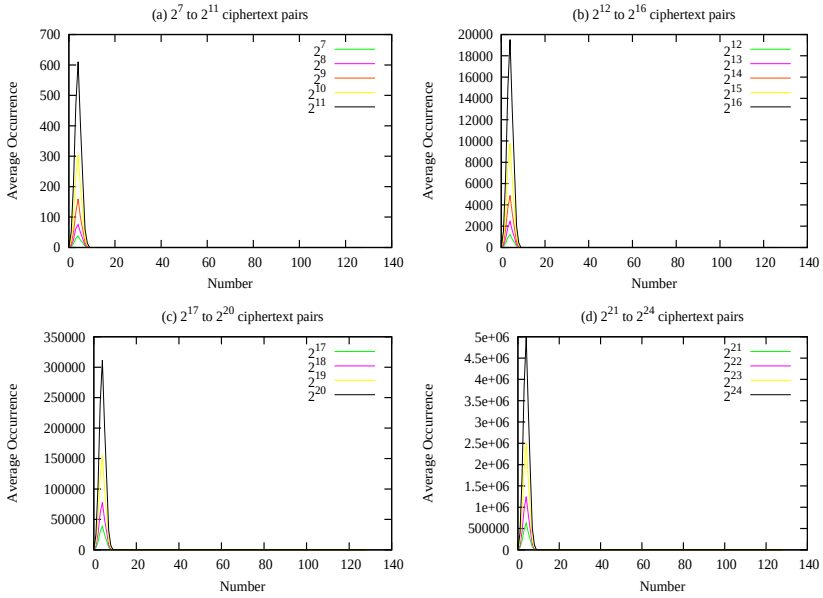

Figure D.5: Average Hamming Weight Distribution between ciphertext pairs for 1-round AES-128 for 10 trials (Autofeeding)

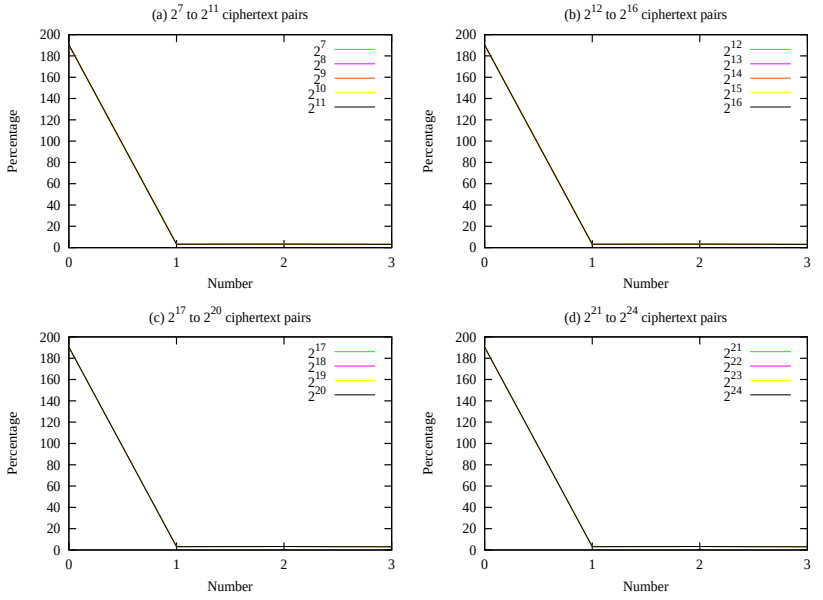

Figure D.6: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 1-round AES-128 for 10 trials (Autofeeding)

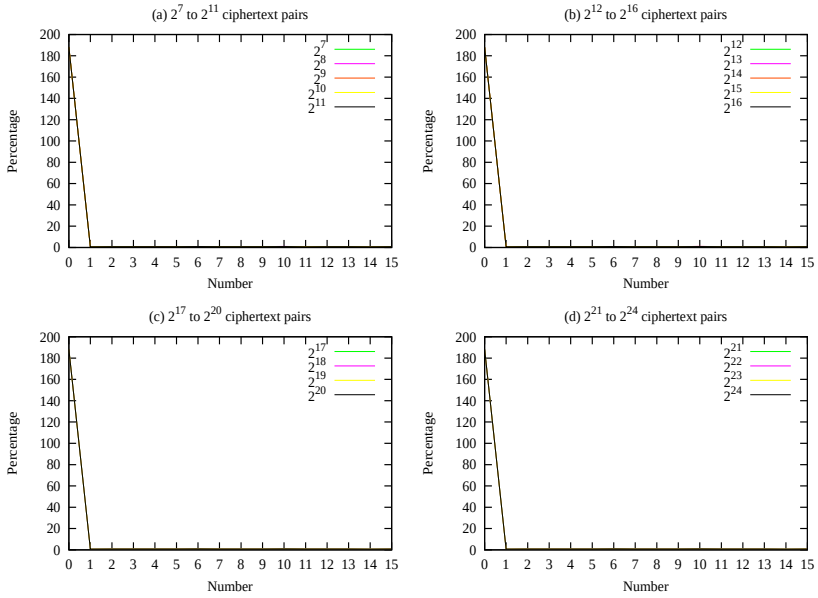

Figure D.7: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 1-round AES-128 for 10 trials (Autofeeding)

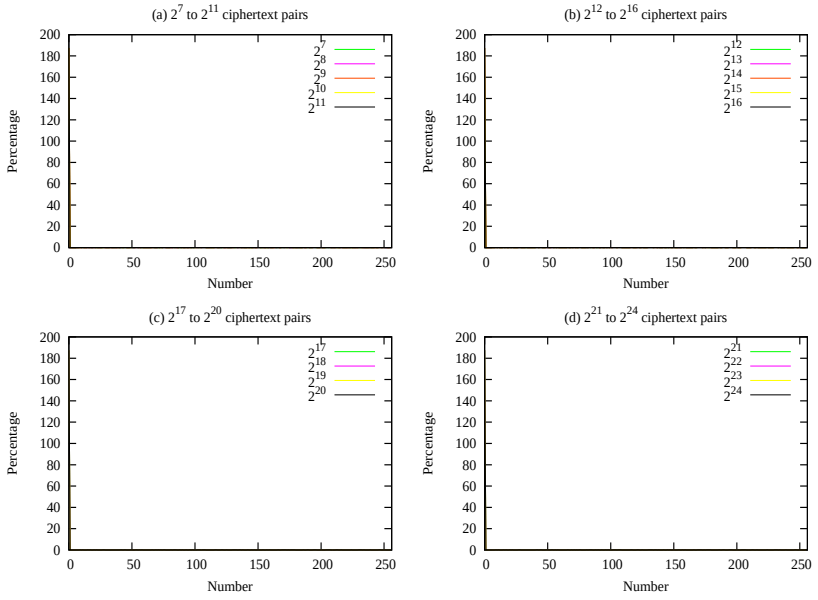

Figure D.8: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 1-round AES-128 for 10 trials (Autofeeding)

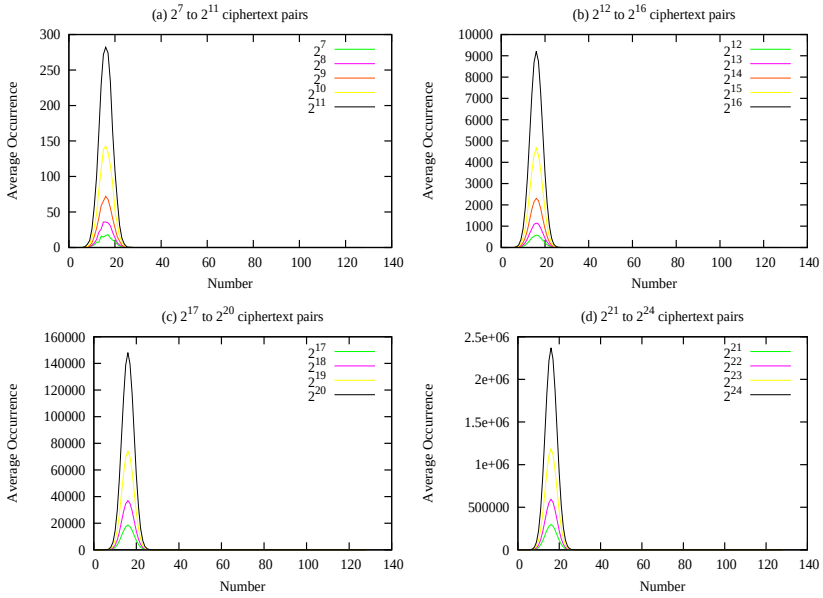

Figure D.9: Average Hamming Weight Distribution between ciphertext pairs for 2-round AES-128 for 10 trials

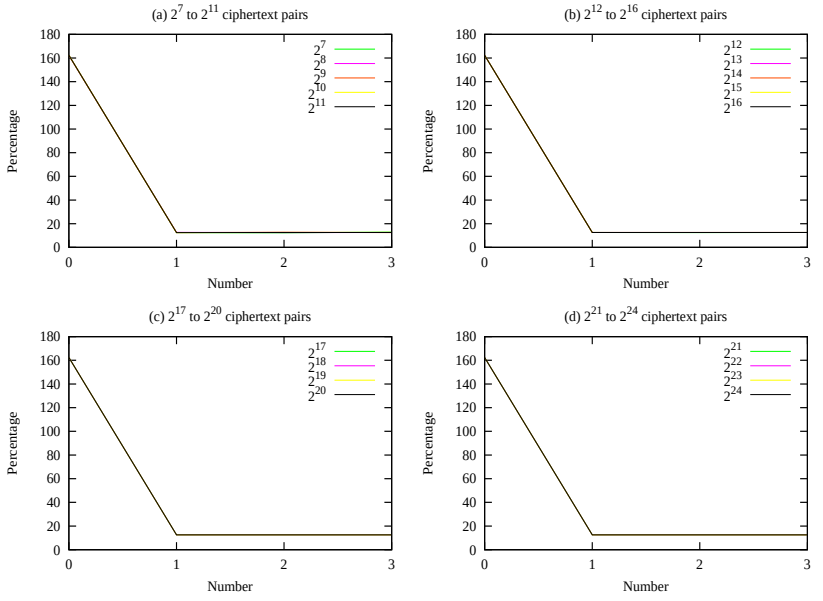

Figure D.10: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 2-round AES-128 for 10 trials

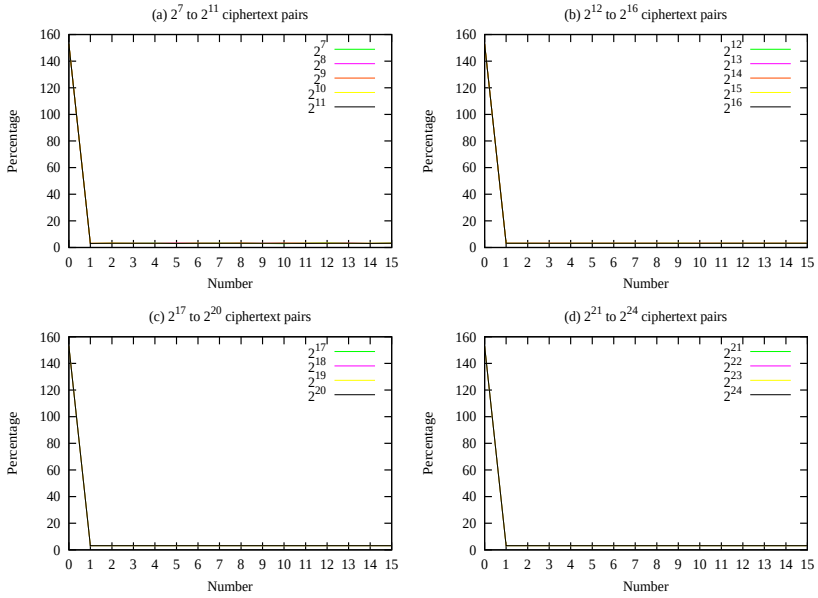

Figure D.11: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 2-round AES-128 for 10 trials

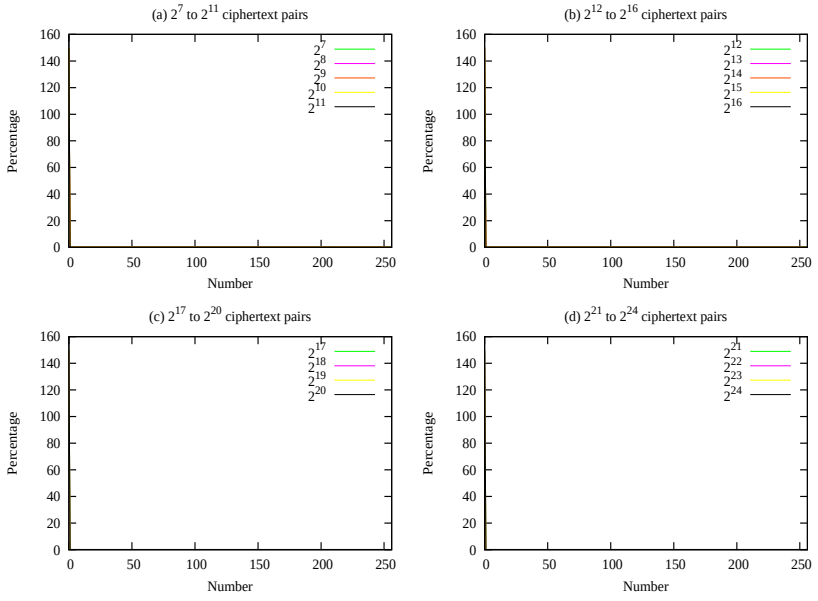

Figure D.12: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 2-round AES-128 for 10 trials

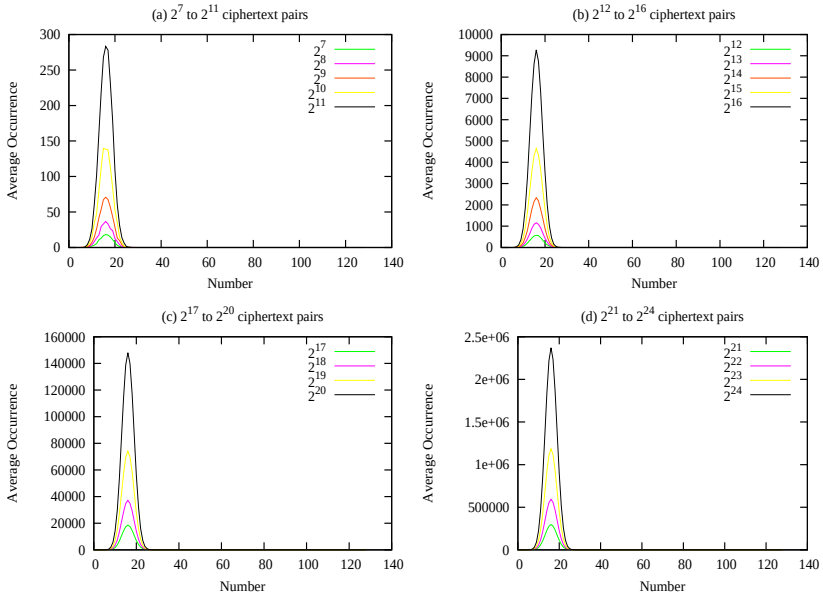

Figure D.13: Average Hamming Weight Distribution between ciphertext pairs for 2-round AES-128 for 10 trials (Autofeeding)

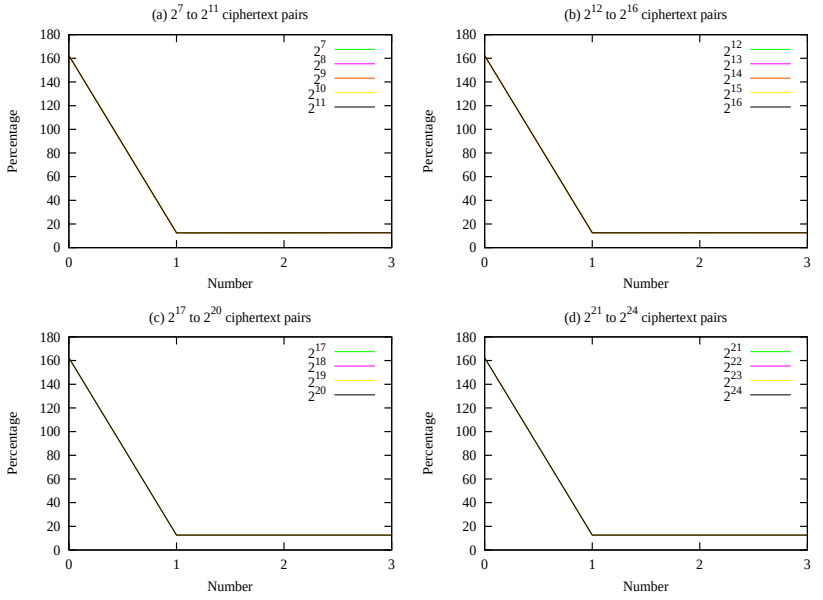

Figure D.14: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 2-round AES-128 for 10 trials (Autofeeding)

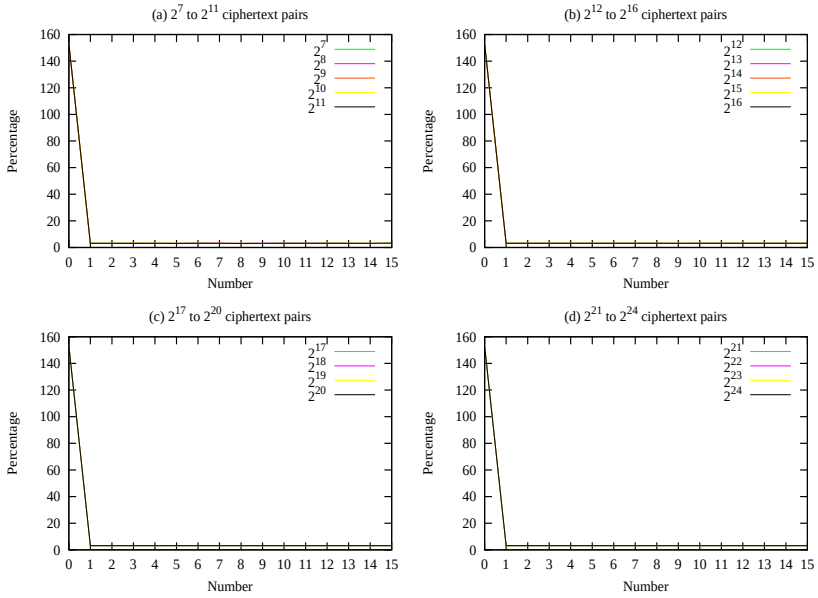

Figure D.15: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 2-round AES-128 for 10 trials (Autofeeding)

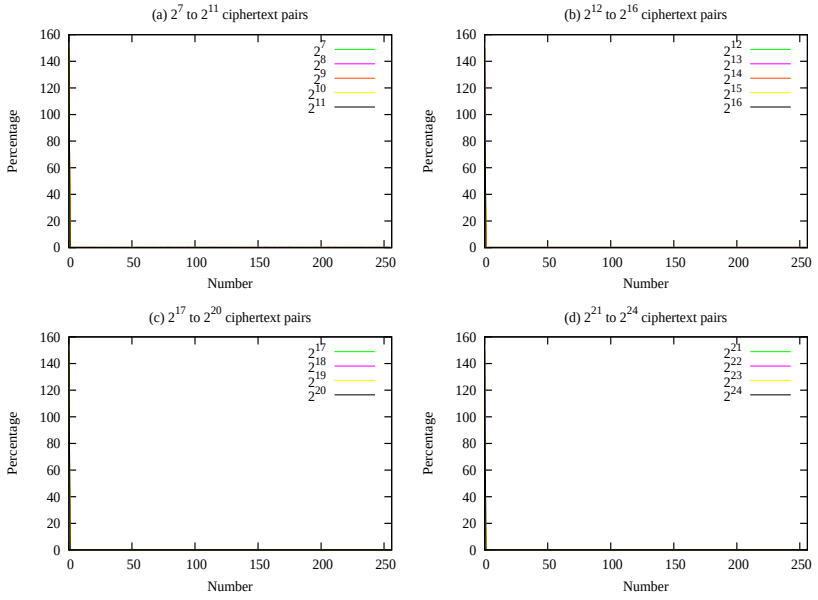

Figure D.16: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 2-round AES-128 for 10 trials (Autofeeding)

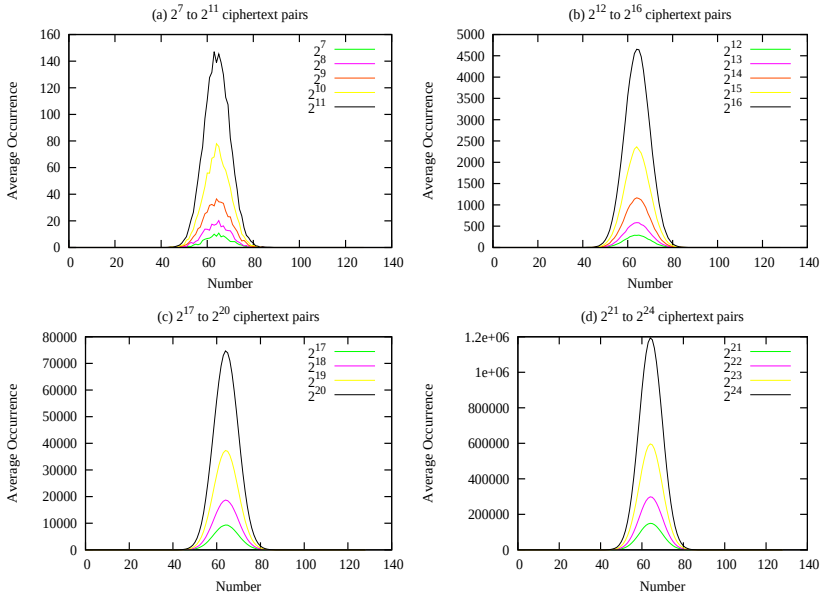

Figure D.17: Average Hamming Weight Distribution between ciphertext pairs for 3-round AES-128 for 10 trials

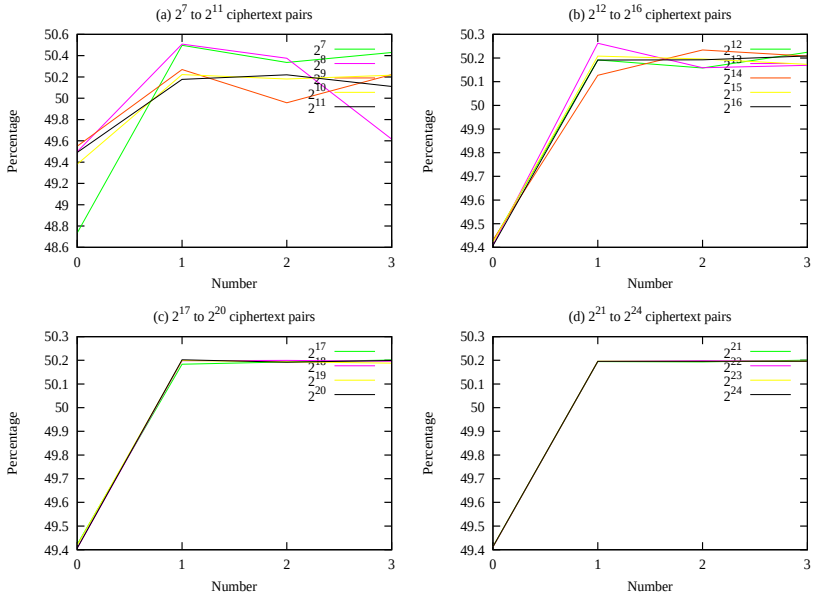

Figure D.18: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 3-round AES-128 for 10 trials

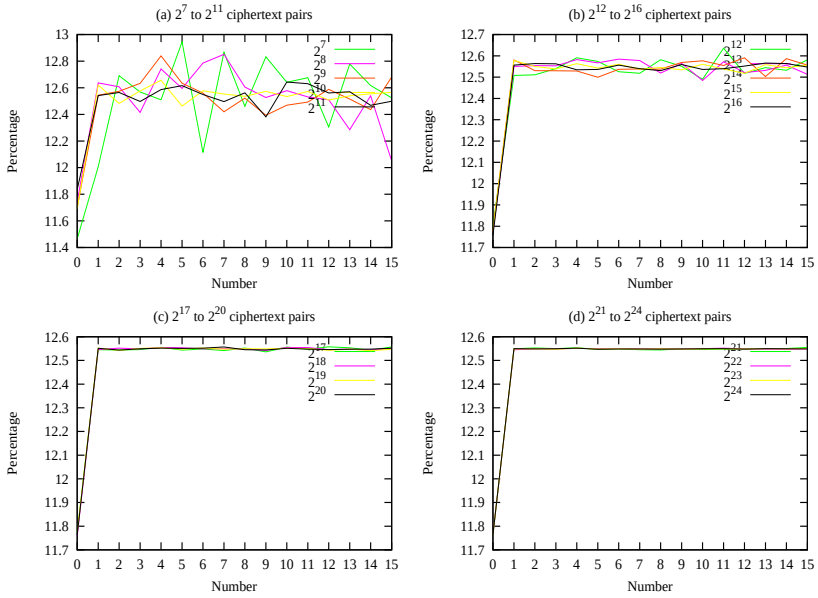

Figure D.19: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 3-round AES-128 for 10 trials

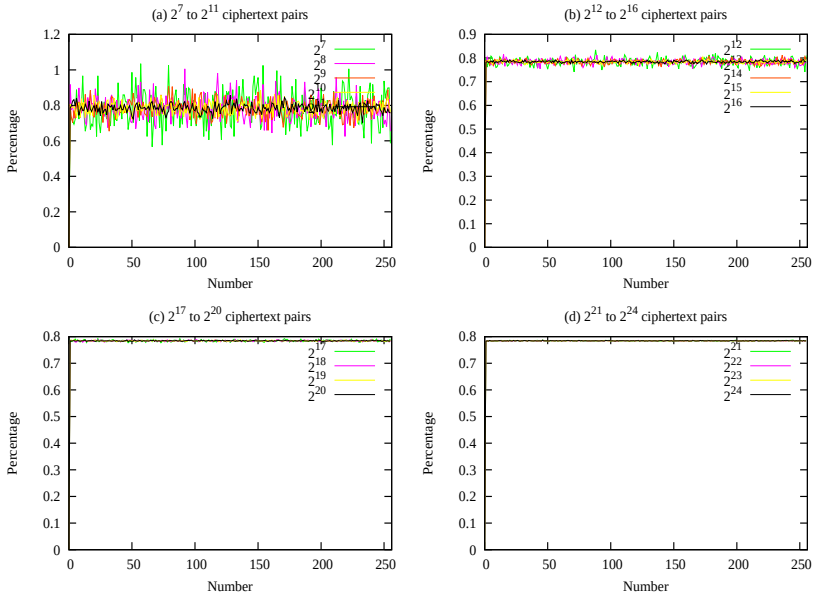

Figure D.20: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 3-round AES-128 for 10 trials

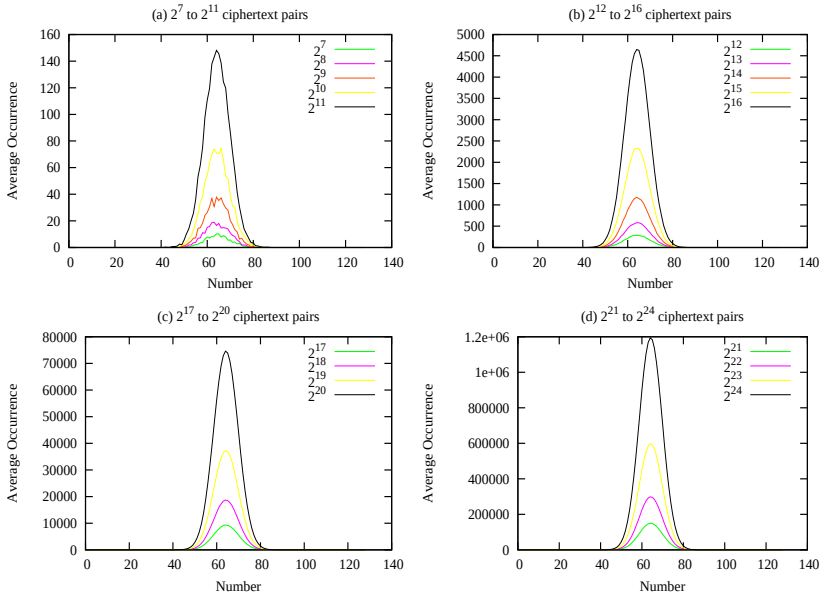

Figure D.21: Average Hamming Weight Distribution between ciphertext pairs for 3-round AES-128 for 10 trials (Autofeeding)

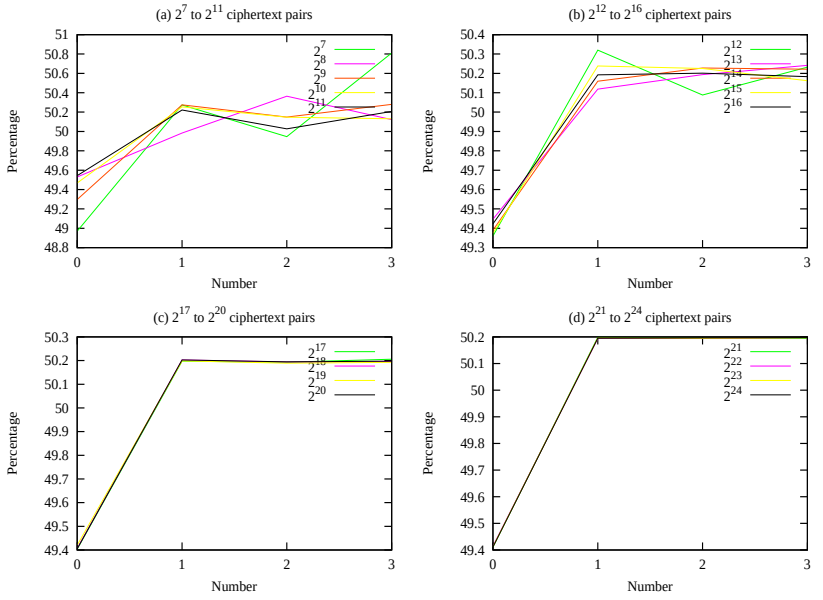

Figure D.22: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 3-round AES-128 for 10 trials (Autofeeding)

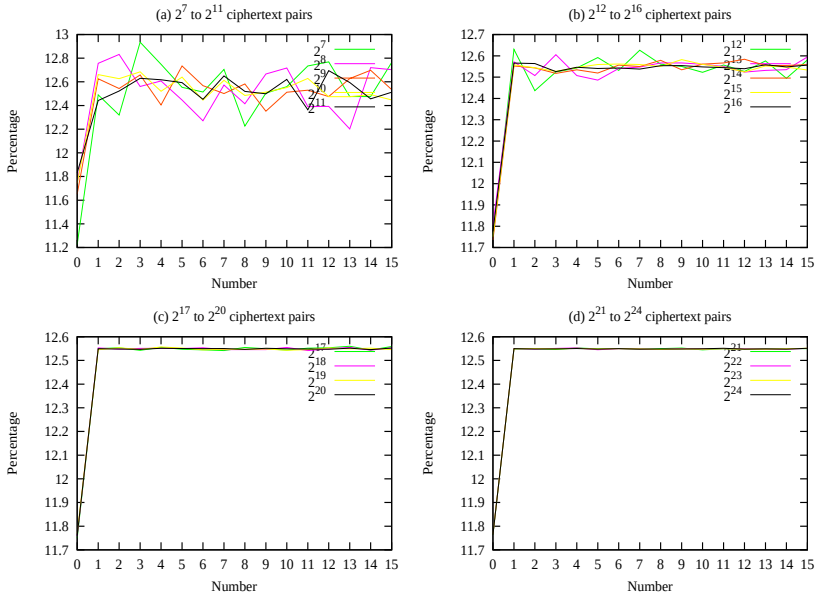

Figure D.23: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 3-round AES-128 for 10 trials (Autofeeding)

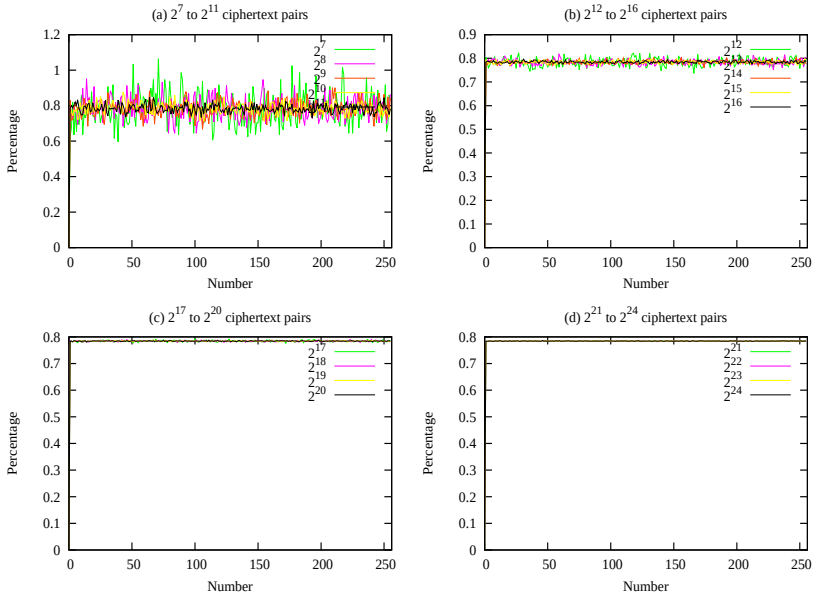

Figure D.24: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 3-round AES-128 for 10 trials (Autofeeding)

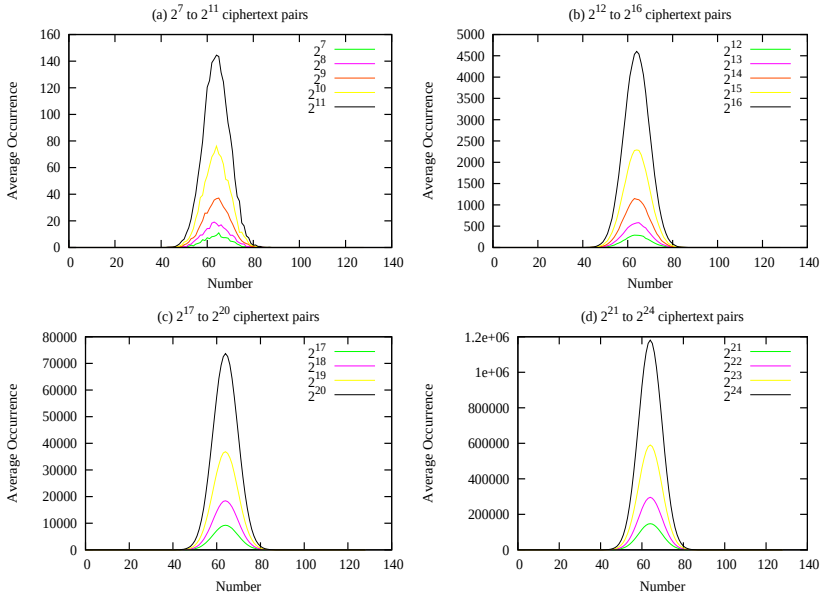

Figure D.25: Average Hamming Weight Distribution between ciphertext pairs for 4-round AES-128 for 10 trials

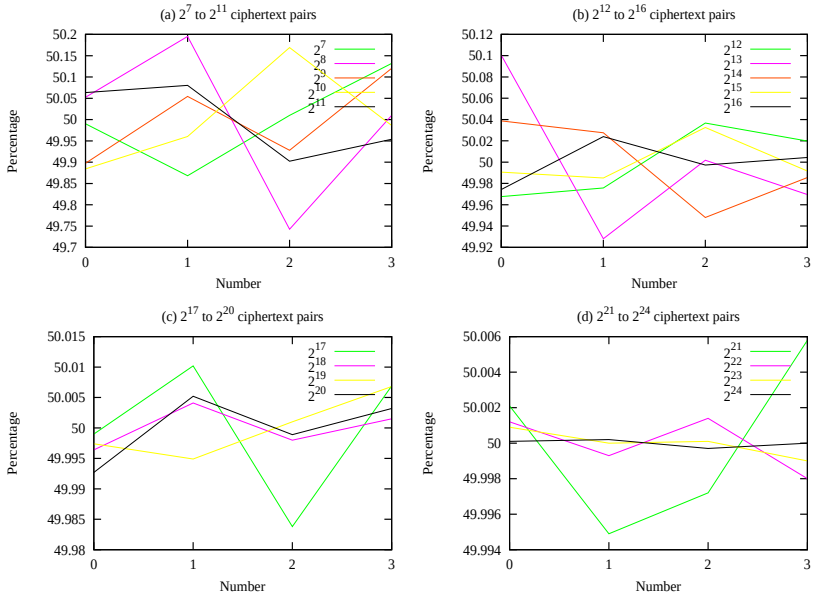

Figure D.26: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 4-round AES-128 for 10 trials

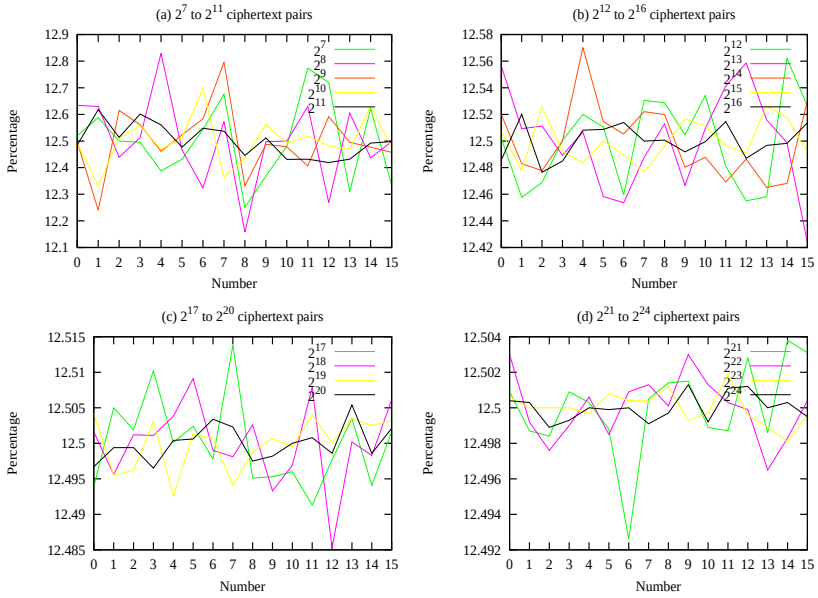

Figure D.27: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 4-round AES-128 for 10 trials

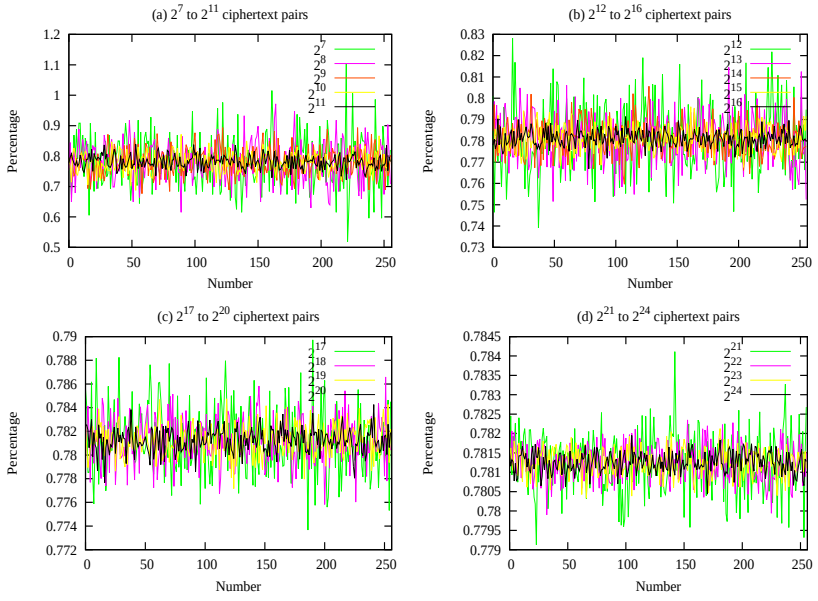

Figure D.28: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 4-round AES-128 for 10 trials

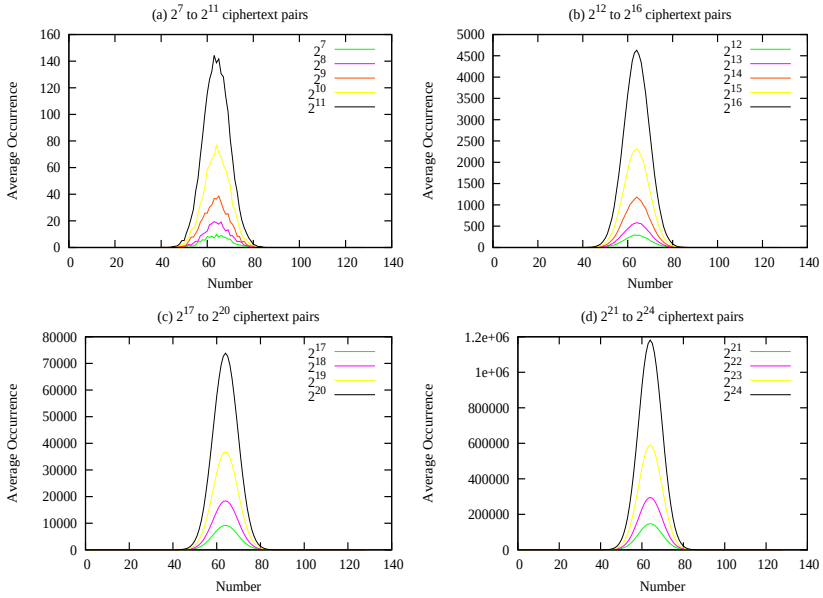

Figure D.29: Average Hamming Weight Distribution between ciphertext pairs for 4-round AES-128 for 10 trials (Autofeeding)

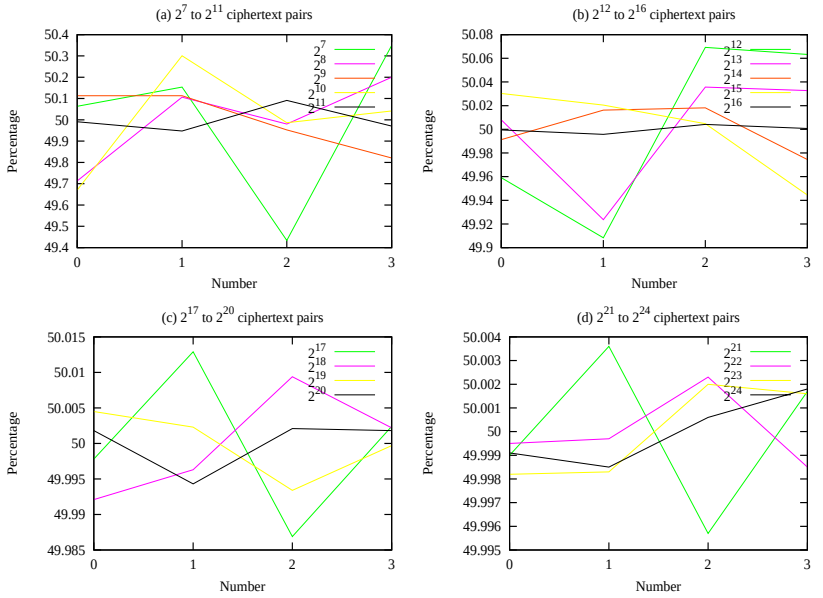

Figure D.30: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 4-round AES-128 for 10 trials (Autofeeding)

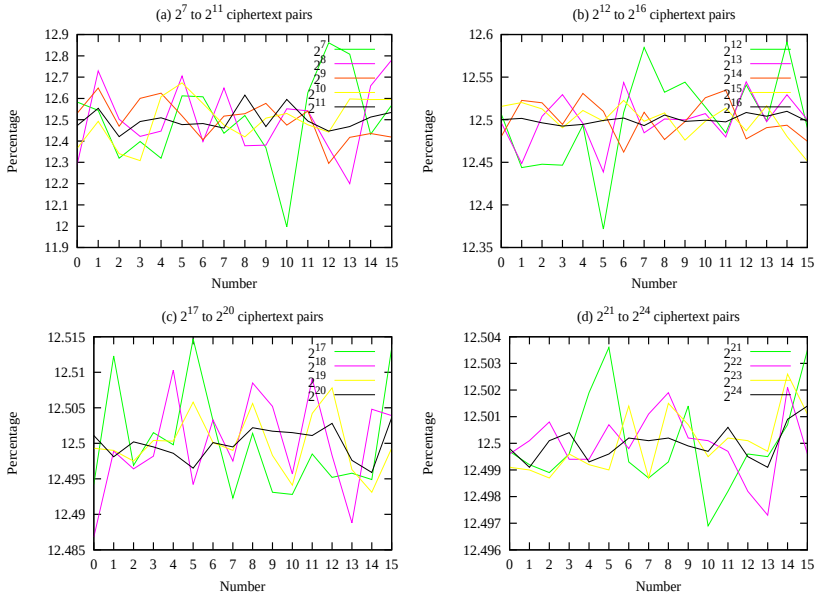

Figure D.31: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 4-round AES-128 for 10 trials (Autofeeding)

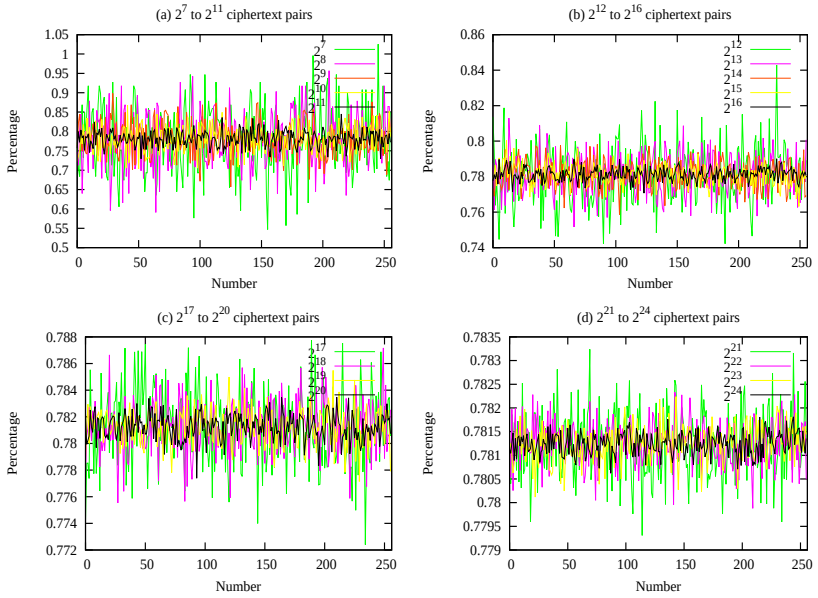

Figure D.32: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 4-round AES-128 for 10 trials (Autofeeding)

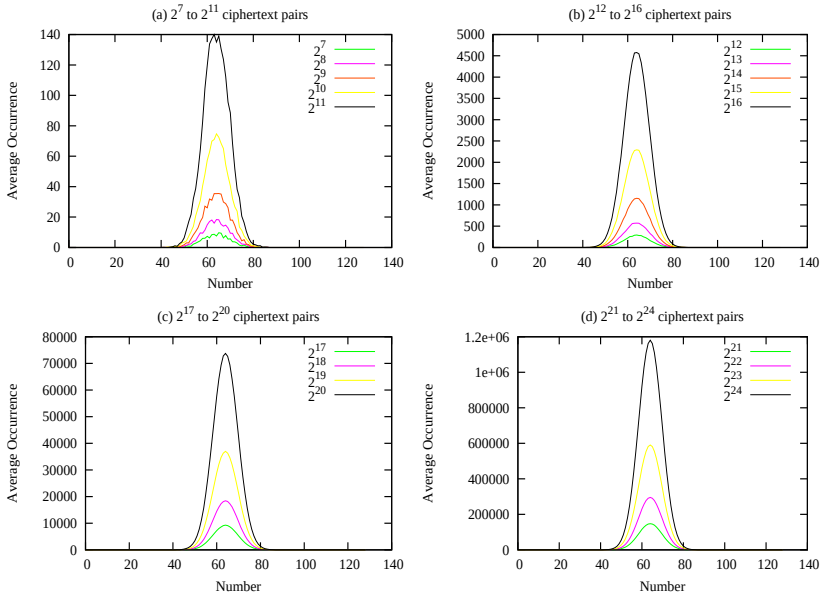

Figure D.33: Average Hamming Weight Distribution between ciphertext pairs for 5-round AES-128 for 10 trials

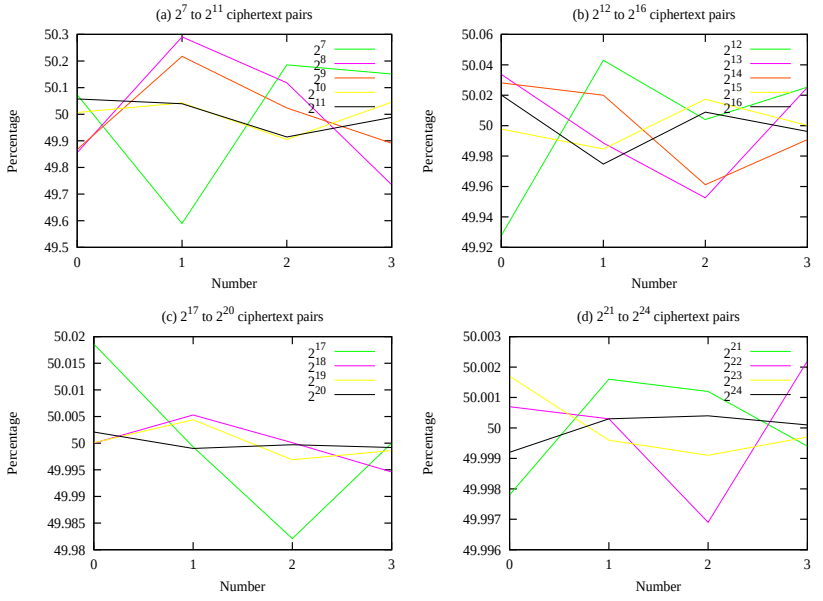

Figure D.34: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 5-round AES-128 for 10 trials

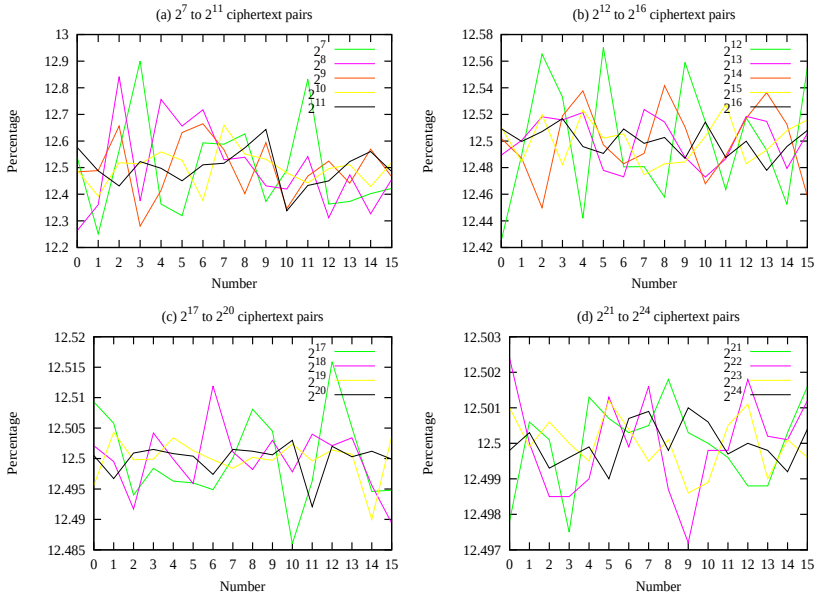

Figure D.35: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 5-round AES-128 for 10 trials

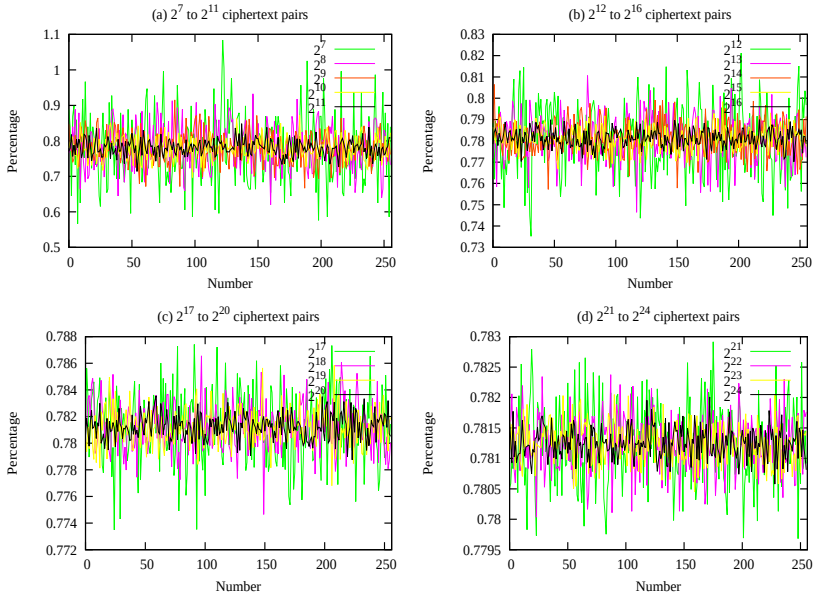

Figure D.36: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 5-round AES-128 for 10 trials

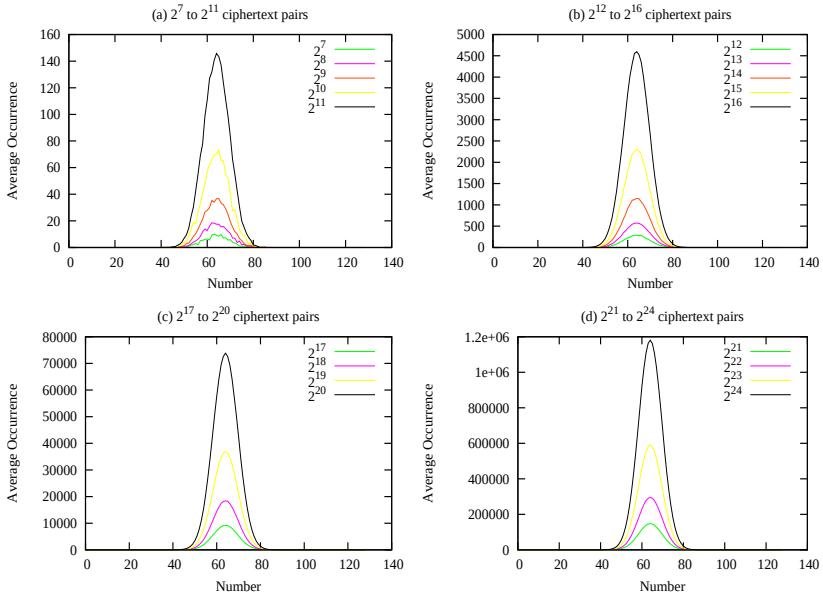

Figure D.37: Average Hamming Weight Distribution between ciphertext pairs for 5-round AES-128 for 10 trials (Autofeeding)

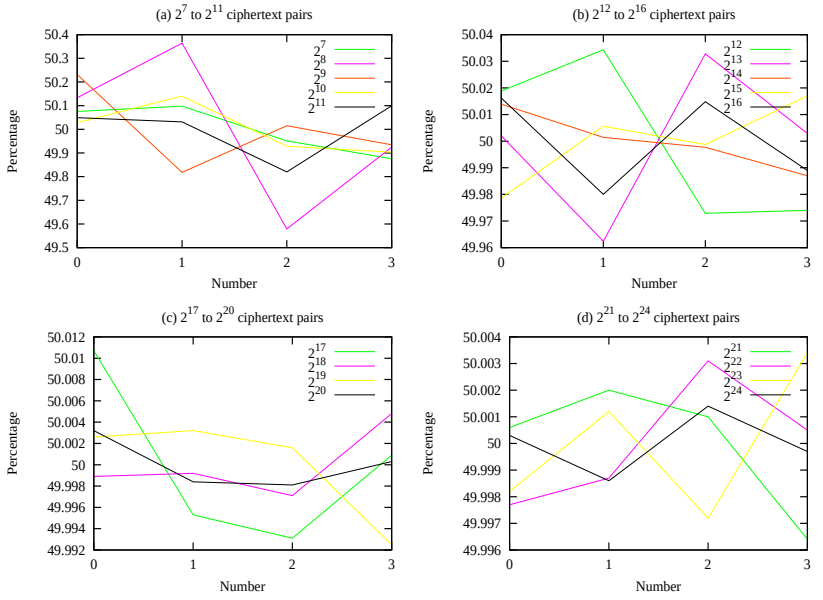

Figure D.38: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 5-round AES-128 for 10 trials (Autofeeding)

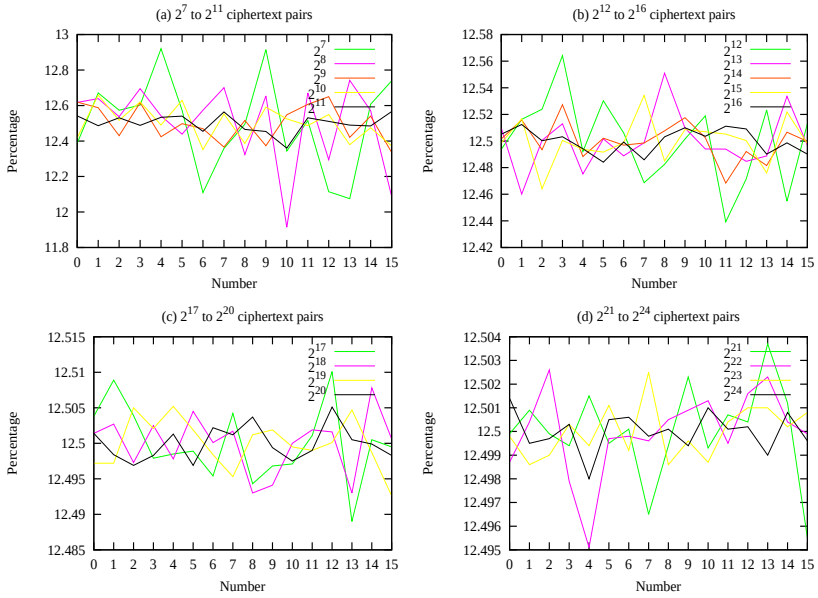

Figure D.39: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 5-round AES-128 for 10 trials (Autofeeding)

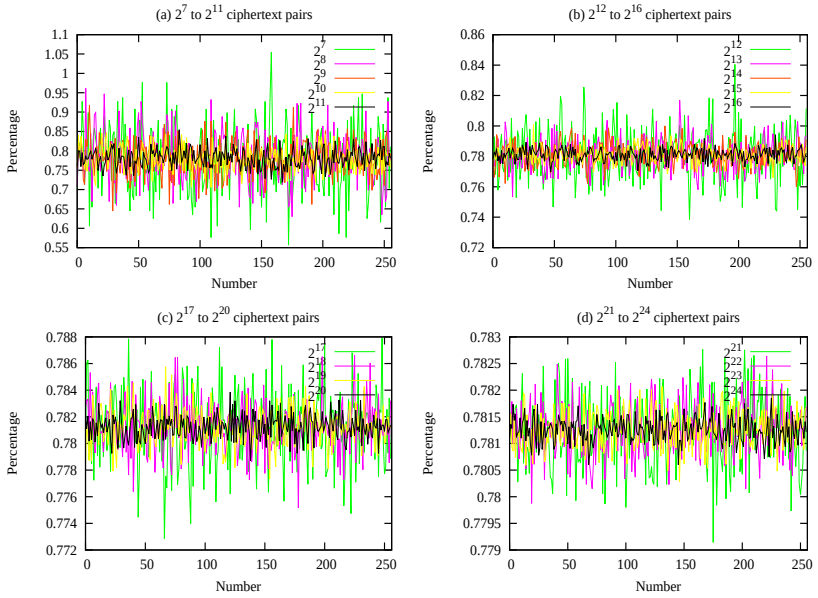

Figure D.40: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 5-round AES-128 for 10 trials (Autofeeding)

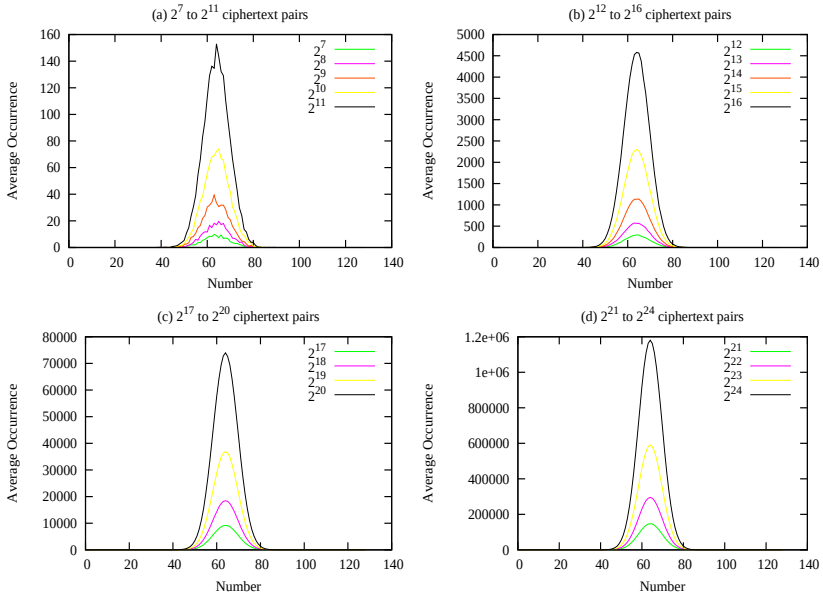

Figure D.41: Average Hamming Weight Distribution between ciphertext pairs for 6-round AES-128 for 10 trials

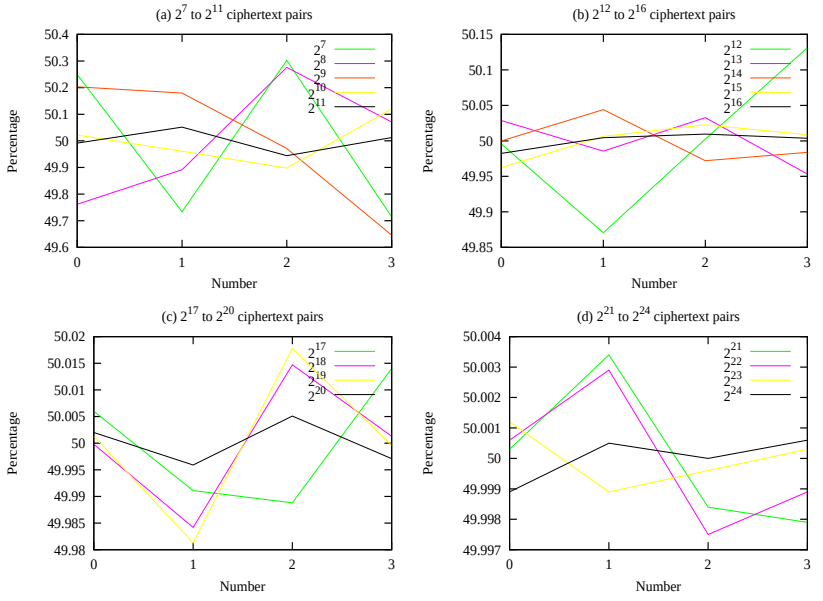

Figure D.42: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 6-round AES-128 for 10 trials

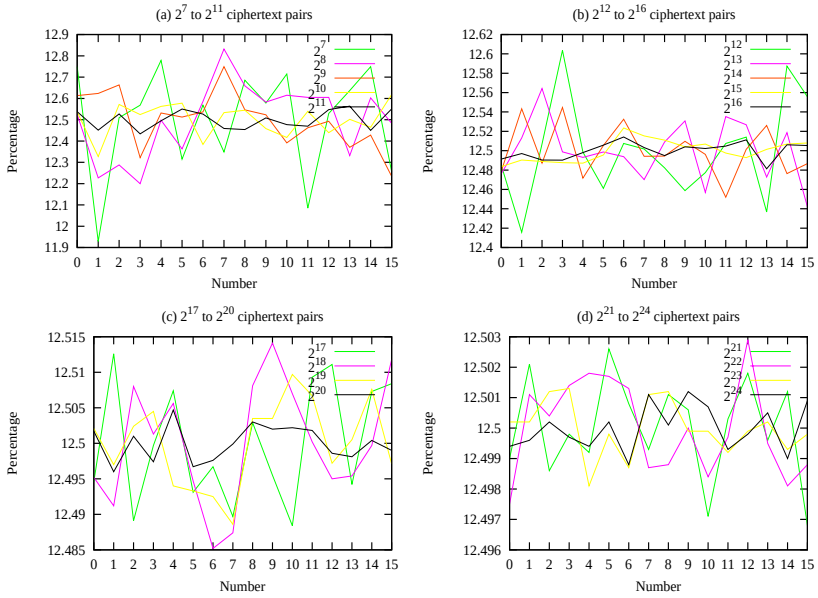

Figure D.43: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 6-round AES-128 for 10 trials

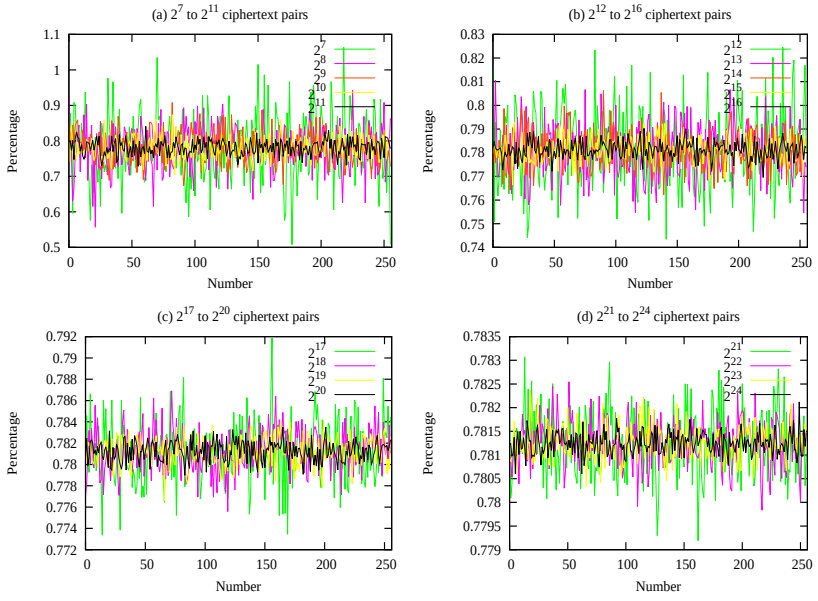

Figure D.44: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 6-round AES-128 for 10 trials

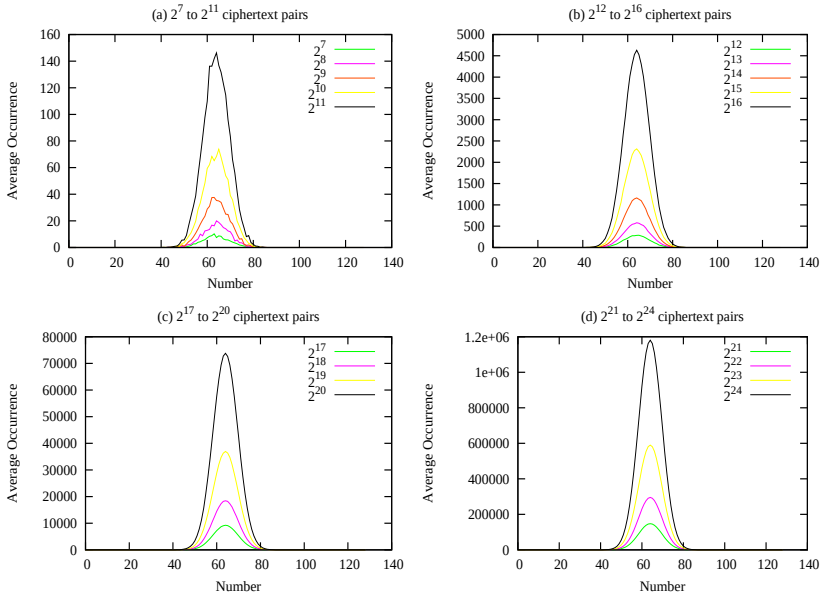

Figure D.45: Average Hamming Weight Distribution between ciphertext pairs for 6-round AES-128 for 10 trials (Autofeeding)

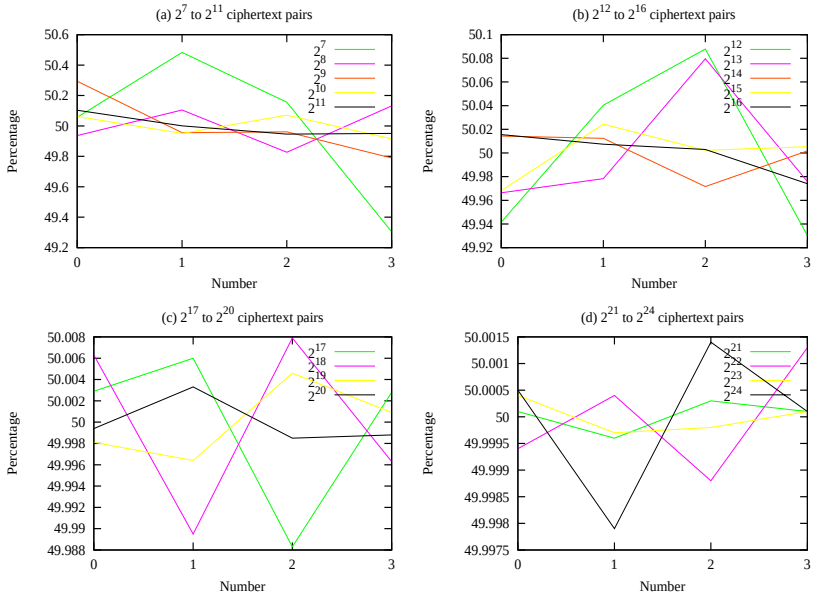

Figure D.46: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 6-round AES-128 for 10 trials (Autofeeding)

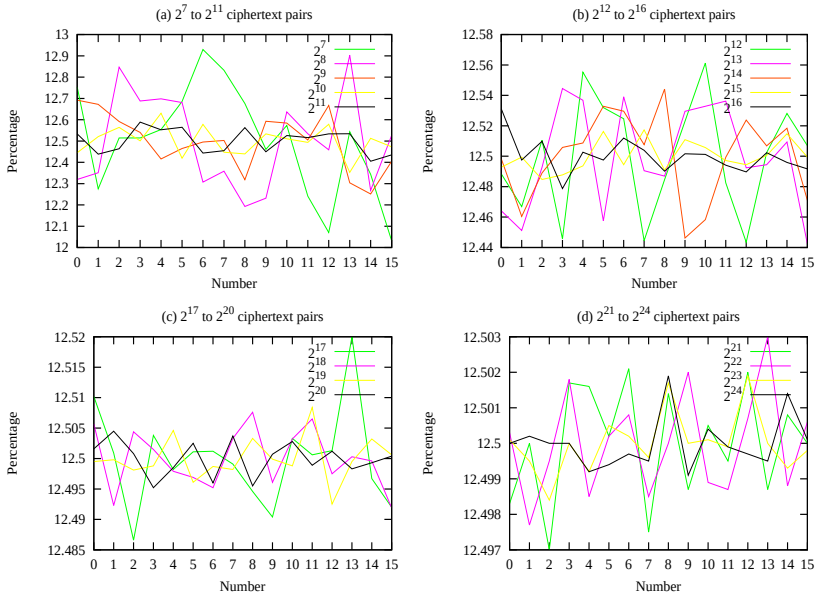

Figure D.47: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 6-round AES-128 for 10 trials (Autofeeding)

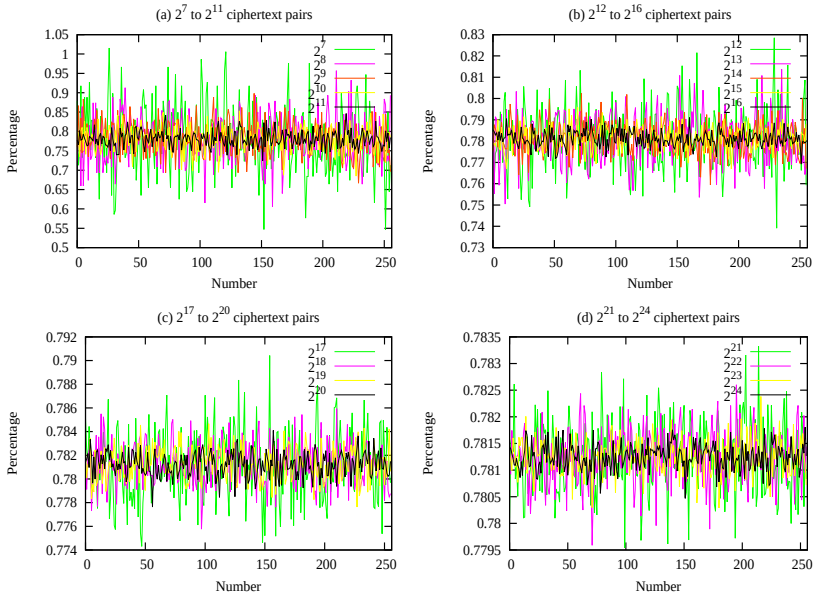

Figure D.48: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 6-round AES-128 for 10 trials (Autofeeding)

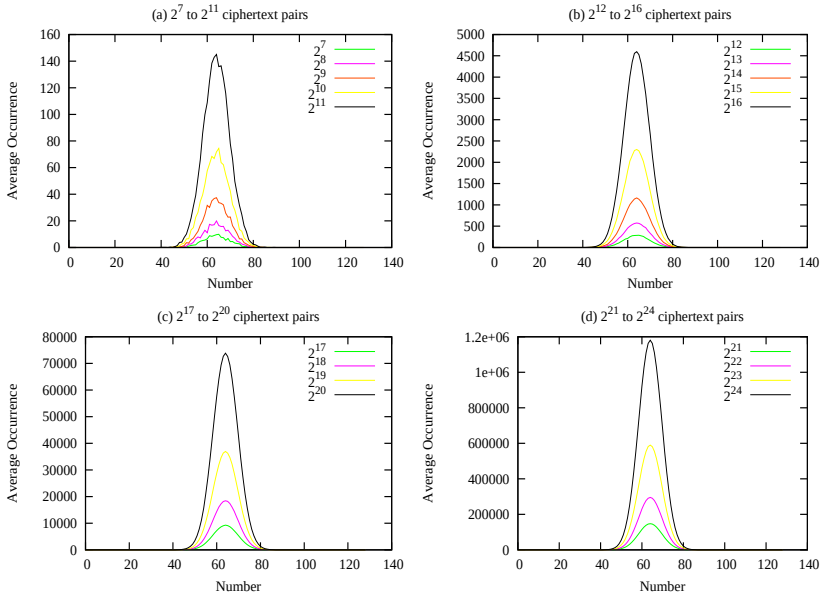

Figure D.49: Average Hamming Weight Distribution between ciphertext pairs for 7-round AES-128 for 10 trials

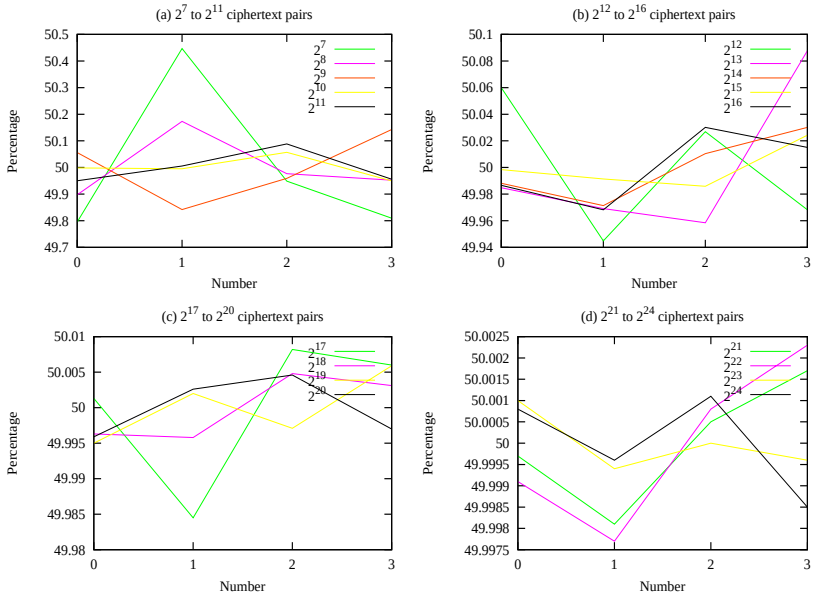

Figure D.50: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 7-round AES-128 for 10 trials

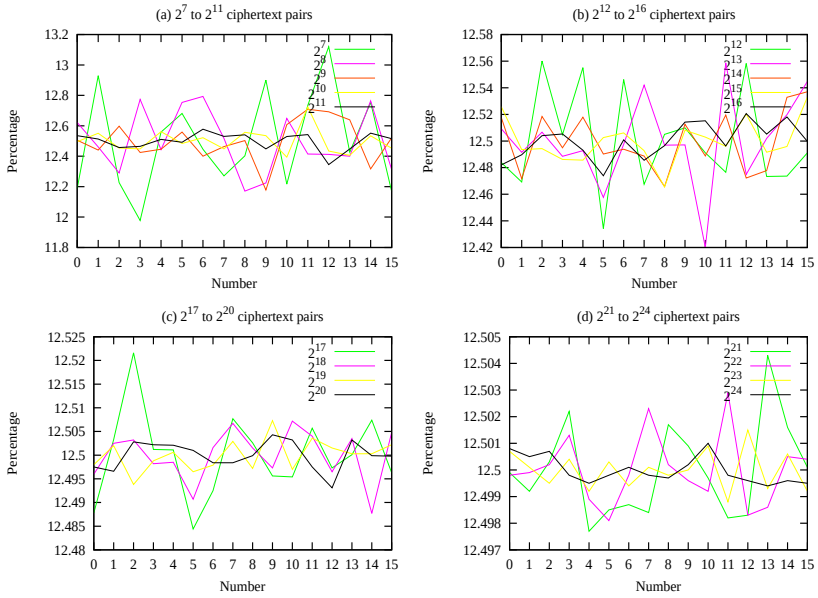

Figure D.51: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 7-round AES-128 for 10 trials

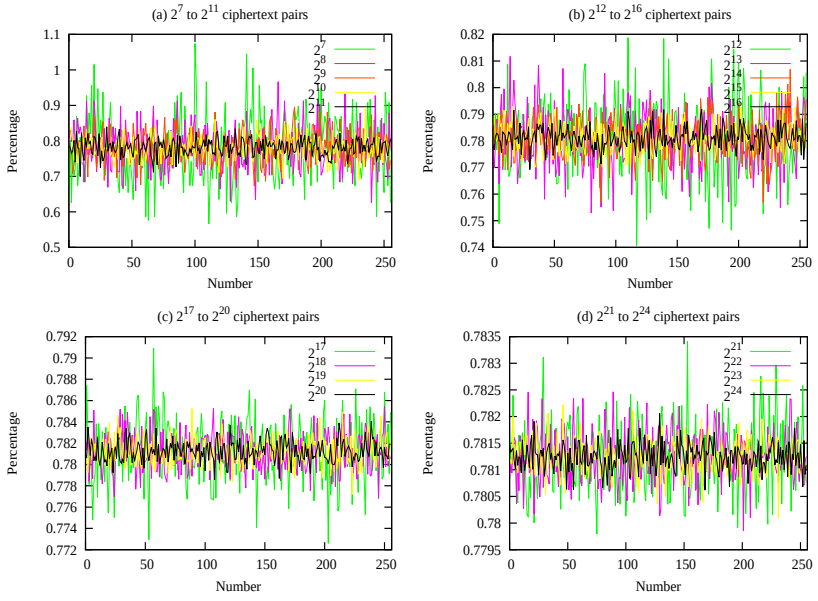

Figure D.52: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 7-round AES-128 for 10 trials

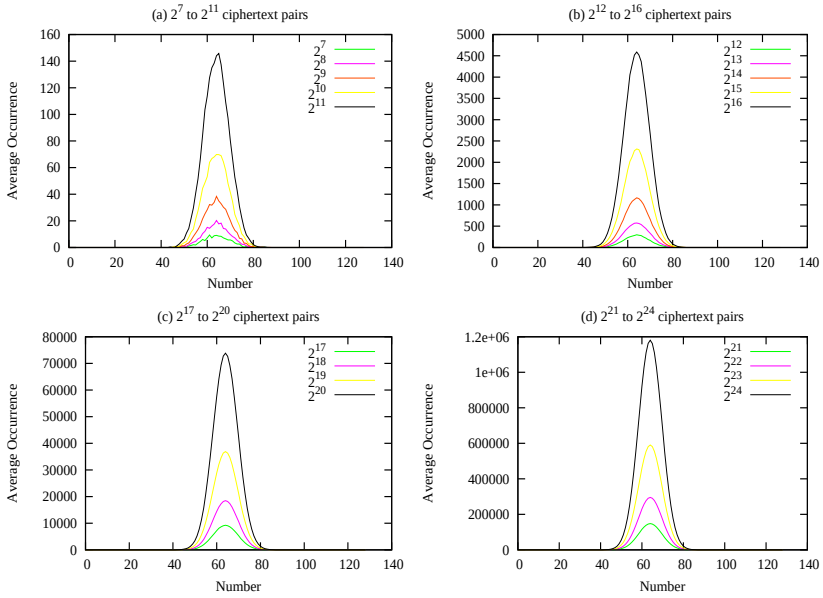

Figure D.53: Average Hamming Weight Distribution between ciphertext pairs for 7-round AES-128 for 10 trials (Autofeeding)

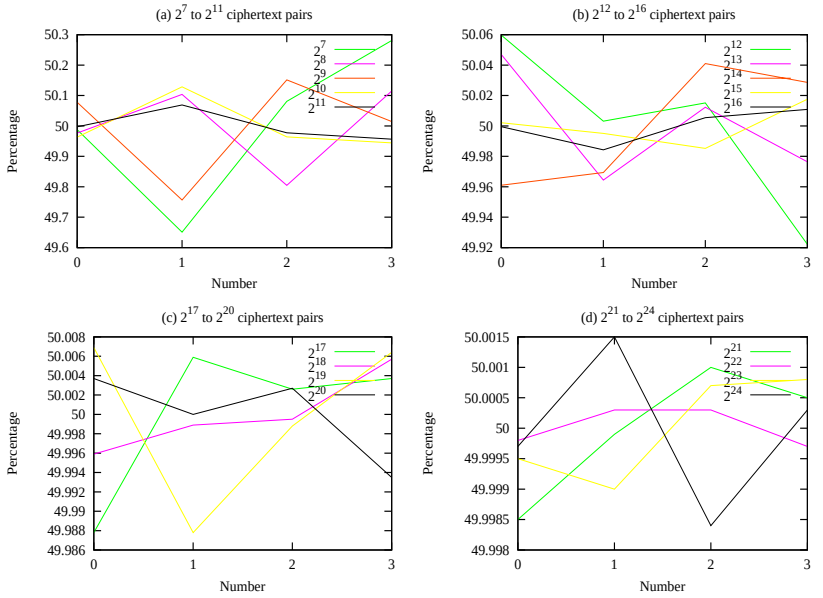

Figure D.54: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 7-round AES-128 for 10 trials (Autofeeding)

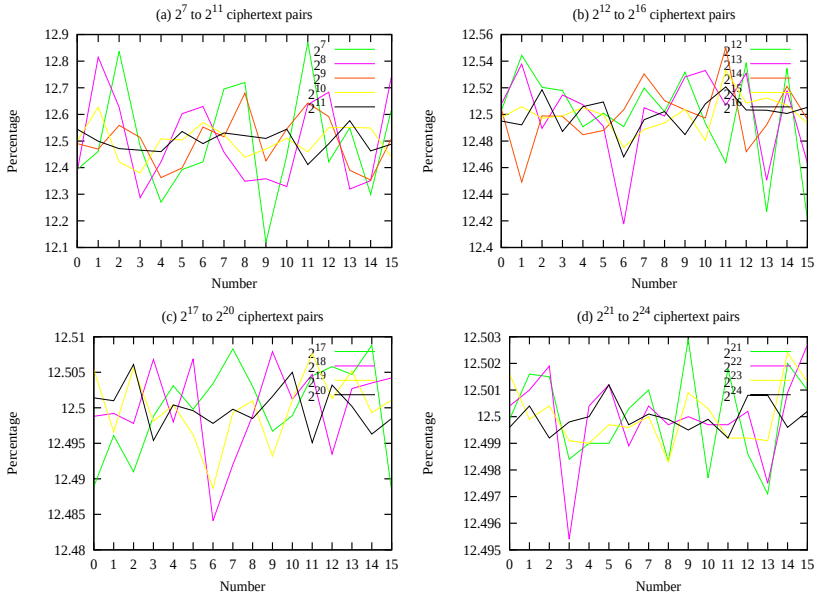

Figure D.55: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 7-round AES-128 for 10 trials (Autofeeding)

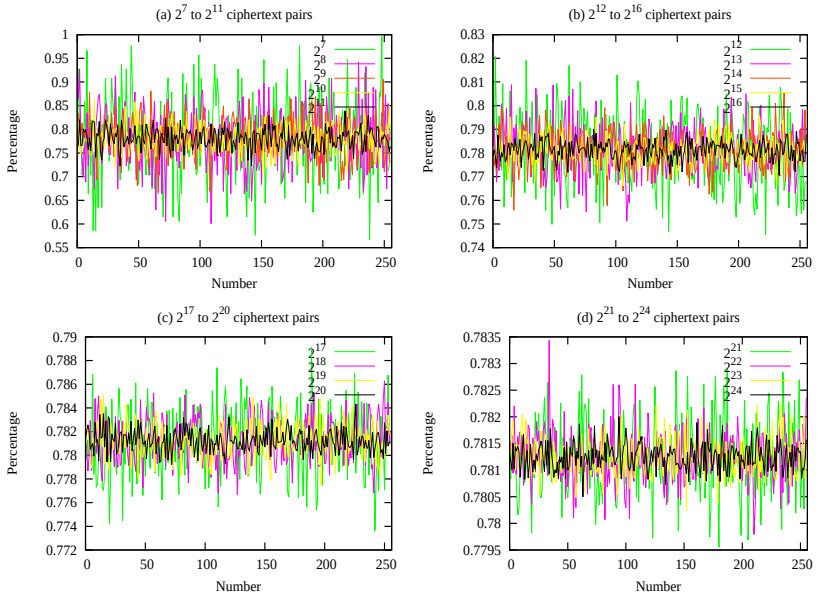

Figure D.56: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 7-round AES-128 for 10 trials (Autofeeding)

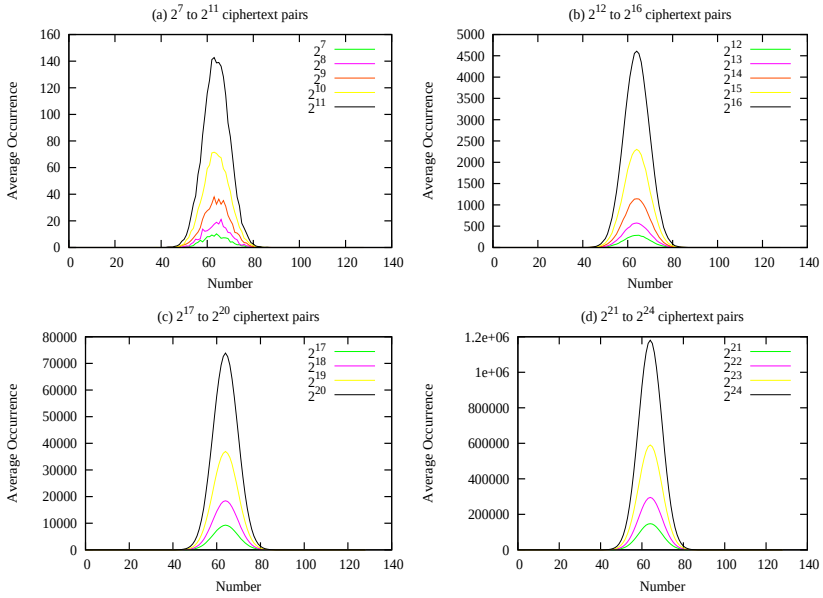

Figure D.57: Average Hamming Weight Distribution between ciphertext pairs for 8-round AES-128 for 10 trials

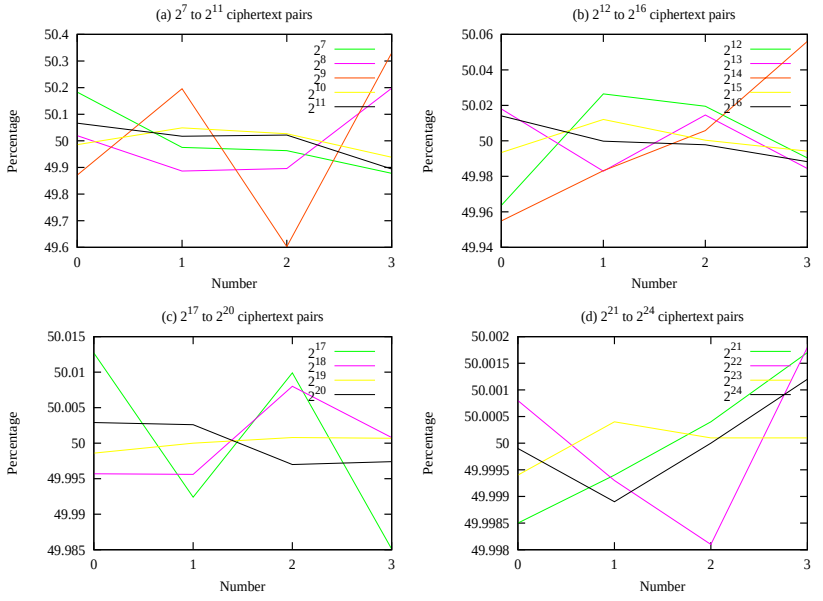

Figure D.58: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 8-round AES-128 for 10 trials

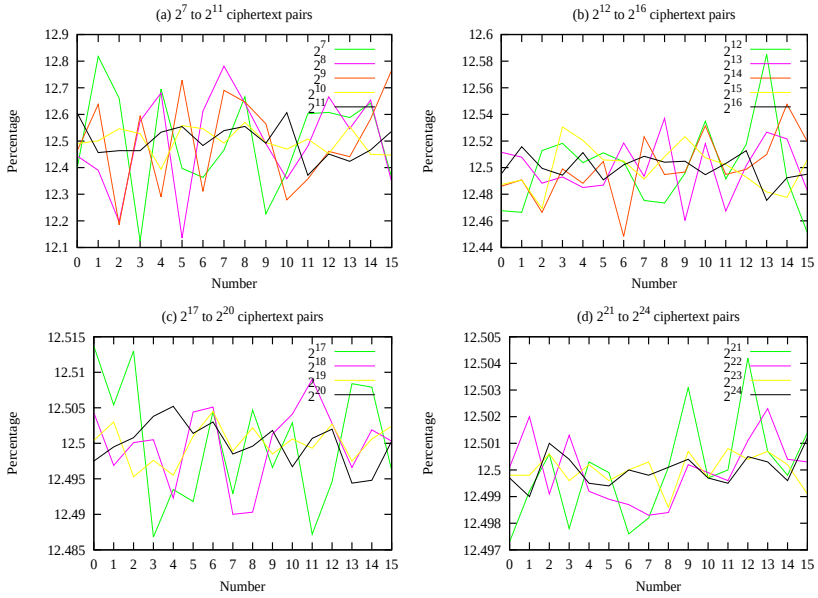

Figure D.59: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 8-round AES-128 for 10 trials

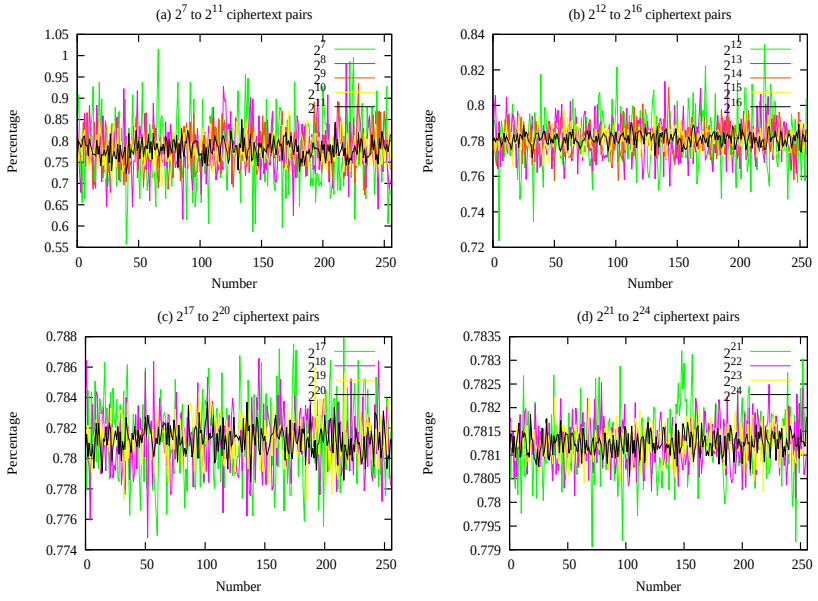

Figure D.60: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 8-round AES-128 for 10 trials

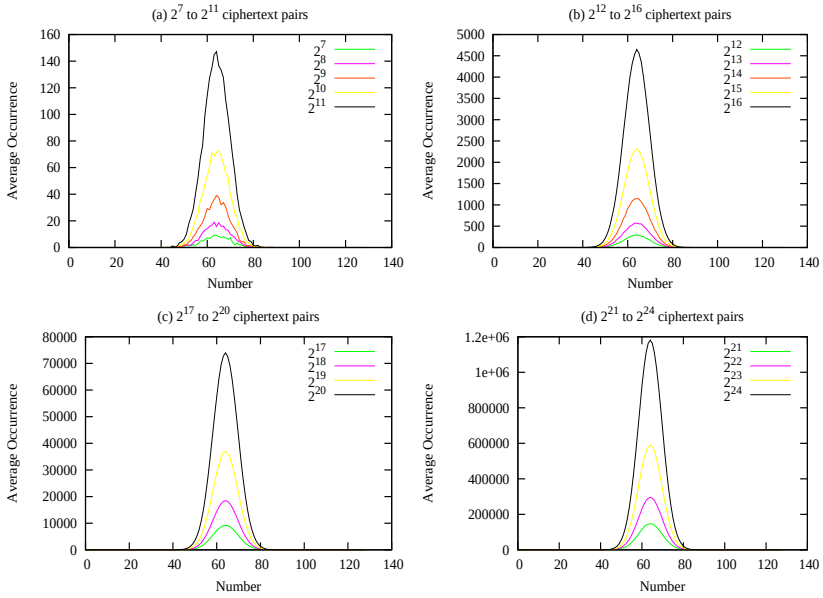

Figure D.61: Average Hamming Weight Distribution between ciphertext pairs for 8-round AES-128 for 10 trials (Autofeeding)

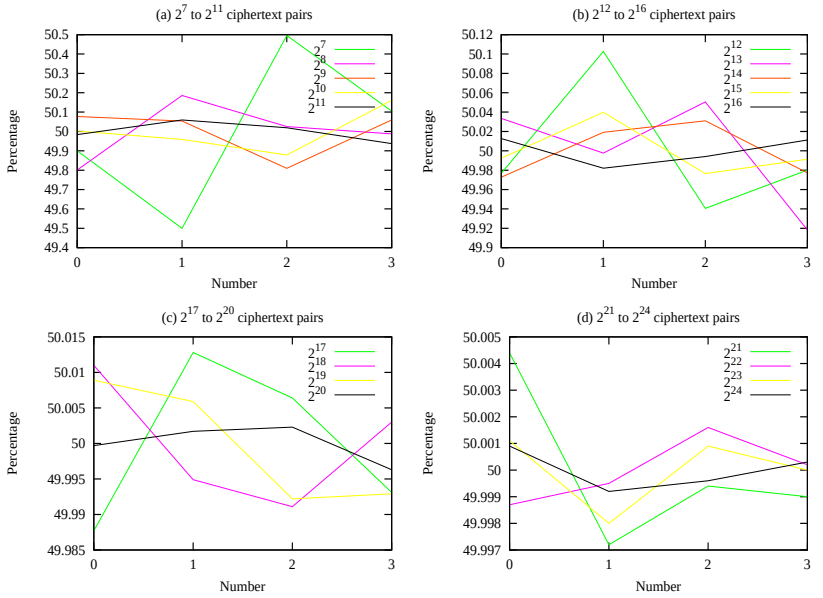

Figure D.62: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 8-round AES-128 for 10 trials (Autofeeding)

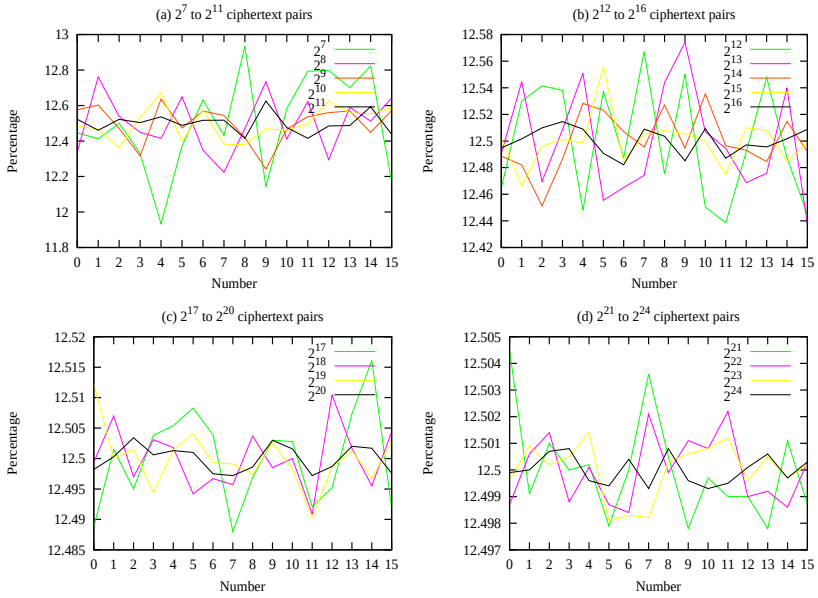

Figure D.63: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 8-round AES-128 for 10 trials (Autofeeding)

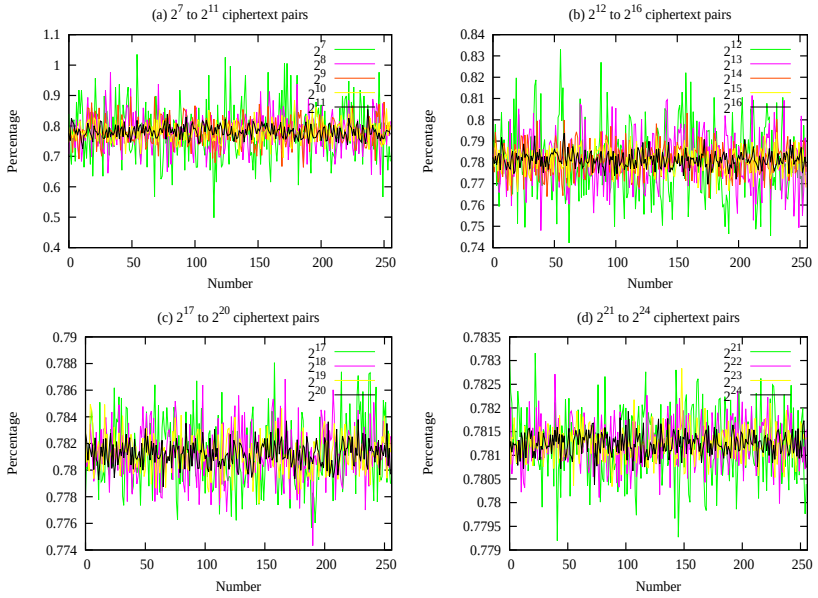

Figure D.64: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 8-round AES-128 for 10 trials (Autofeeding)

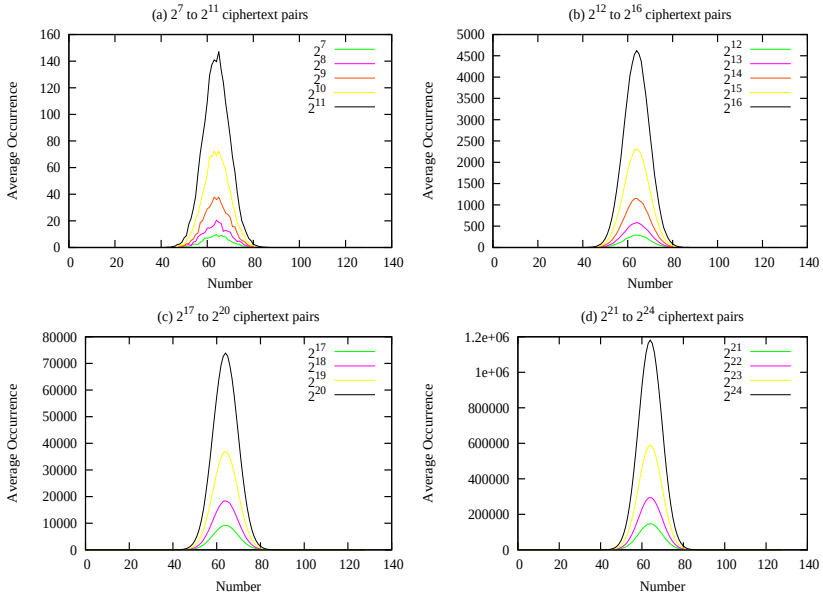

Figure D.65: Average Hamming Weight Distribution between ciphertext pairs for 9-round AES-128 for 10 trials

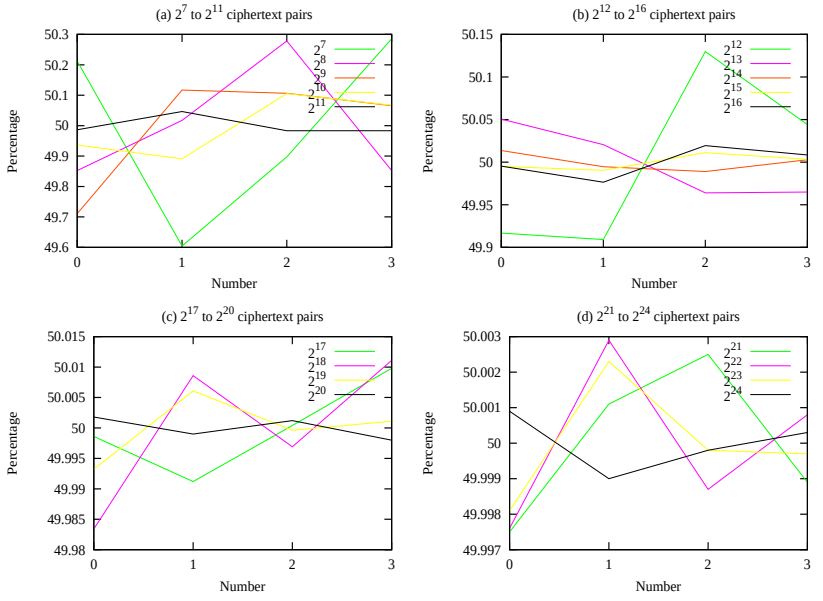

Figure D.66: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 9-round AES-128 for 10 trials

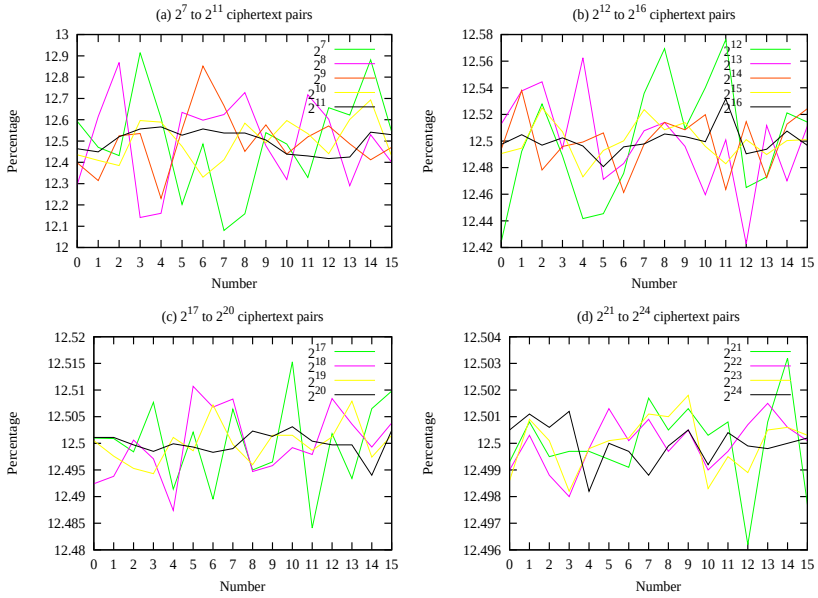

Figure D.67: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 9-round AES-128 for 10 trials

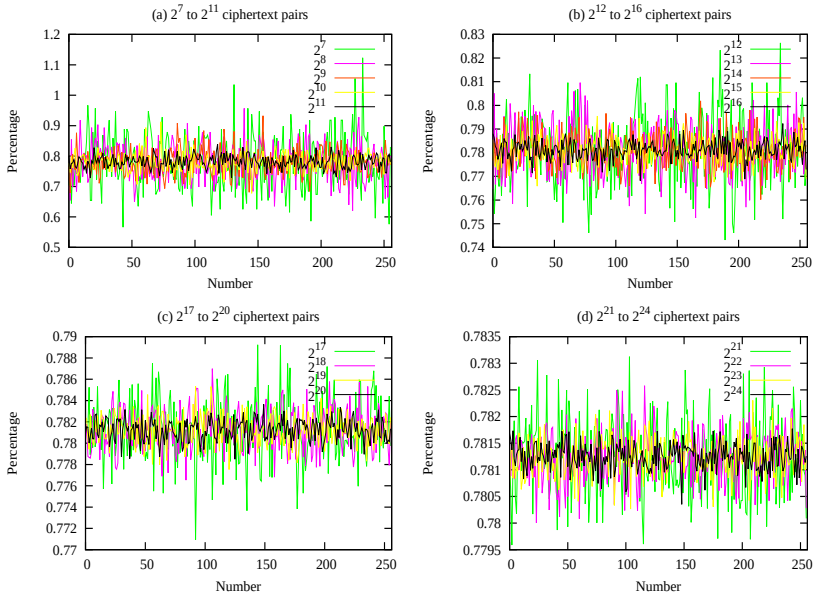

Figure D.68: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 9-round AES-128 for 10 trials

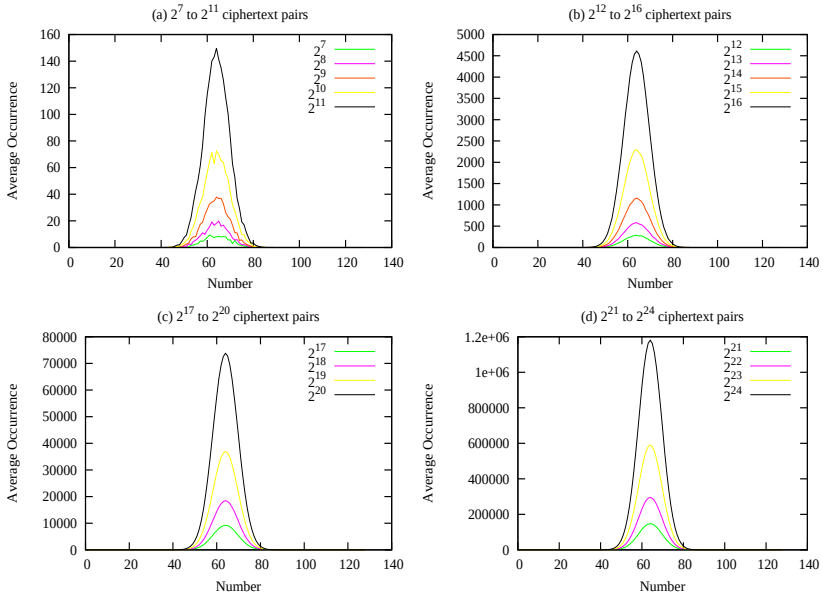

Figure D.69: Average Hamming Weight Distribution between ciphertext pairs for 9-round AES-128 for 10 trials (Autofeeding)

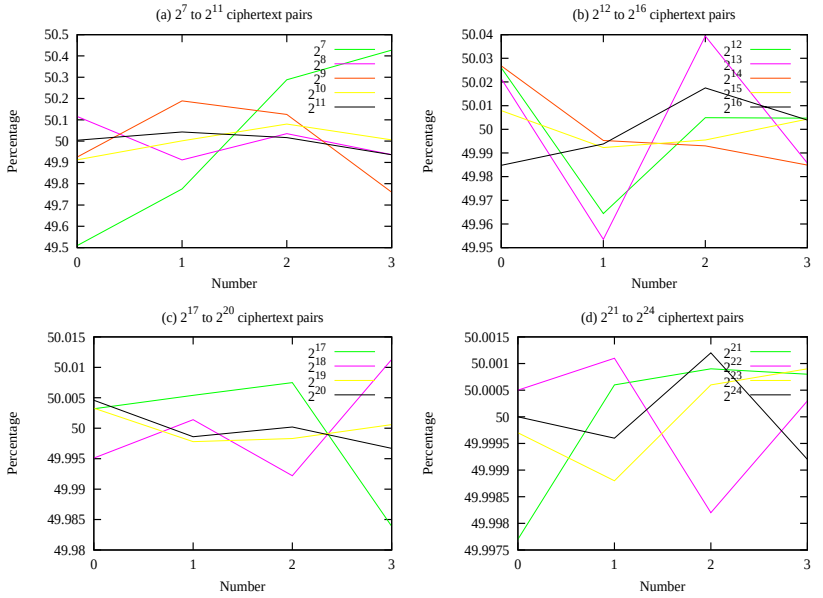

Figure D.70: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 9-round AES-128 for 10 trials (Autofeeding)

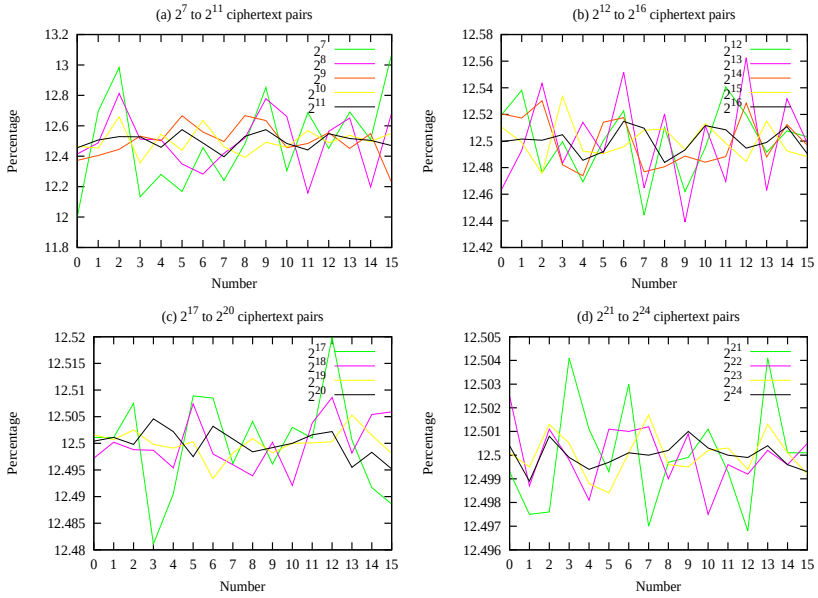

Figure D.71: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 9-round AES-128 for 10 trials (Autofeeding)

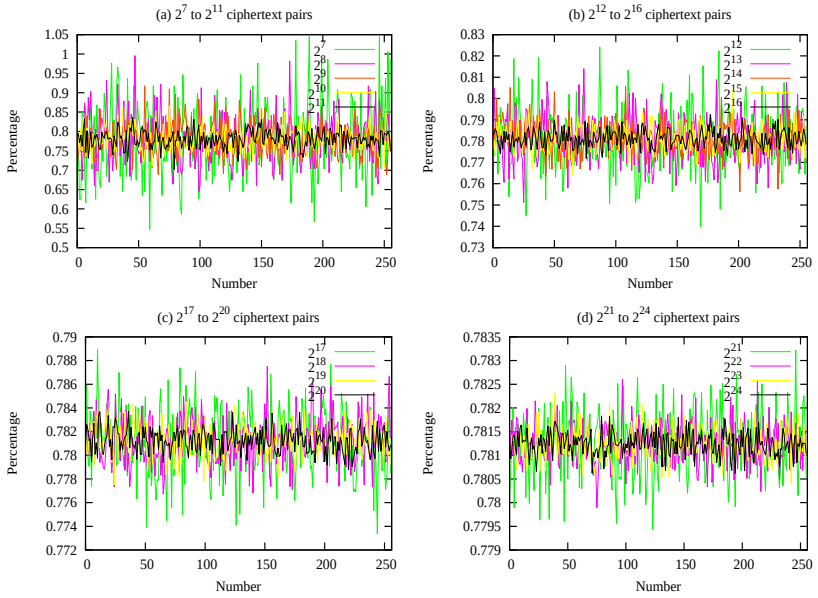

Figure D.72: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 9-round AES-128 for 10 trials (Autofeeding)

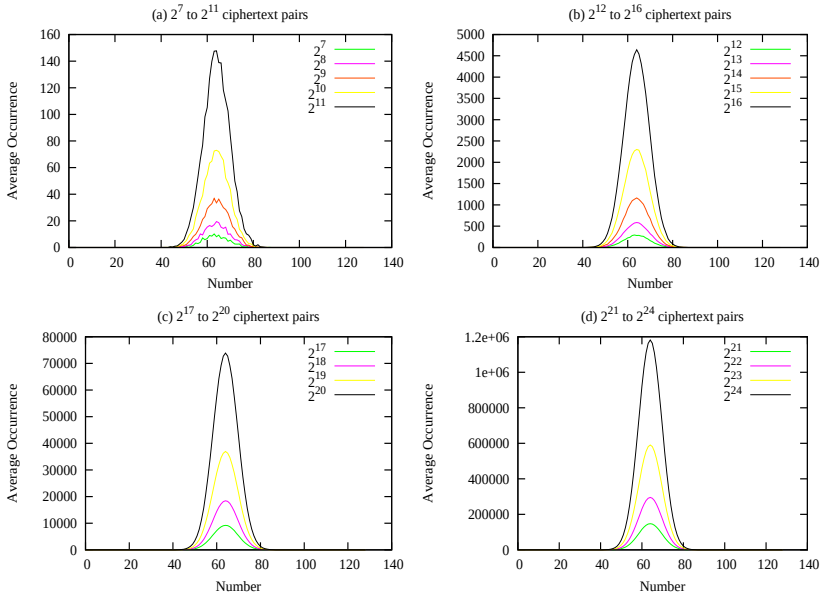

Figure D.73: Average Hamming Weight Distribution between ciphertext pairs for 10-round AES-128 for 10 trials

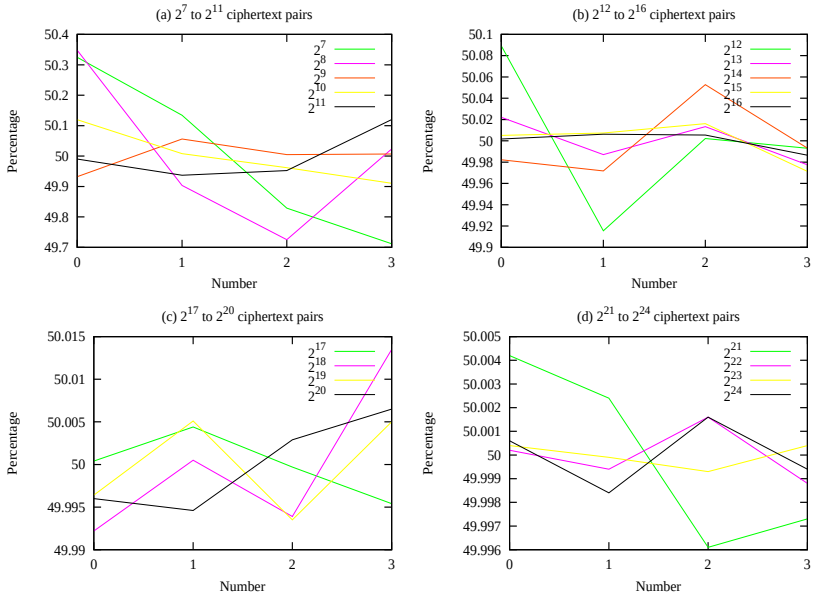

Figure D.74: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 10-round AES-128 for 10 trials

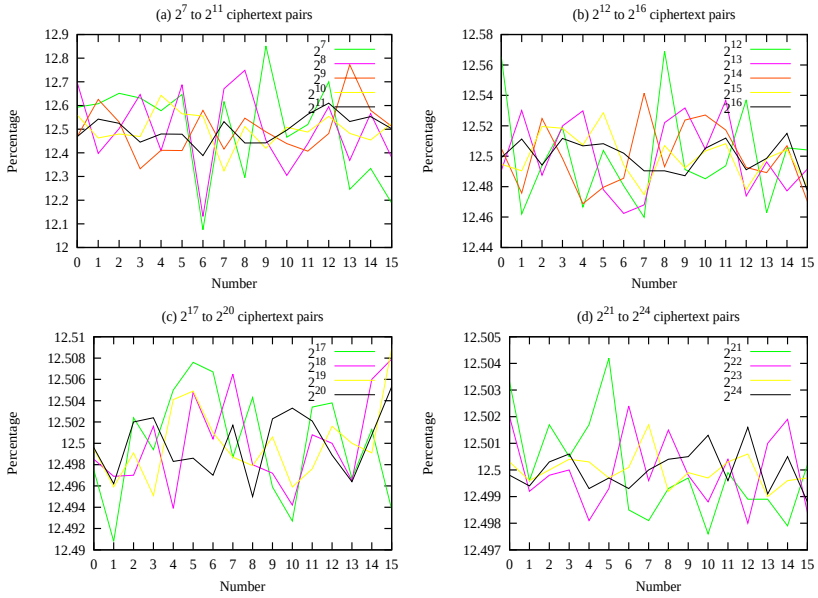

Figure D.75: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 10-round AES-128 for 10 trials

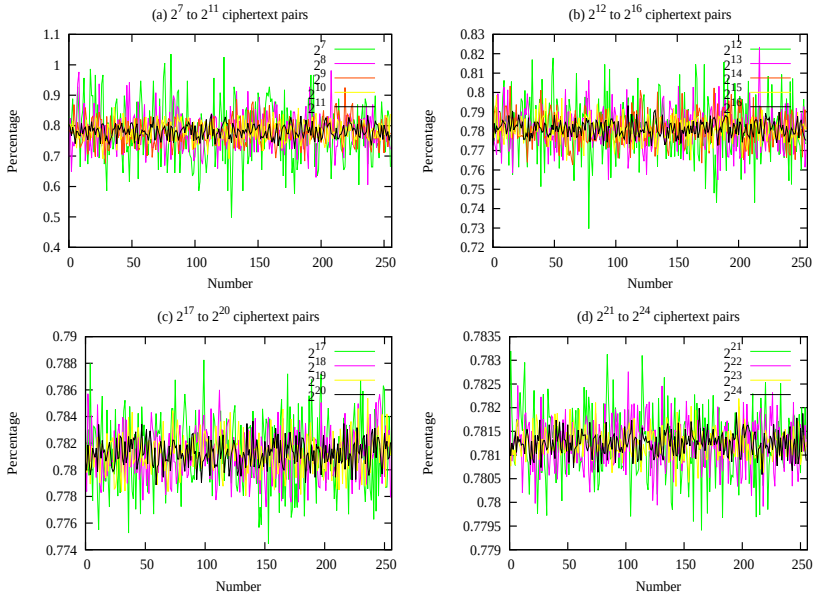

Figure D.76: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 10-round AES-128 for 10 trials

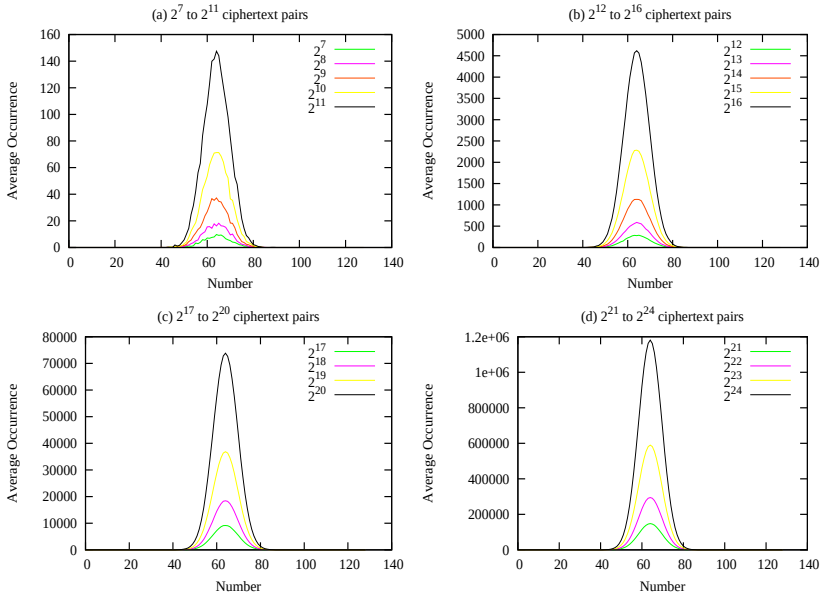

Figure D.77: Average Hamming Weight Distribution between ciphertext pairs for 10-round AES-128 for 10 trials (Autofeeding)

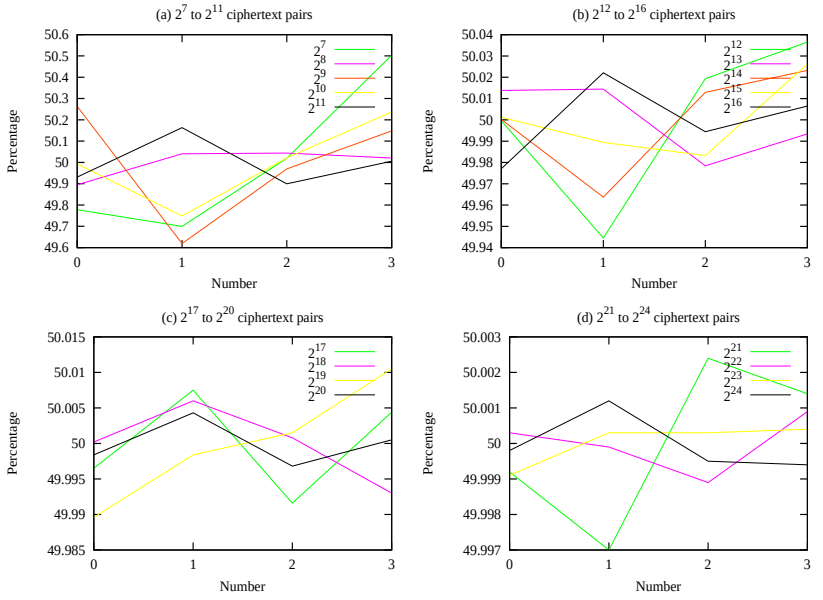

Figure D.78: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 10-round AES-128 for 10 trials (Autofeeding)

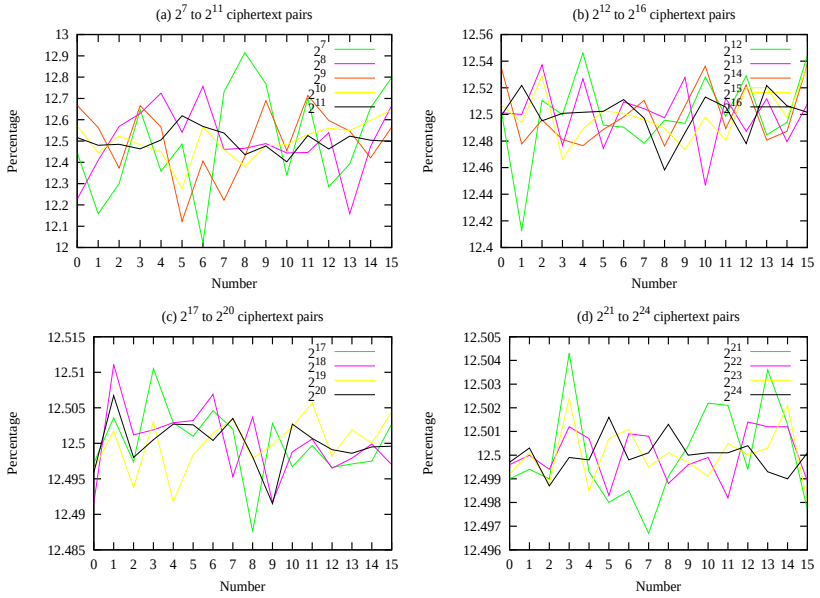

Figure D.79: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 10-round AES-128 for 10 trials (Autofeeding)

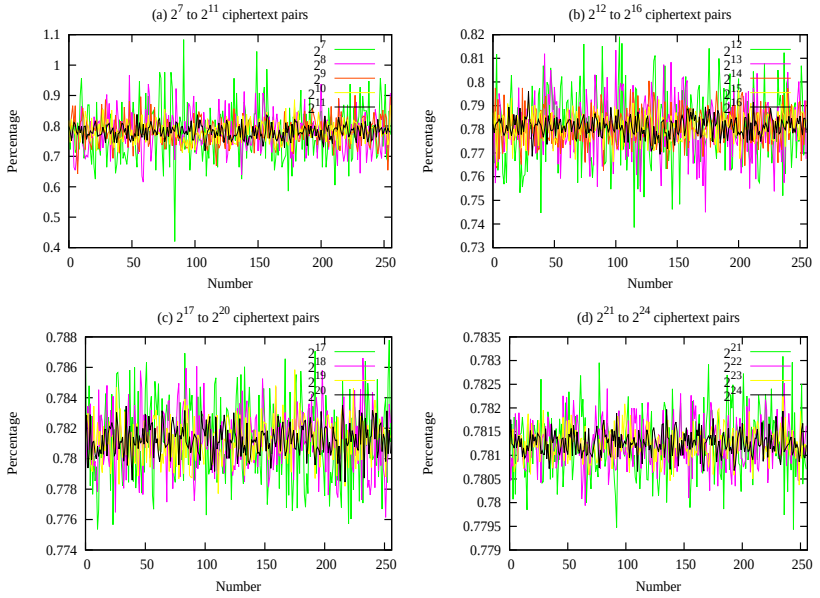

Figure D.80: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 10-round AES-128 for 10 trials (Autofeeding)

E Experimental Test Results for KASUMI

Table E.1: Experimental Results of SAC Tests on Various Rounds of KASUMI

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	63	0.005	95.649	133.121	512($i.e.$ 2^9)	Fail	Fig.E.1
2	63	0.005	95.649	132.195	512($i.e.$ 2^9)	Fail	Fig.E.9
3	63	0.005	95.649	38.5322	16777216($i.e.$ 2^{24})	Pass	Fig.E.17
4	63	0.005	95.649	39.8434	16777216($i.e.$ 2^{24})	Pass	Fig.E.25
5	63	0.005	95.649	35.7763	16777216($i.e.$ 2^{24})	Pass	Fig.E.33
6	63	0.005	95.649	38.8697	16777216($i.e.$ 2^{24})	Pass	Fig.E.41
7	63	0.005	95.649	38.1706	16777216($i.e.$ 2^{24})	Pass	Fig.E.49
8	63	0.005	95.649	35.1613	16777216($i.e.$ 2^{24})	Pass	Fig.E.57

Table E.2: Experimental Results of modSAC Tests for 2-bit substring on Various Rounds of KASUMI

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	3	0.005	12.838	680.203	128($i.e.$ 2^7)	Fail	Fig.E.2
2	3	0.005	12.838	694.815	128($i.e.$ 2^7)	Fail	Fig.E.10
3	3	0.005	12.838	1.44081	16777216($i.e.$ 2^{24})	Pass	Fig.E.18
4	3	0.005	12.838	3.38654	16777216($i.e.$ 2^{24})	Pass	Fig.E.26
5	3	0.005	12.838	2.1743	16777216($i.e.$ 2^{24})	Pass	Fig.E.34
6	3	0.005	12.838	3.83326	16777216($i.e.$ 2^{24})	Pass	Fig.E.42
7	3	0.005	12.838	3.87644	16777216($i.e.$ 2^{24})	Pass	Fig.E.50
8	3	0.005	12.838	2.73519	16777216($i.e.$ 2^{24})	Pass	Fig.E.58

Table E.3: Experimental Results of modSAC Tests for 4-bit substring on Various Rounds of KASUMI

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	15	0.005	32.801	1511.62	128($i.e.$ 2^7)	Fail	Fig.E.3
2	15	0.005	32.801	1531.89	128($i.e.$ 2^7)	Fail	Fig.E.11
3	15	0.005	32.801	15.9325	16777216($i.e.$ 2^{24})	Pass	Fig.E.19
4	15	0.005	32.801	18.0409	16777216($i.e.$ 2^{24})	Pass	Fig.E.27
5	15	0.005	32.801	12.368	16777216($i.e.$ 2^{24})	Pass	Fig.E.35
6	15	0.005	32.801	15.7602	16777216($i.e.$ 2^{24})	Pass	Fig.E.43
7	15	0.005	32.801	14.3777	16777216($i.e.$ 2^{24})	Pass	Fig.E.51
8	15	0.005	32.801	14.5424	16777216($i.e.$ 2^{24})	Pass	Fig.E.59

Table E.4: Experimental Results of modSAC Tests for 8-bit substring on Various Rounds of KASUMI

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	255	0.005	316.919	18976.9	256($i.e.$ 2^8)	Fail	Fig.E.4
2	255	0.005	316.919	19204.1	256($i.e.$ 2^8)	Fail	Fig.E.12
3	255	0.005	316.919	270.829	16777216($i.e.$ 2^{24})	Pass	Fig.E.20
4	255	0.005	316.919	258.034	16777216($i.e.$ 2^{24})	Pass	Fig.E.28
5	255	0.005	316.919	246.686	16777216($i.e.$ 2^{24})	Pass	Fig.E.36
6	255	0.005	316.919	250.273	16777216($i.e.$ 2^{24})	Pass	Fig.E.44
7	255	0.005	316.919	256.778	16777216($i.e.$ 2^{24})	Pass	Fig.E.52
8	255	0.005	316.919	261.773	16777216($i.e.$ 2^{24})	Pass	Fig.E.60

Table E.5: Experimental Results of SAC Tests on Various Rounds of KASUMI (Autofeeding)

No. of Rounds	d.o.f	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	63	0.005	95.649	132.992	512($i.e.$ 2^9)	Fail	Fig.E.5
2	63	0.005	95.649	133.965	512($i.e.$ 2^9)	Pass	Fig.E.13
3	63	0.005	95.649	37.9476	16777216($i.e.$ 2^{24})	Pass	Fig.E.21
4	63	0.005	95.649	36.2361	16777216($i.e.$ 2^{24})	Pass	Fig.E.29
5	63	0.005	95.649	42.0592	16777216($i.e.$ 2^{24})	Pass	Fig.E.37
6	63	0.005	95.649	40.7559	16777216($i.e.$ 2^{24})	Pass	Fig.E.45
7	63	0.005	95.649	37.052	16777216($i.e.$ 2^{24})	Pass	Fig.E.53
8	63	0.005	95.649	35.1955	16777216($i.e.$ 2^{24})	Pass	Fig.E.61

Table E.6: Experimental Results of modSAC Tests for 2-bit substring on Various Rounds of KASUMI (Autofeeding)

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	3	0.005	12.838	663.666	128($i.e.$ 2^7)	Fail	Fig.E.6
2	3	0.005	12.838	685.067	128($i.e.$ 2^7)	Fail	Fig.E.14
3	3	0.005	12.838	2.86048	16777216($i.e.$ 2^{24})	Pass	Fig.E.22
4	3	0.005	12.838	2.36669	16777216($i.e.$ 2^{24})	Pass	Fig.E.30
5	3	0.005	12.838	2.45476	16777216($i.e.$ 2^{24})	Pass	Fig.E.38
6	3	0.005	12.838	4.2658	16777216($i.e.$ 2^{24})	Pass	Fig.E.46
7	3	0.005	12.838	2.45983	16777216($i.e.$ 2^{24})	Pass	Fig.E.54
8	3	0.005	12.838	2.31681	16777216($i.e.$ 2^{24})	Pass	Fig.E.62

Table E.7: Experimental Results of modSAC Tests for 4-bit substring on Various Rounds of KASUMI (Autofeeding)

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	15	0.005	32.801	1484.09	128($i.e.$ 2^7)	Fail	Fig.E.7
2	15	0.005	32.801	1522.55	128($i.e.$ 2^7)	Fail	Fig.E.15
3	15	0.005	32.801	15.3465	16777216($i.e.$ 2^{24})	Pass	Fig.E.23
4	15	0.005	32.801	14.3438	16777216($i.e.$ 2^{24})	Pass	Fig.E.31
5	15	0.005	32.801	13.4412	16777216($i.e.$ 2^{24})	Pass	Fig.E.39
6	15	0.005	32.801	16.2691	16777216($i.e.$ 2^{24})	Pass	Fig.E.47
7	15	0.005	32.801	14.6663	16777216($i.e.$ 2^{24})	Pass	Fig.E.55
8	15	0.005	32.801	14.3111	16777216($i.e.$ 2^{24})	Pass	Fig.E.63

Table E.8: Experimental Results of modSAC Tests for 8-bit substring on Various Rounds of KASUMI (Autofeeding)

No. of Rounds	degrees of freedom	p-value	χ^2 value	Experimental χ^2 value	No. of ciphertext pairs	Test Result	*Figure No. in this thesis*
1	255	0.005	316.919	19073.4	256($i.e.$ 2^8)	Fail	Fig.E.8
2	255	0.005	316.919	19099.3	256($i.e.$ 2^8)	Fail	Fig.E.16
3	255	0.005	316.919	253.827	16777216($i.e.$ 2^{24})	Pass	Fig.E.24
4	255	0.005	316.919	240.434	16777216($i.e.$ 2^{24})	Pass	Fig.E.32
5	255	0.005	316.919	246.118	16777216($i.e.$ 2^{24})	Pass	Fig.E.40
6	255	0.005	316.919	263.1	16777216($i.e.$ 2^{24})	Pass	Fig.E.48
7	255	0.005	316.919	250.015	16777216($i.e.$ 2^{24})	Pass	Fig.E.56
8	255	0.005	316.919	247.565	16777216($i.e.$ 2^{24})	Pass	Fig.E.64

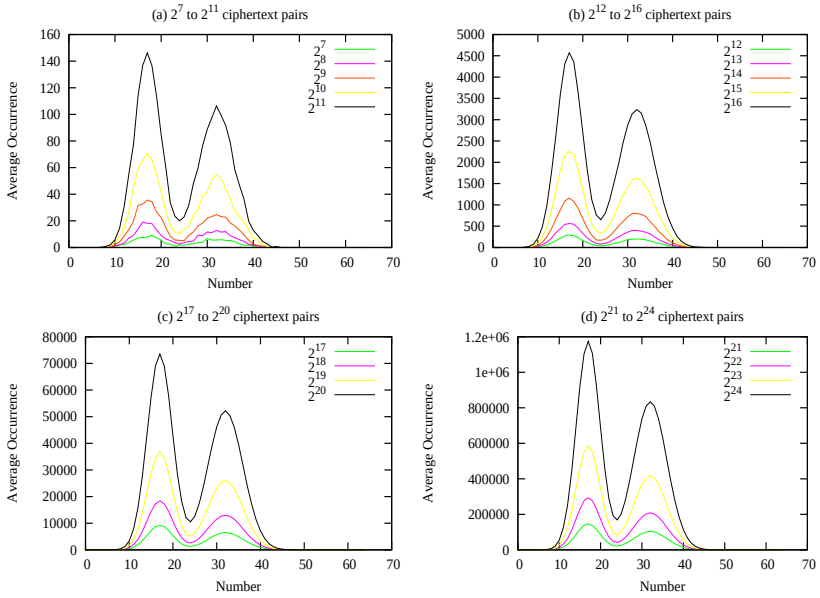

Figure E.1: Average Hamming Weight Distribution between ciphertext pairs for 1-round KASUMI for 10 trials

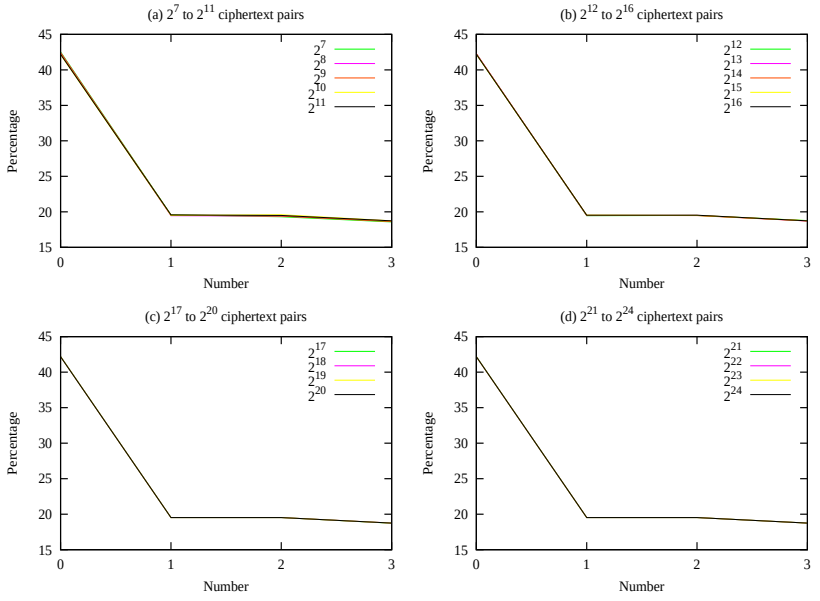

Figure E.2: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 1-round KASUMI for 10 trials

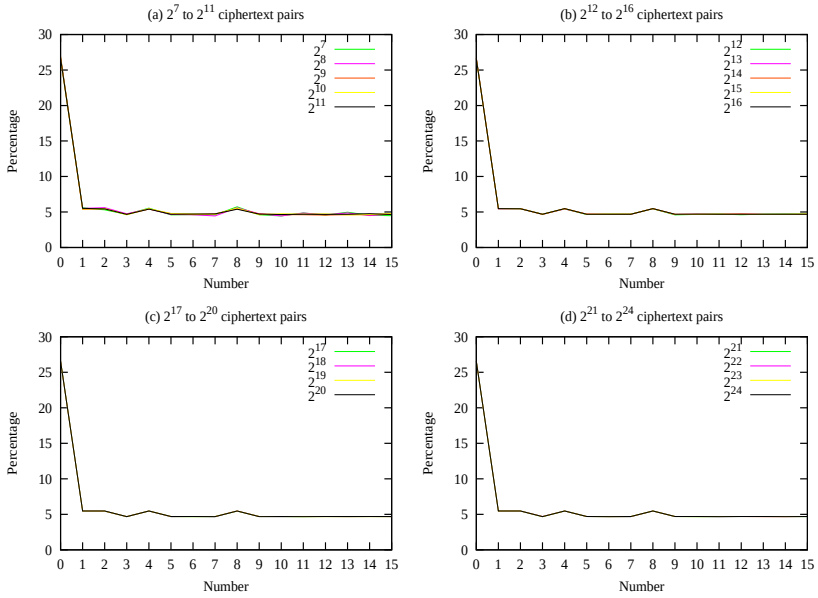

Figure E.3: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 1-round KASUMI for 10 trials

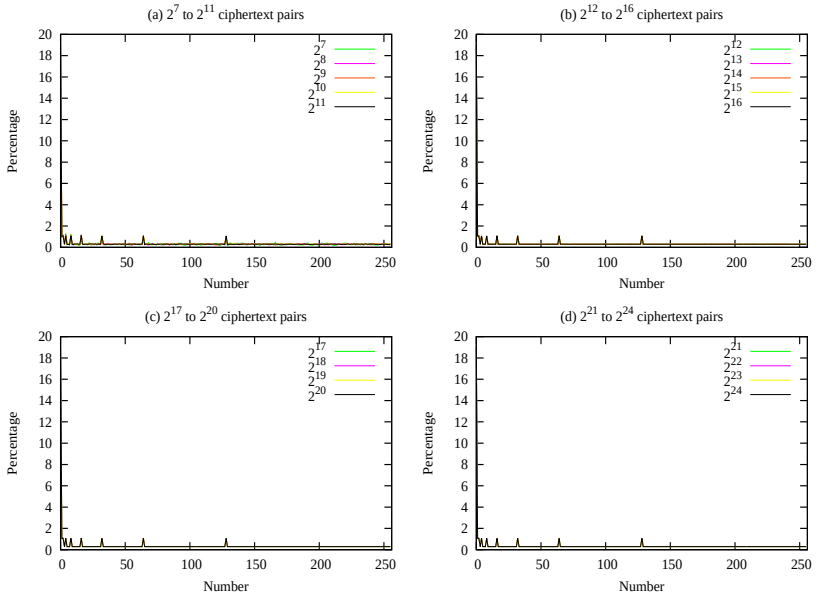

Figure E.4: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 1-round KASUMI for 10 trials

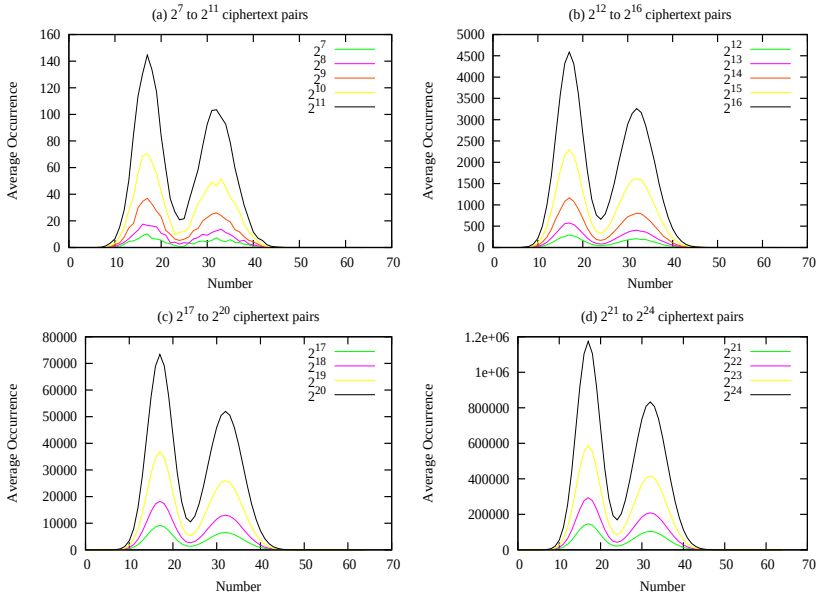

Figure E.5: Average Hamming Weight Distribution between ciphertext pairs for 1-round KASUMI for 10 trials (Autofeeding)

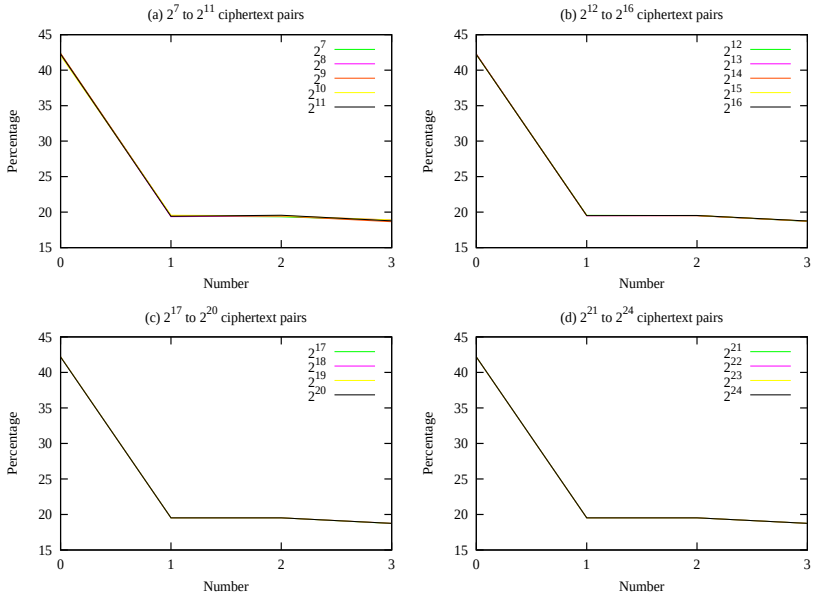

Figure E.6: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 1-round KASUMI for 10 trials (Autofeeding)

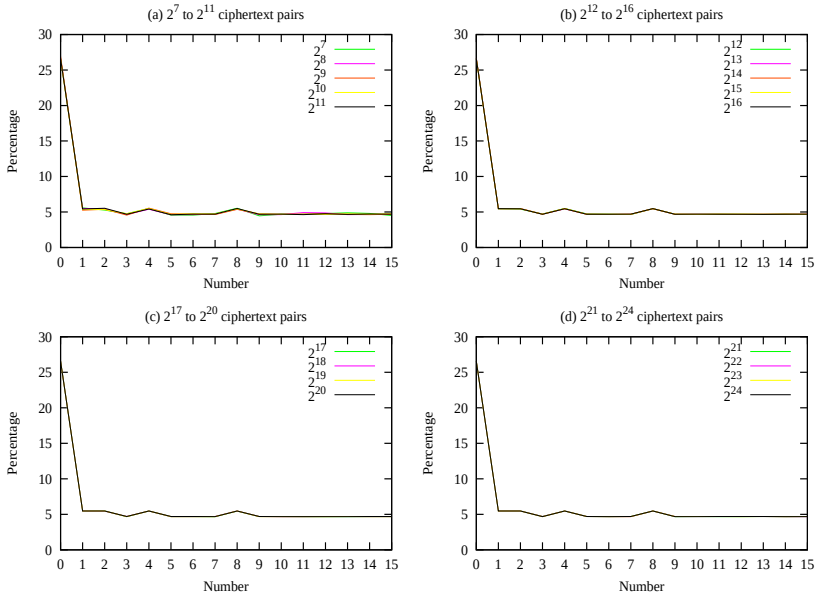

Figure E.7: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 1-round KASUMI for 10 trials (Autofeeding)

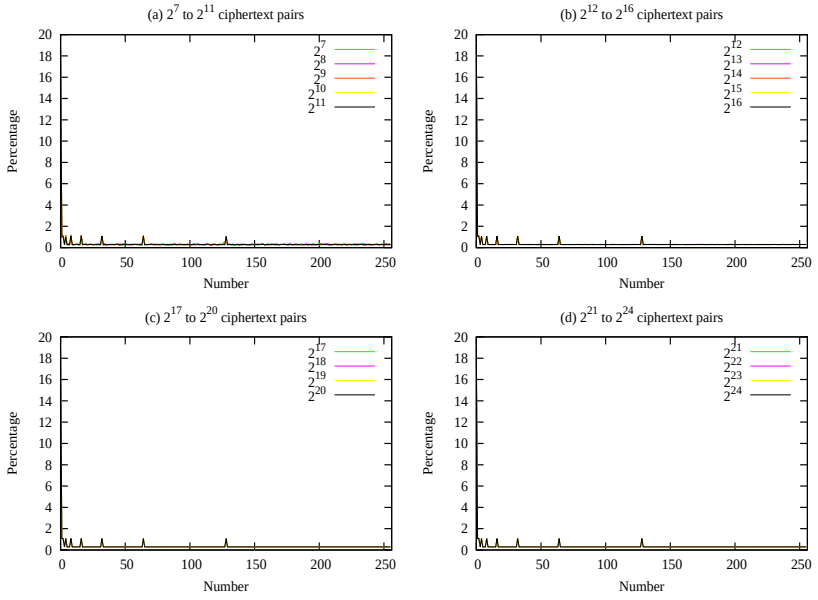

Figure E.8: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 1-round KASUMI for 10 trials (Autofeeding)

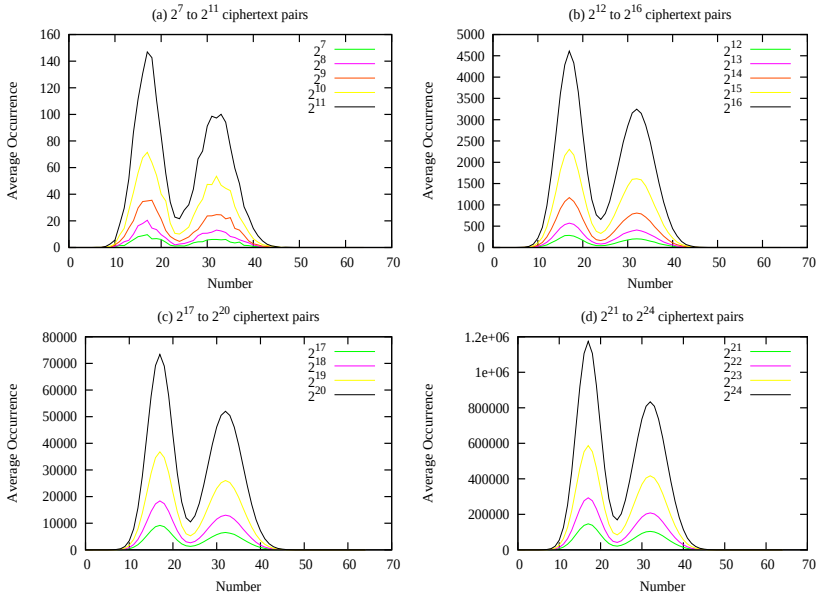

Figure E.9: Average Hamming Weight Distribution between ciphertext pairs for 2-round KASUMI for 10 trials

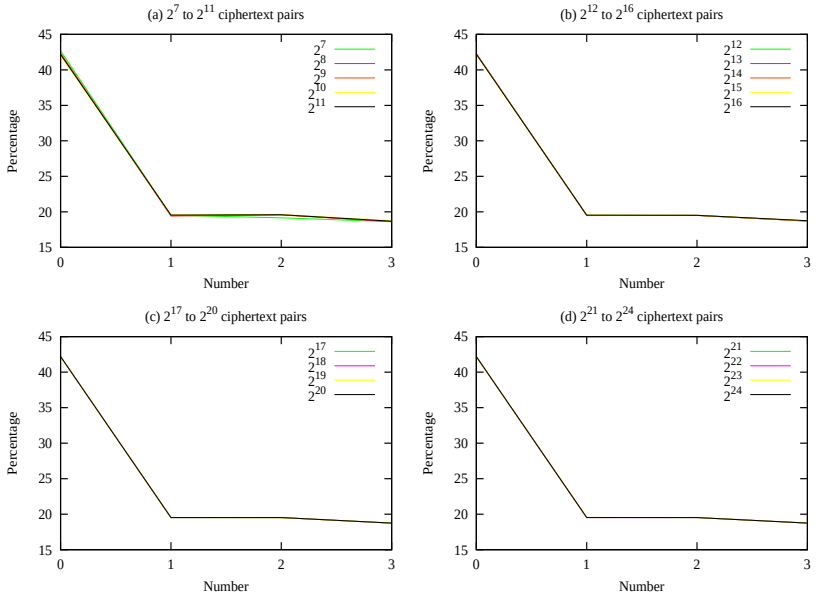

Figure E.10: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 2-round KASUMI for 10 trials

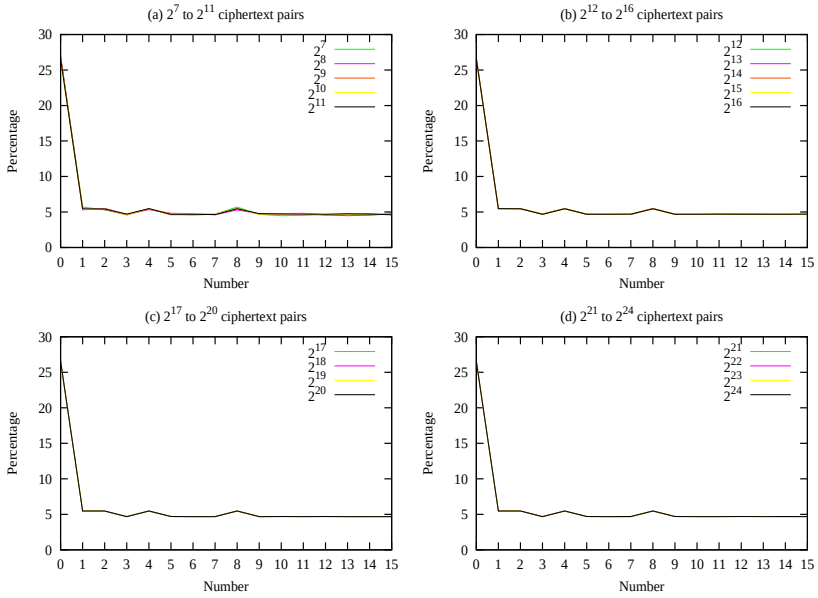

Figure E.11: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 2-round KASUMI for 10 trials

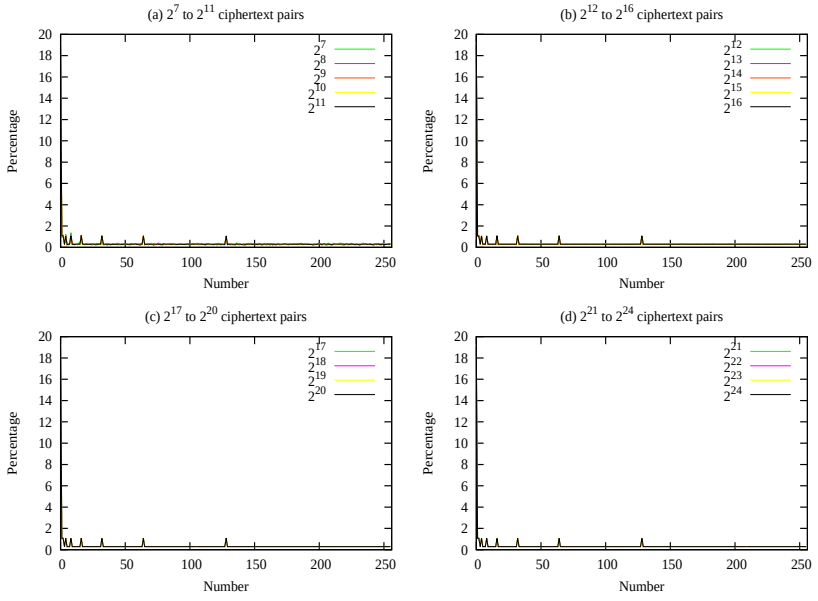

Figure E.12: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 2-round KASUMI for 10 trials

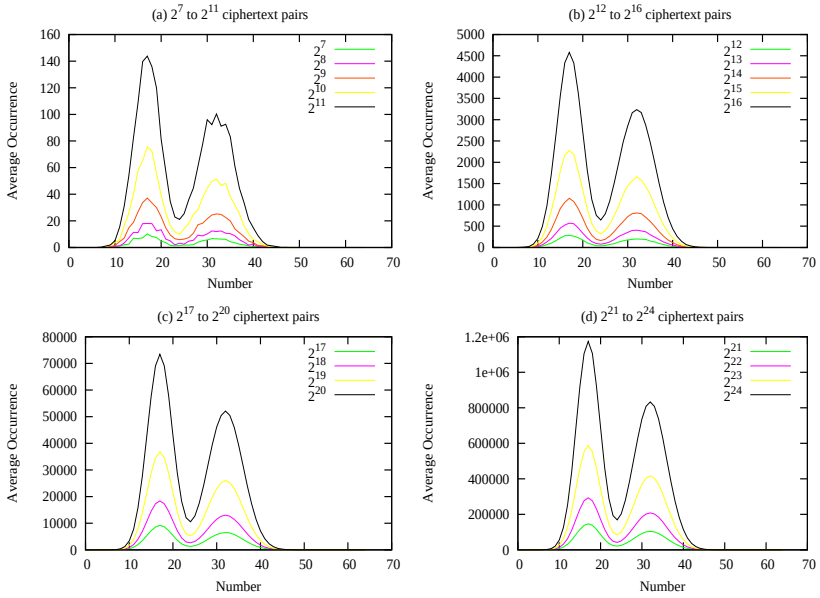

Figure E.13: Average Hamming Weight Distribution between ciphertext pairs for 2-round KASUMI for 10 trials (Autofeeding)

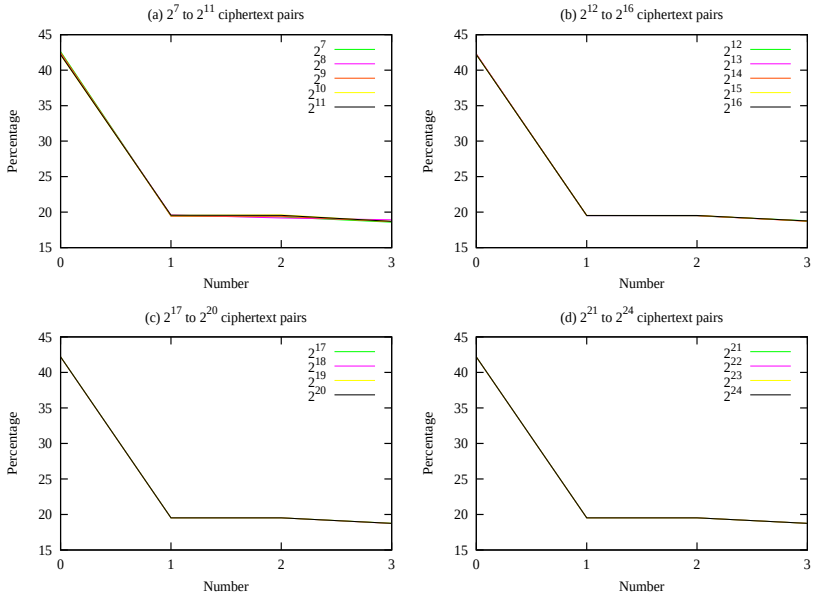

Figure E.14: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 2-round KASUMI for 10 trials (Autofeeding)

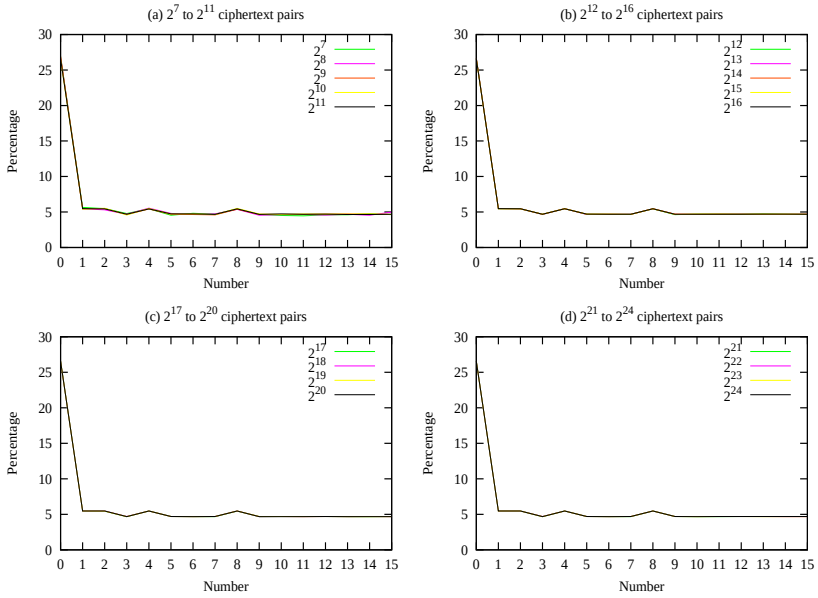

Figure E.15: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 2-round KASUMI for 10 trials (Autofeeding)

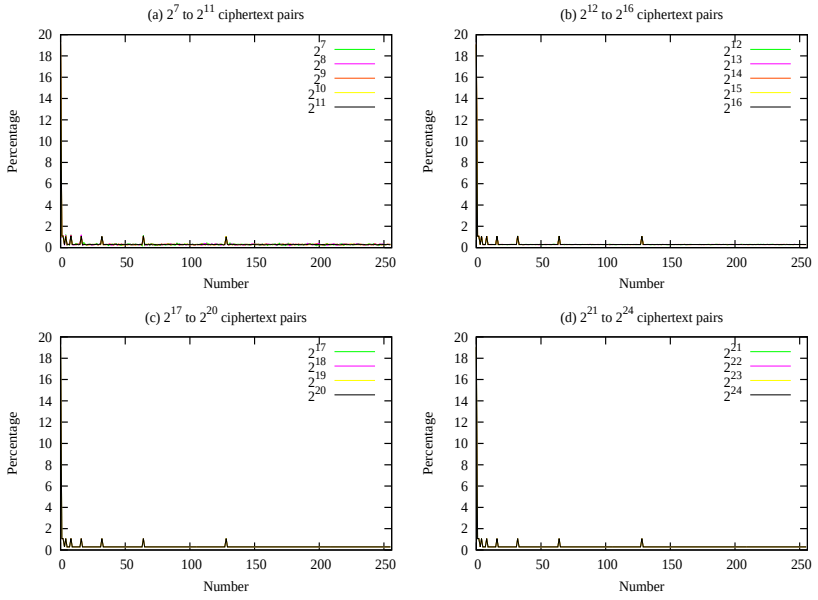

Figure E.16: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 2-round KASUMI for 10 trials (Autofeeding)

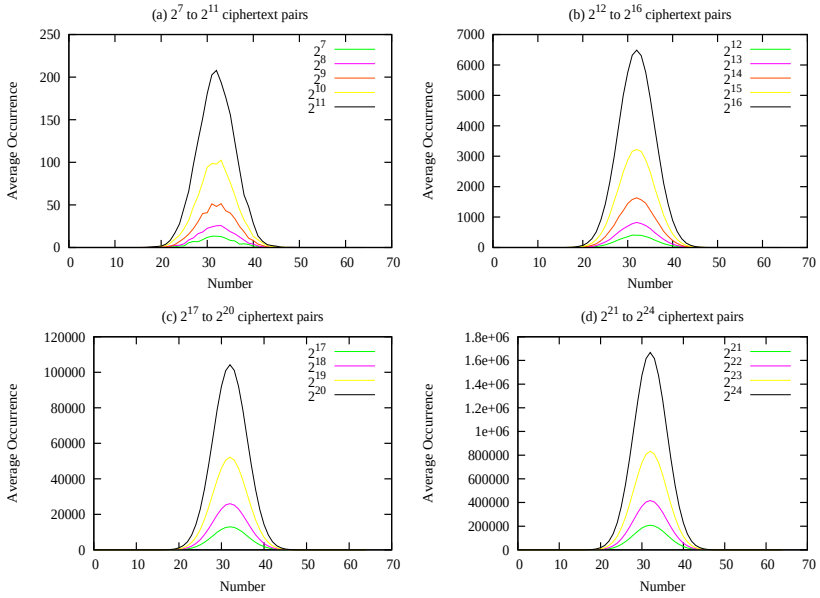

Figure E.17: Average Hamming Weight Distribution between ciphertext pairs for 3-round KASUMI for 10 trials

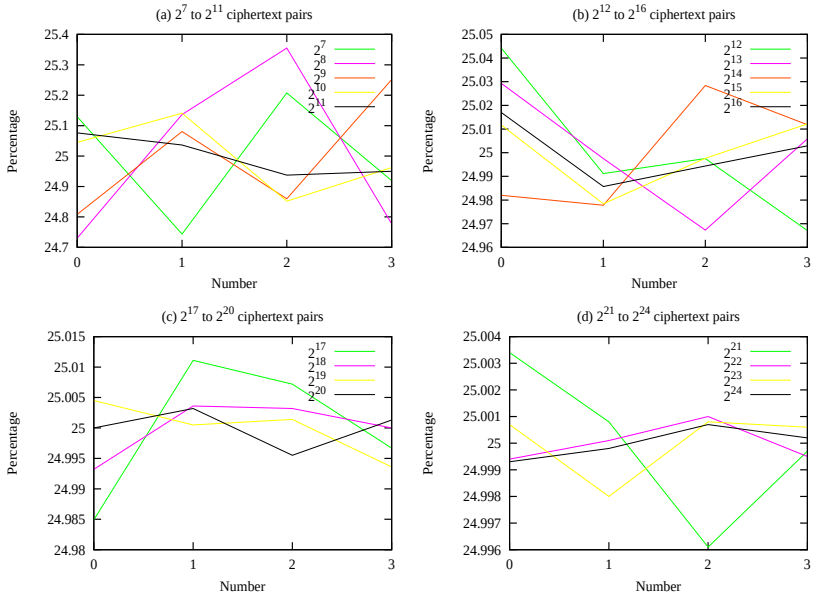

Figure E.18: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 3-round KASUMI for 10 trials

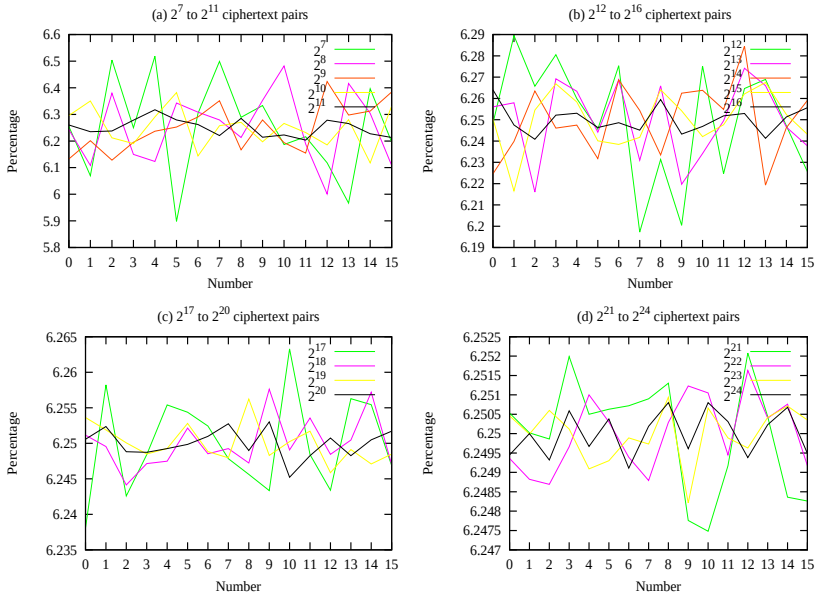

Figure E.19: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 3-round KASUMI for 10 trials

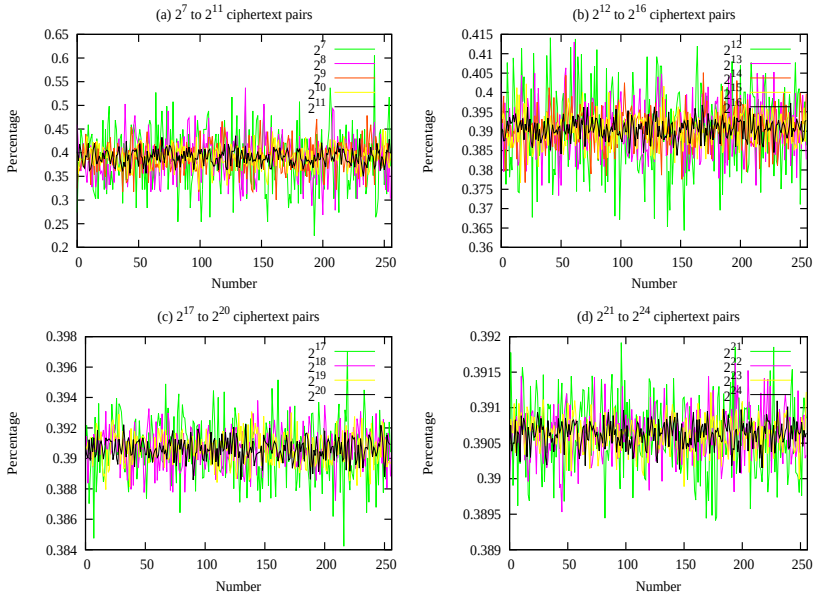

Figure E.20: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 3-round KASUMI for 10 trials

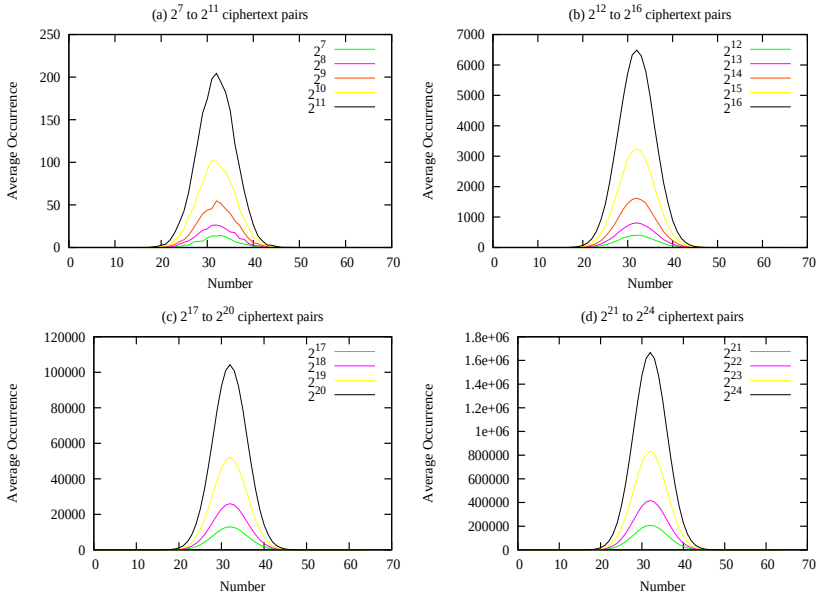

Figure E.21: Average Hamming Weight Distribution between ciphertext pairs for 3-round KASUMI for 10 trials (Autofeeding)

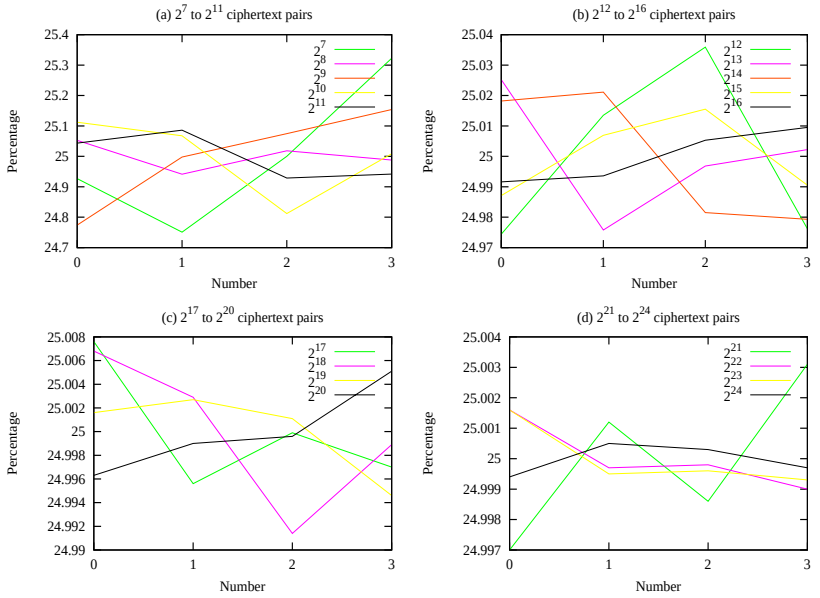

Figure E.22: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 3-round KASUMI for 10 trials (Autofeeding)

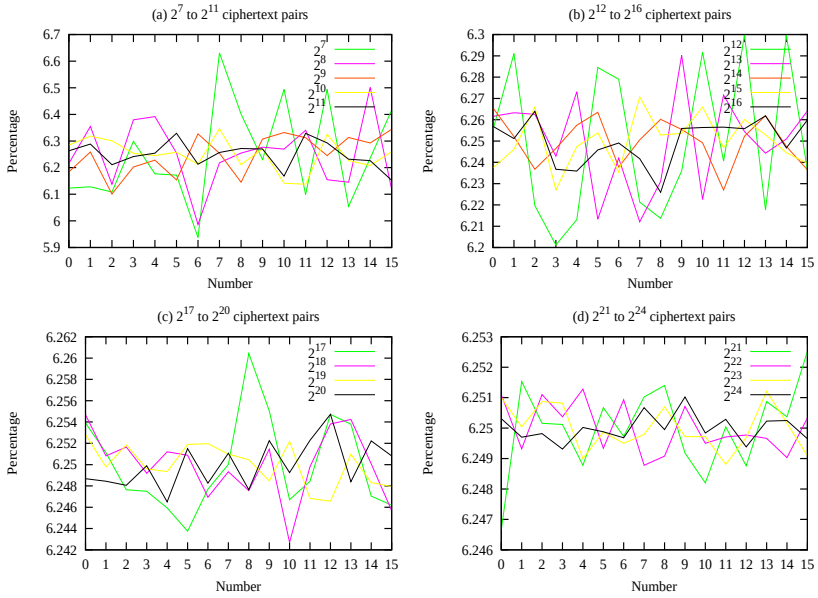

Figure E.23: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 3-round KASUMI for 10 trials (Autofeeding)

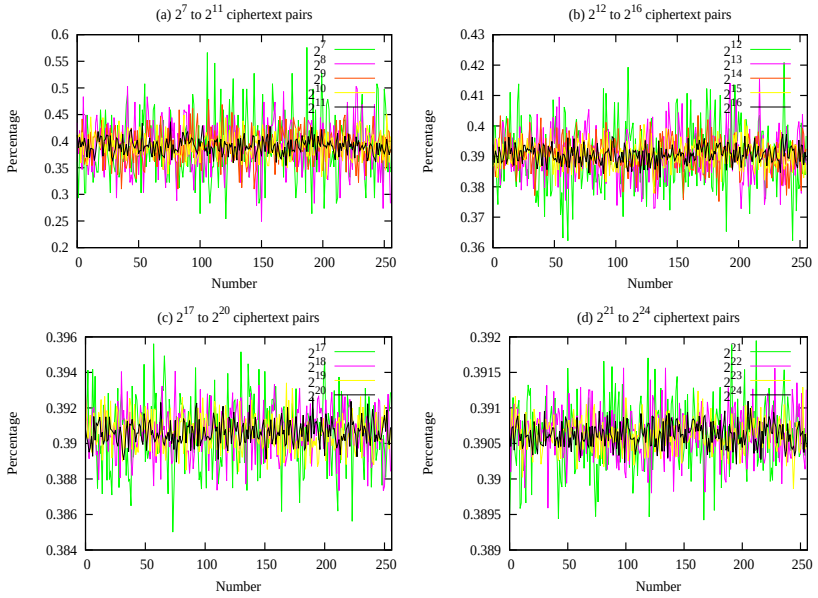

Figure E.24: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 3-round KASUMI for 10 trials (Autofeeding)

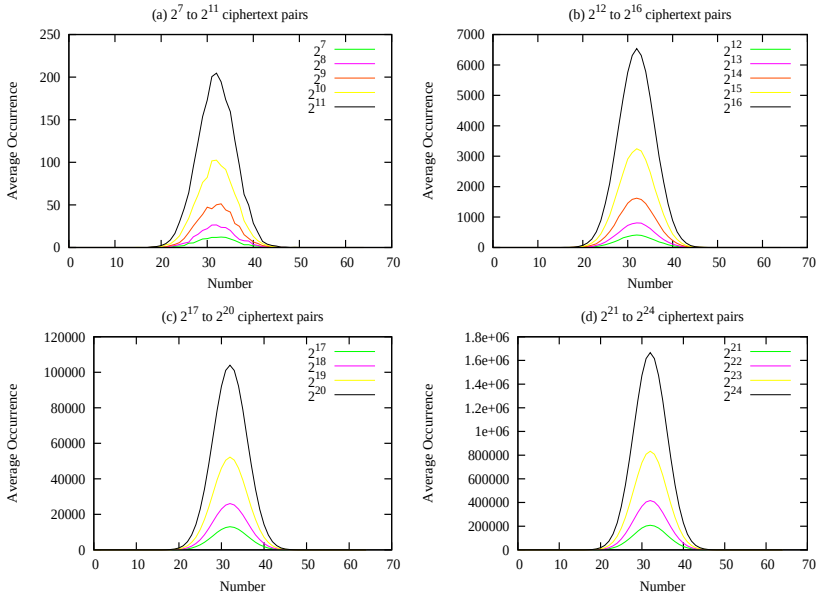

Figure E.25: Average Hamming Weight Distribution between ciphertext pairs for 4-round KASUMI for 10 trials

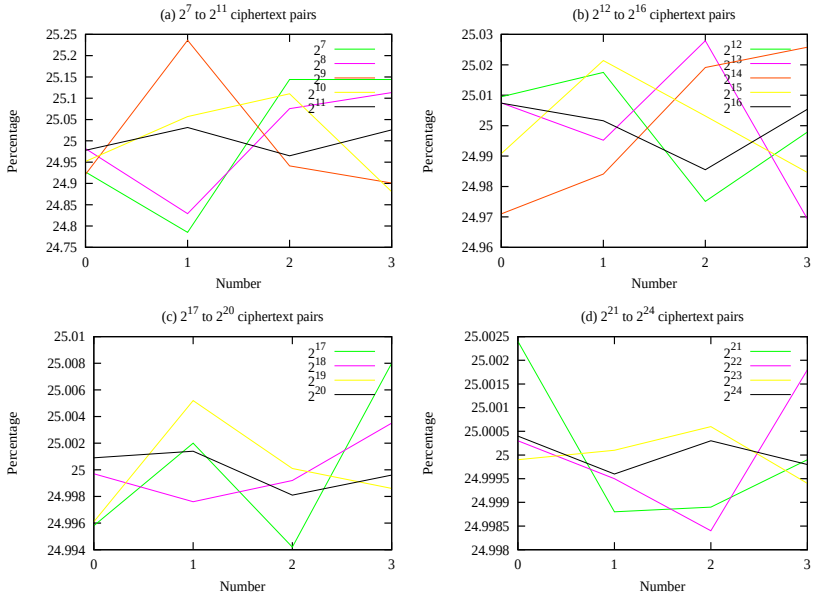

Figure E.26: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 4-round KASUMI for 10 trials

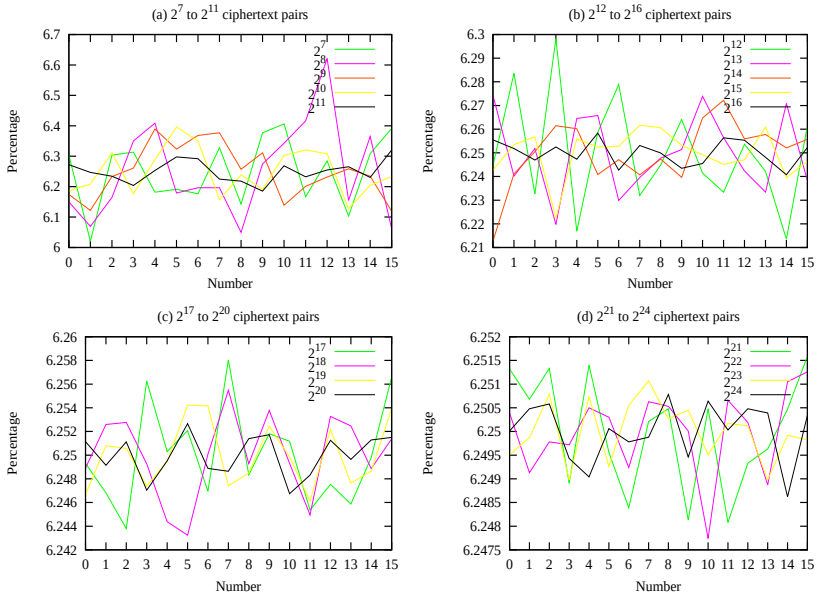

Figure E.27: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 4-round KASUMI for 10 trials

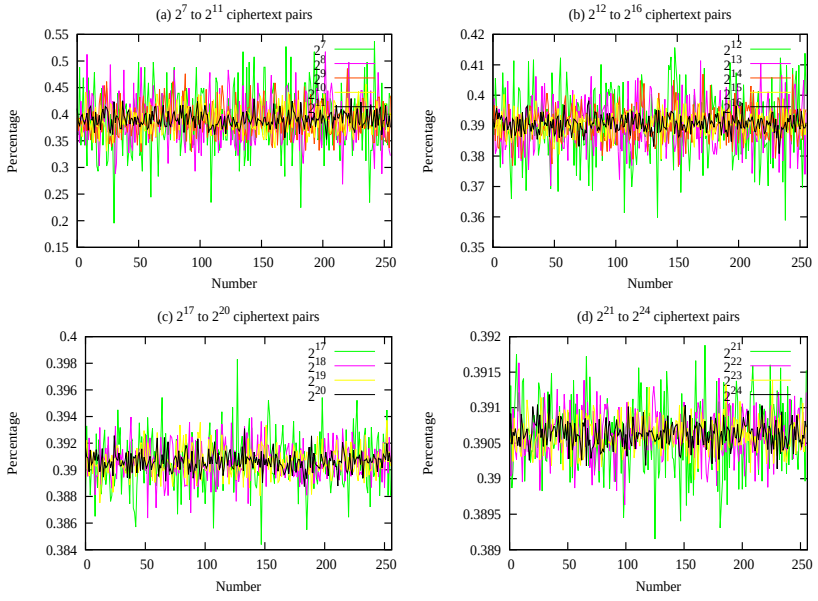

Figure E.28: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 4-round KASUMI for 10 trials

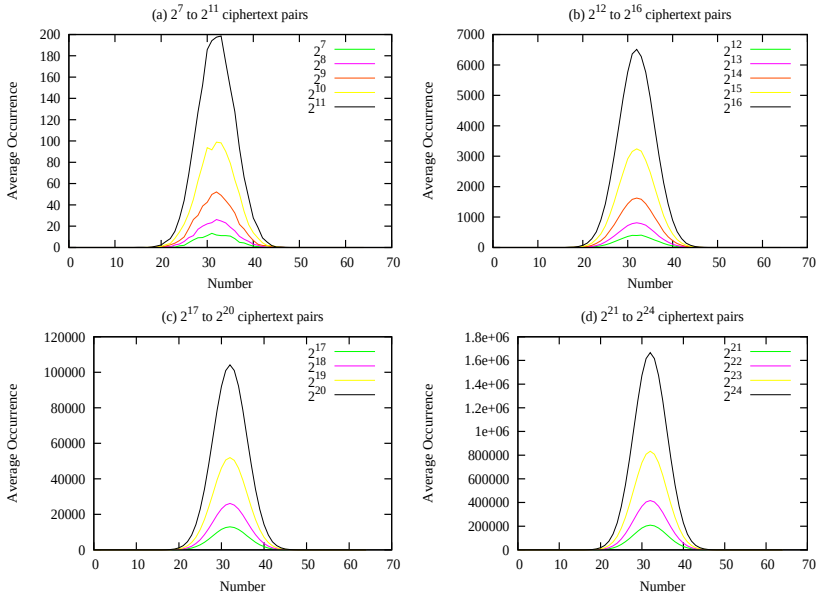

Figure E.29: Average Hamming Weight Distribution between ciphertext pairs for 4-round KASUMI for 10 trials (Autofeeding)

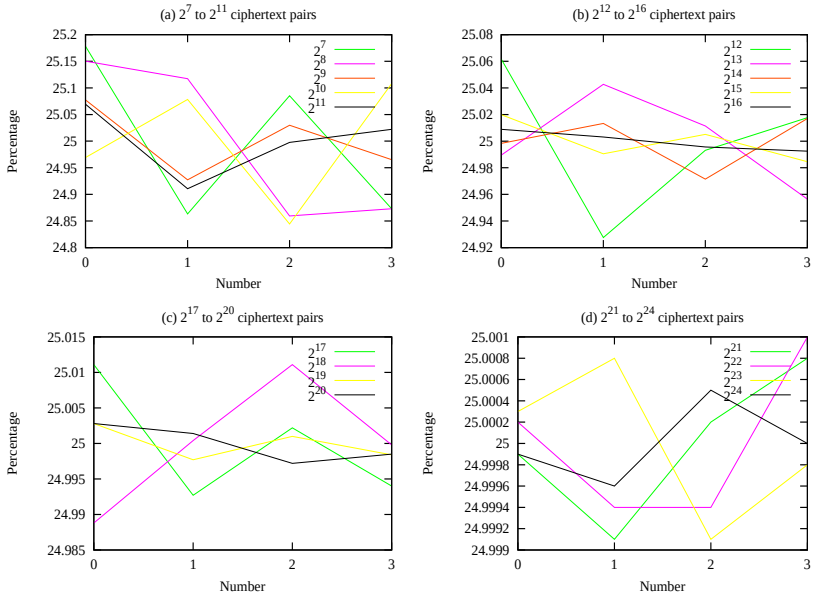

Figure E.30: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 4-round KASUMI for 10 trials (Autofeeding)

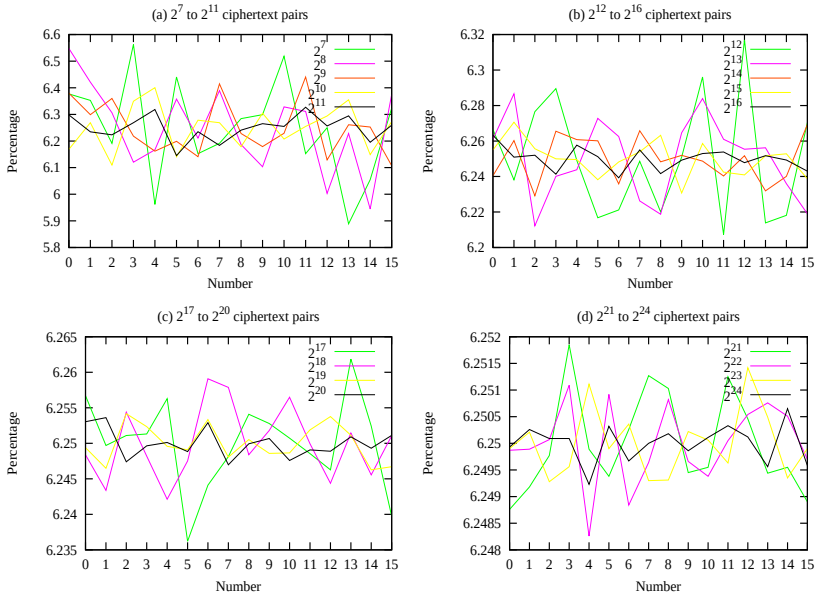

Figure E.31: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 4-round KASUMI for 10 trials (Autofeeding)

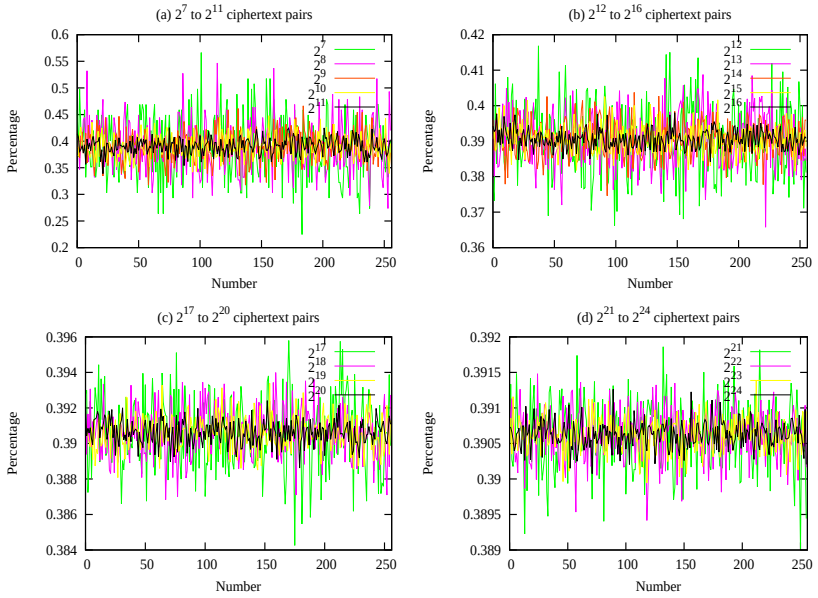

Figure E.32: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 4-round KASUMI for 10 trials (Autofeeding)

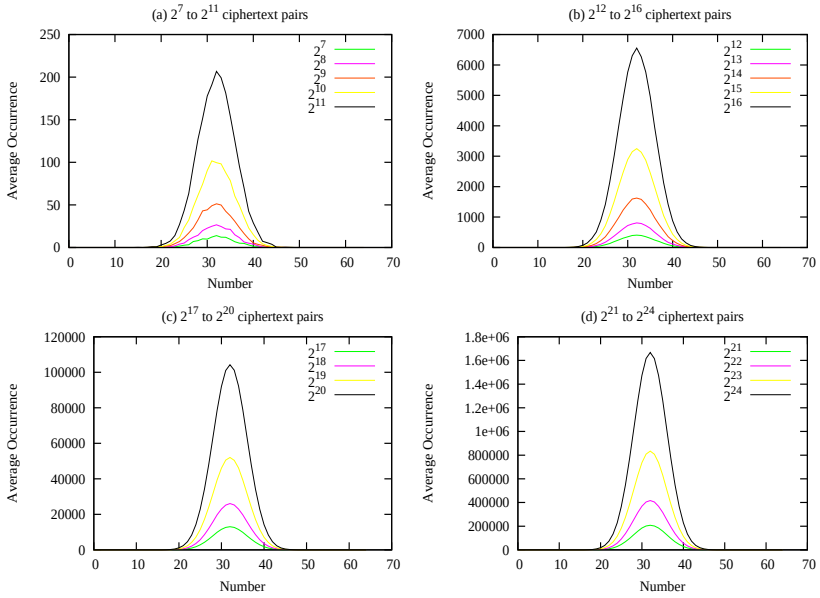

Figure E.33: Average Hamming Weight Distribution between ciphertext pairs for 5-round KASUMI for 10 trials

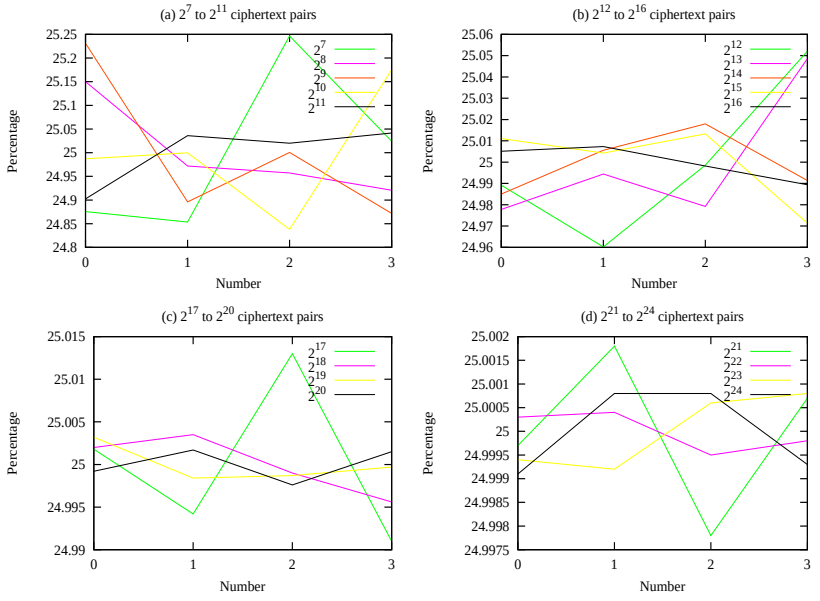

Figure E.34: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 5-round KASUMI for 10 trials

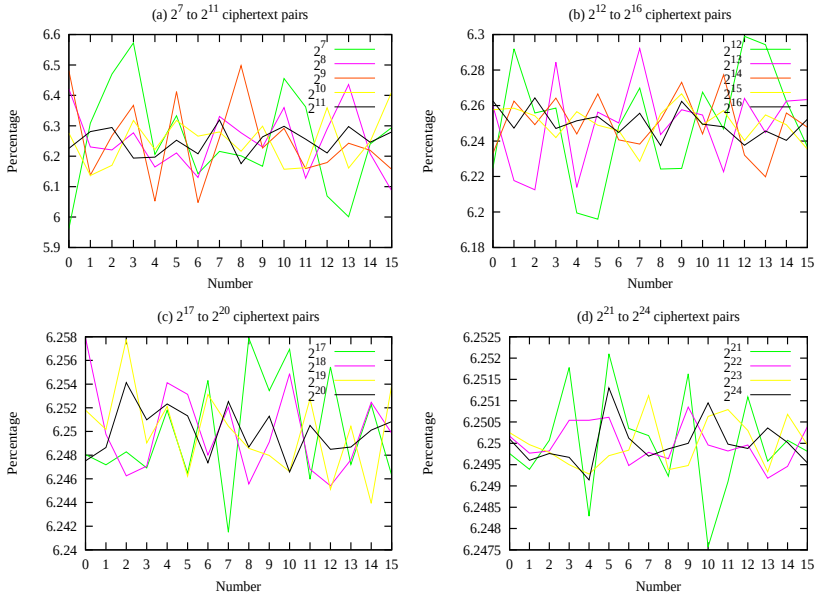

Figure E.35: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 5-round KASUMI for 10 trials

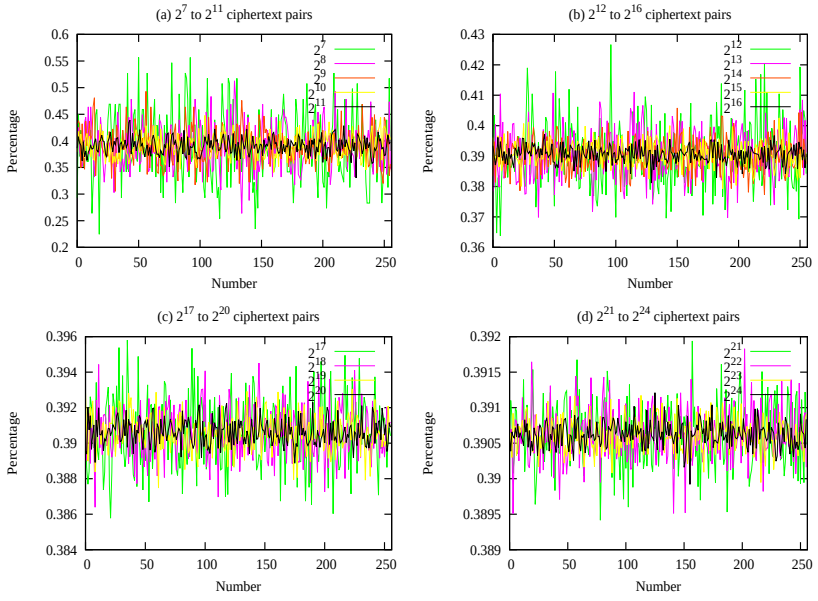

Figure E.36: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 5-round KASUMI for 10 trials

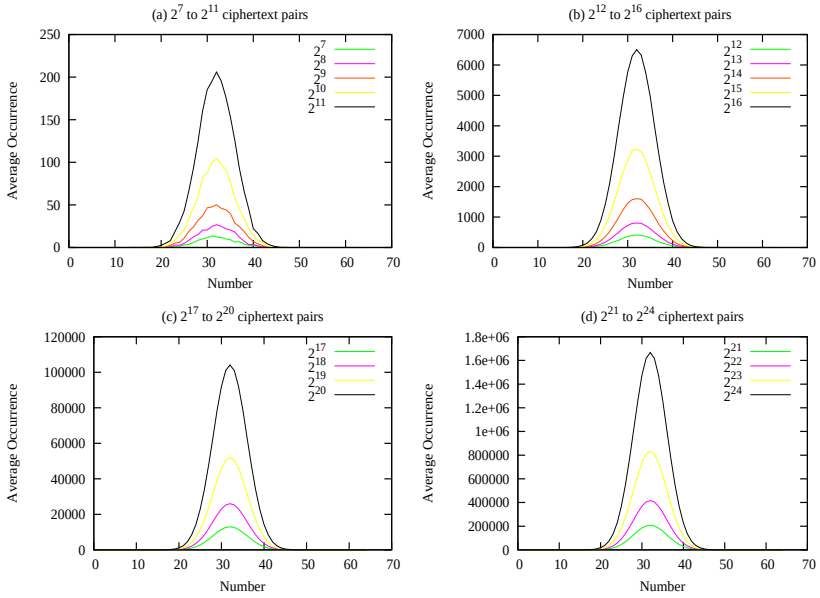

Figure E.37: Average Hamming Weight Distribution between ciphertext pairs for 5-round KASUMI for 10 trials (Autofeeding)

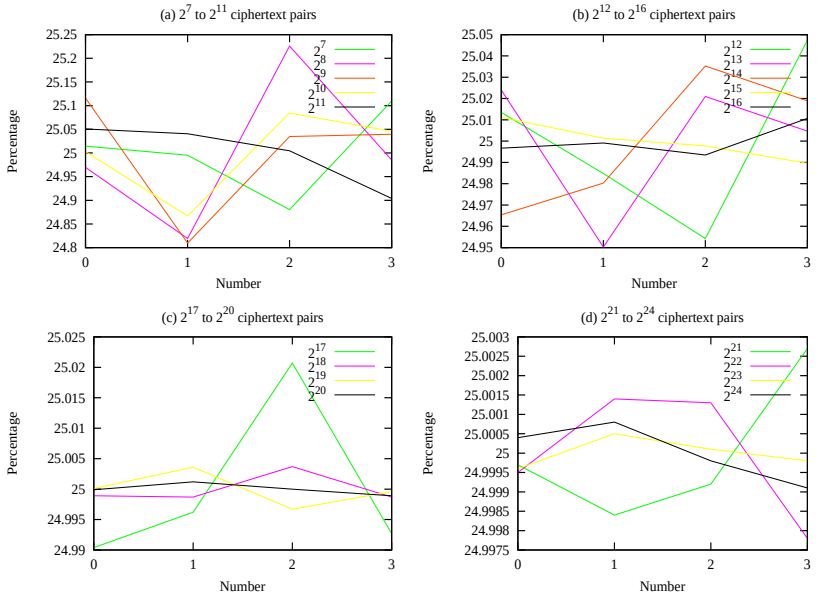

Figure E.38: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 5-round KASUMI for 10 trials (Autofeeding)

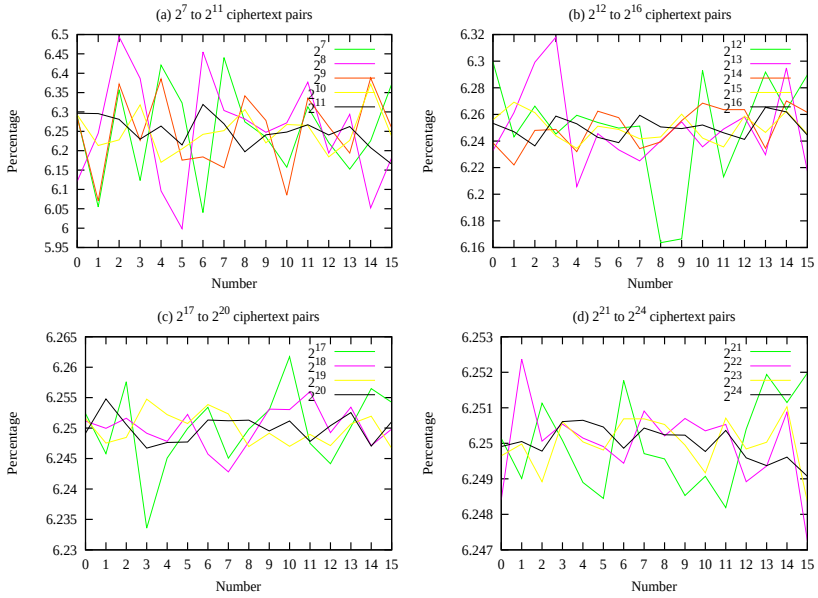

Figure E.39: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 5-round KASUMI for 10 trials (Autofeeding)

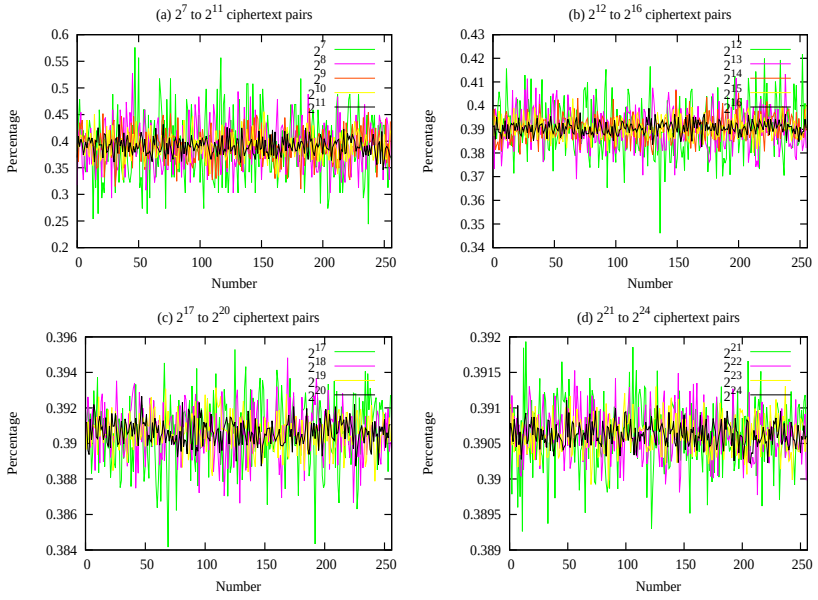

Figure E.40: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 5-round KASUMI for 10 trials (Autofeeding)

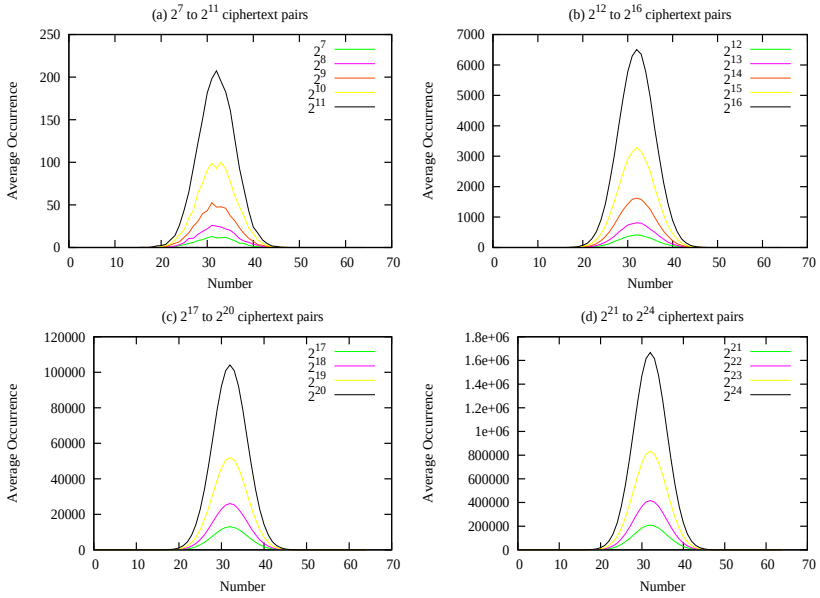

Figure E.41: Average Hamming Weight Distribution between ciphertext pairs for 6-round KASUMI for 10 trials

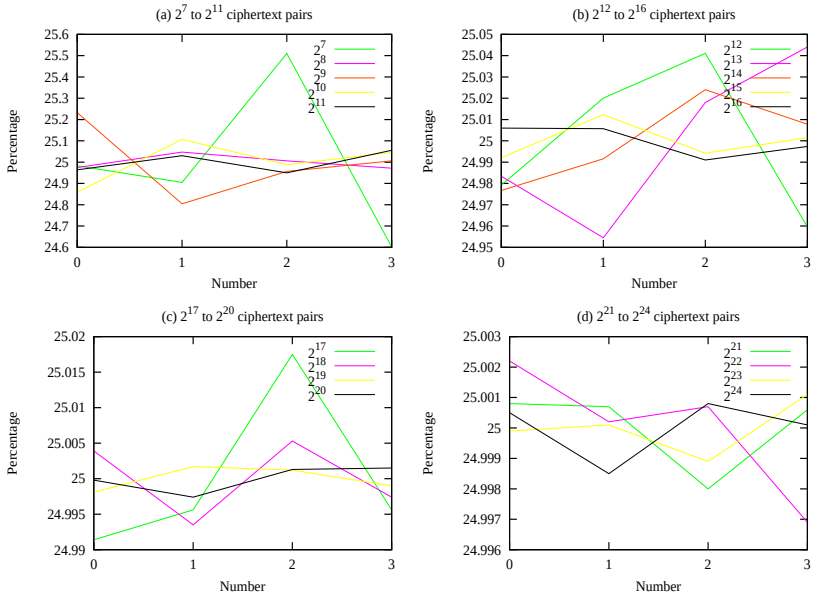

Figure E.42: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 6-round KASUMI for 10 trials

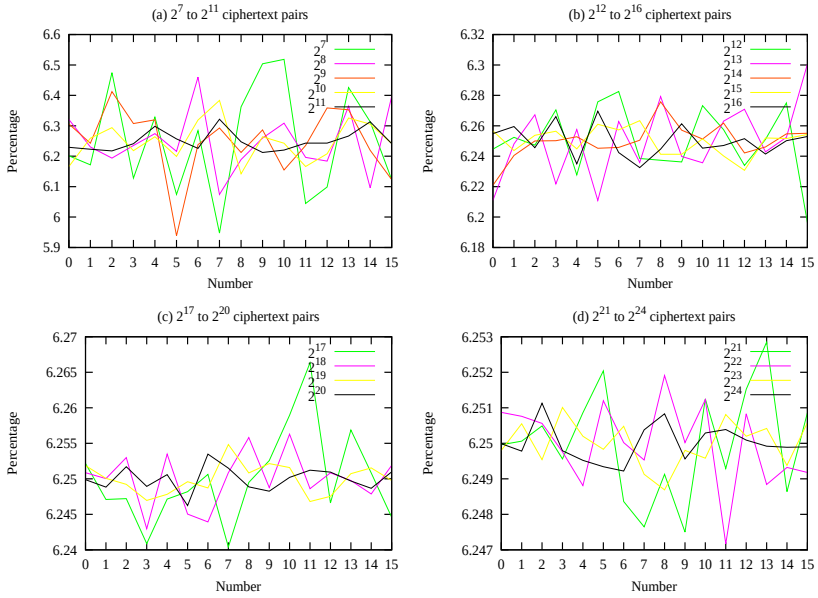

Figure E.43: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 6-round KASUMI for 10 trials

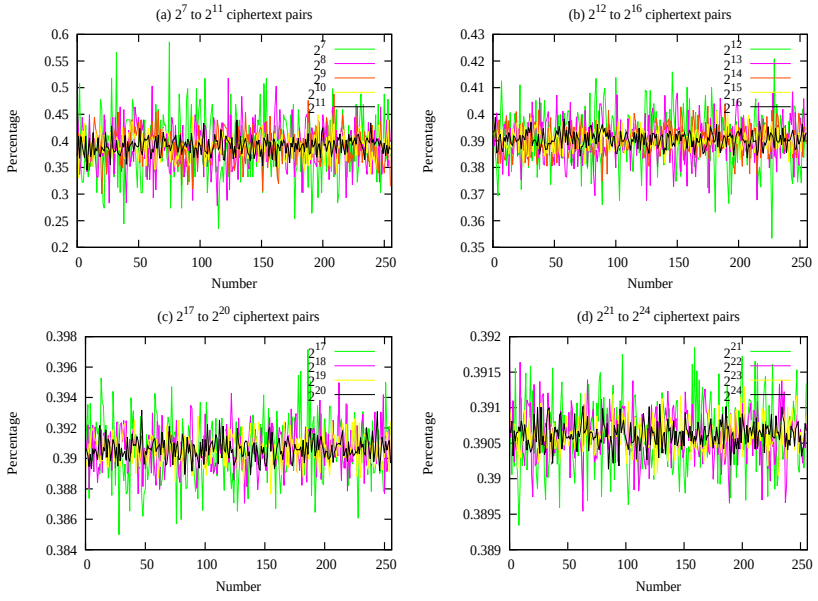

Figure E.44: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 6-round KASUMI for 10 trials

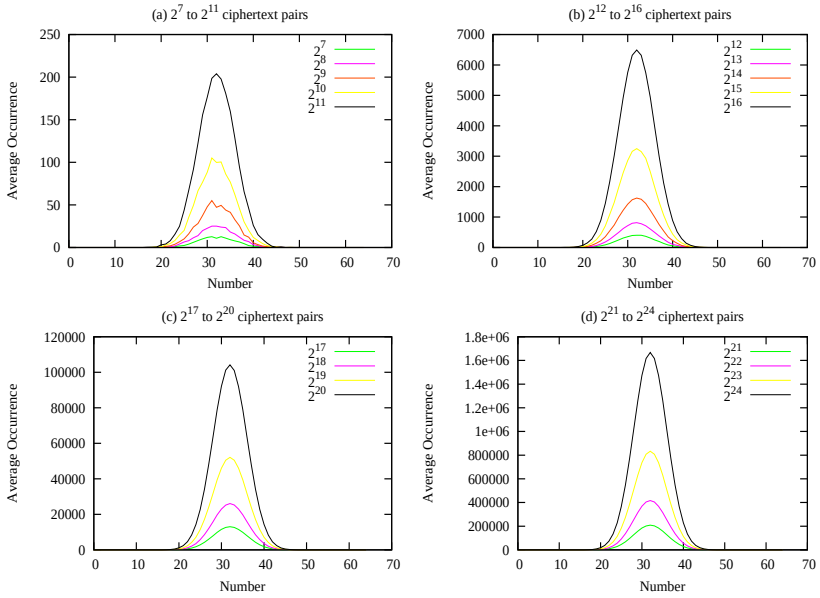

Figure E.45: Average Hamming Weight Distribution between ciphertext pairs for 6-round KASUMI for 10 trials (Autofeeding)

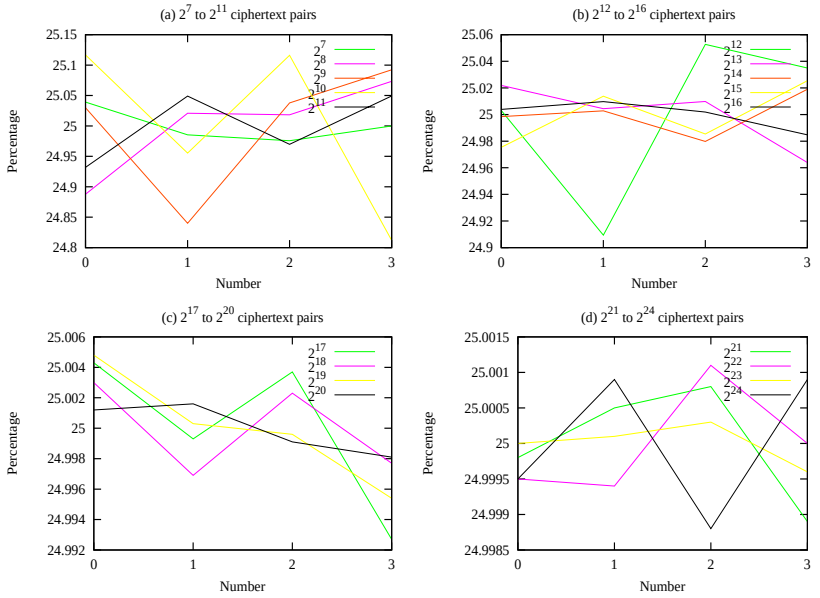

Figure E.46: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 6-round KASUMI for 10 trials (Autofeeding)

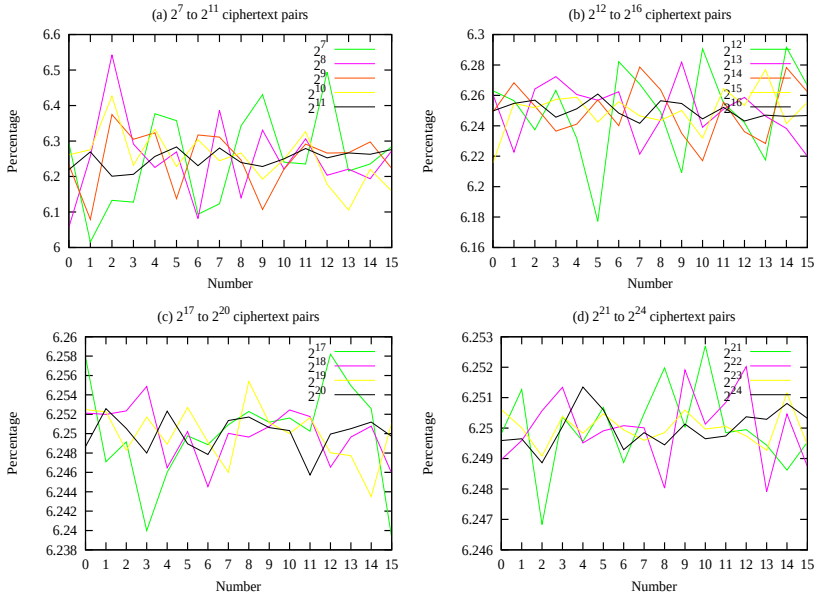

Figure E.47: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 6-round KASUMI for 10 trials (Autofeeding)

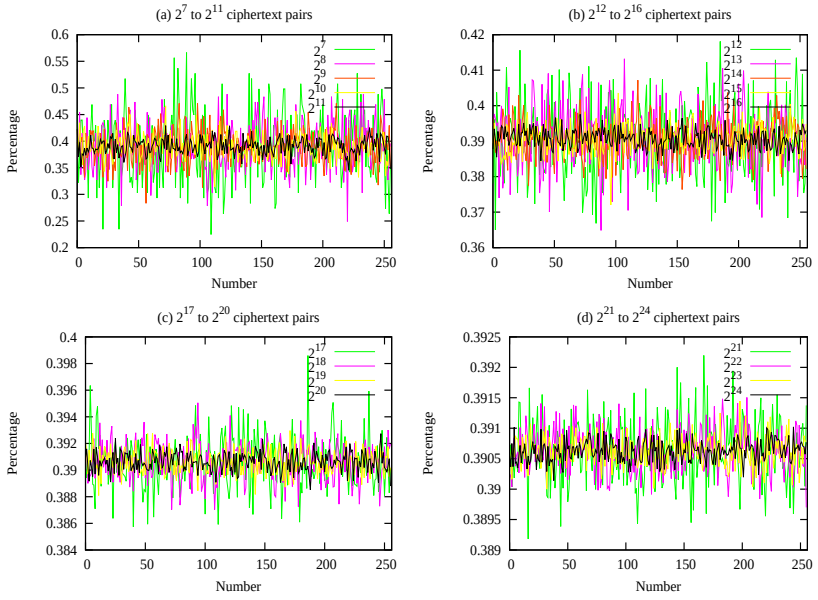

Figure E.48: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 6-round KASUMI for 10 trials (Autofeeding)

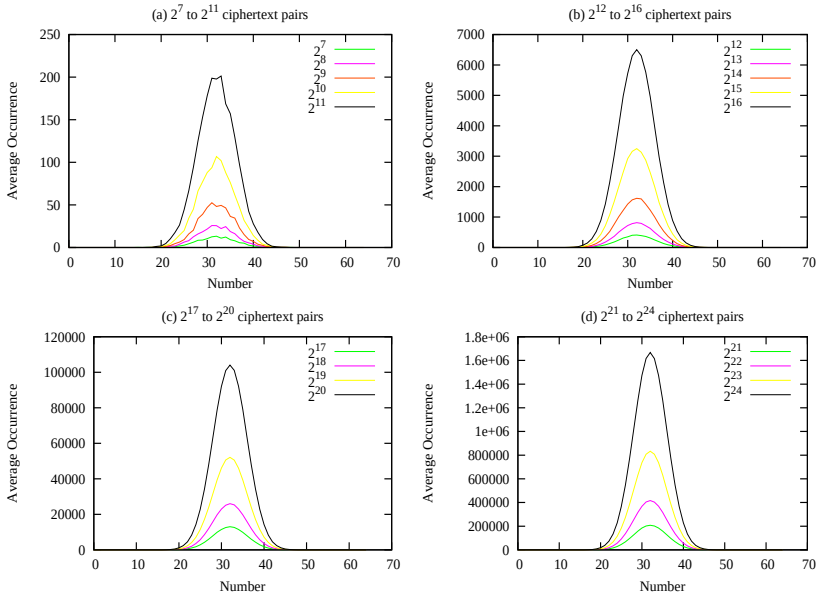

Figure E.49: Average Hamming Weight Distribution between ciphertext pairs for 7-round KASUMI for 10 trials

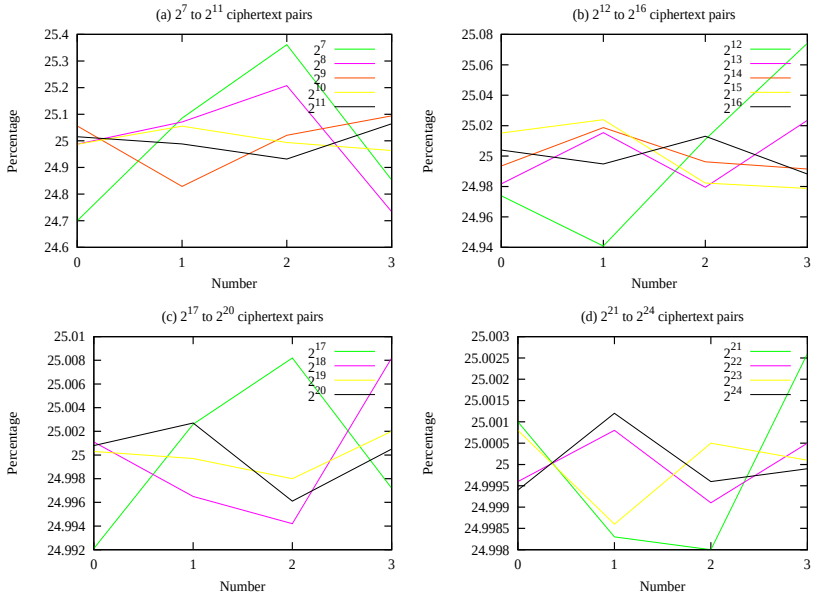

Figure E.50: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 7-round KASUMI for 10 trials

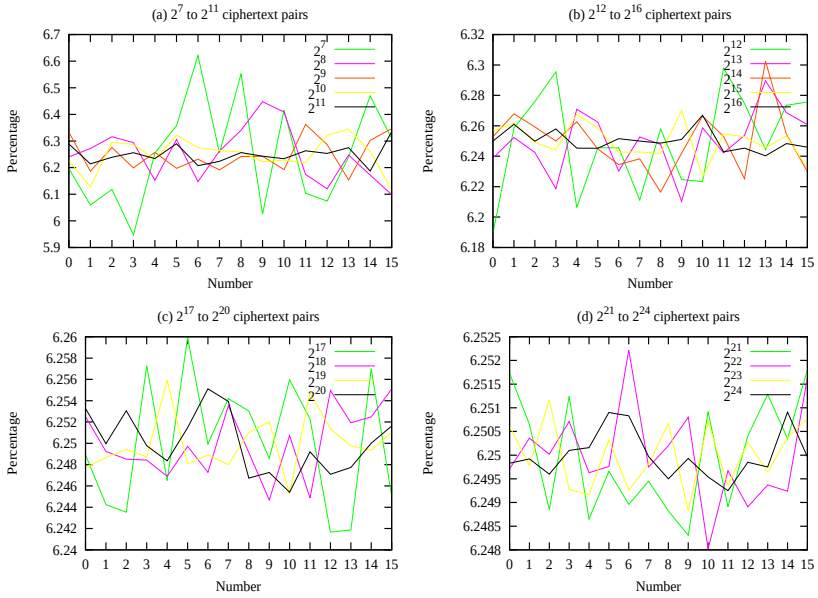

Figure E.51: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 7-round KASUMI for 10 trials

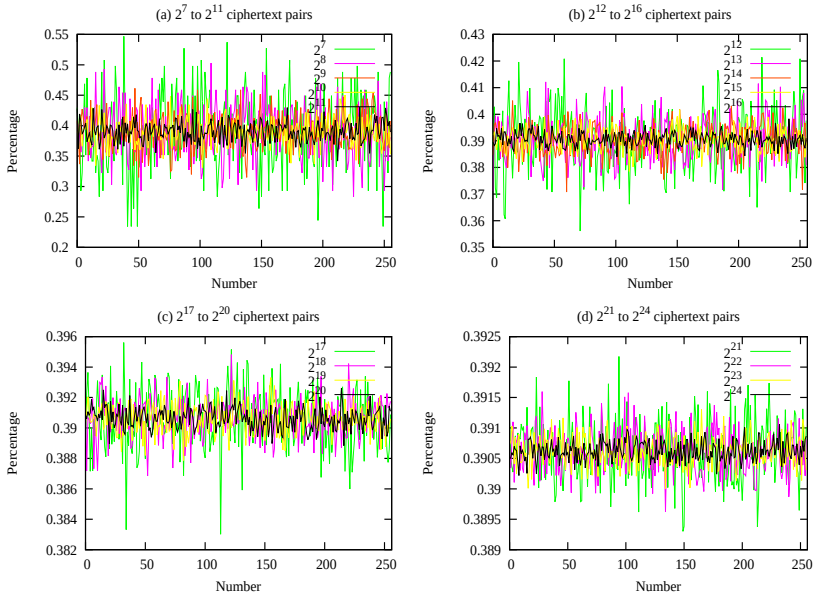

Figure E.52: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 7-round KASUMI for 10 trials

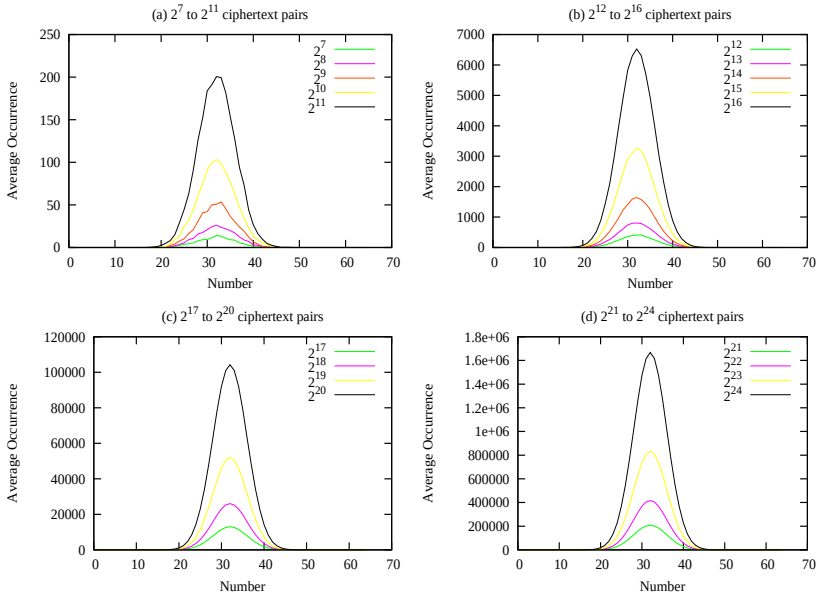

Figure E.53: Average Hamming Weight Distribution between ciphertext pairs for 7-round KASUMI for 10 trials (Autofeeding)

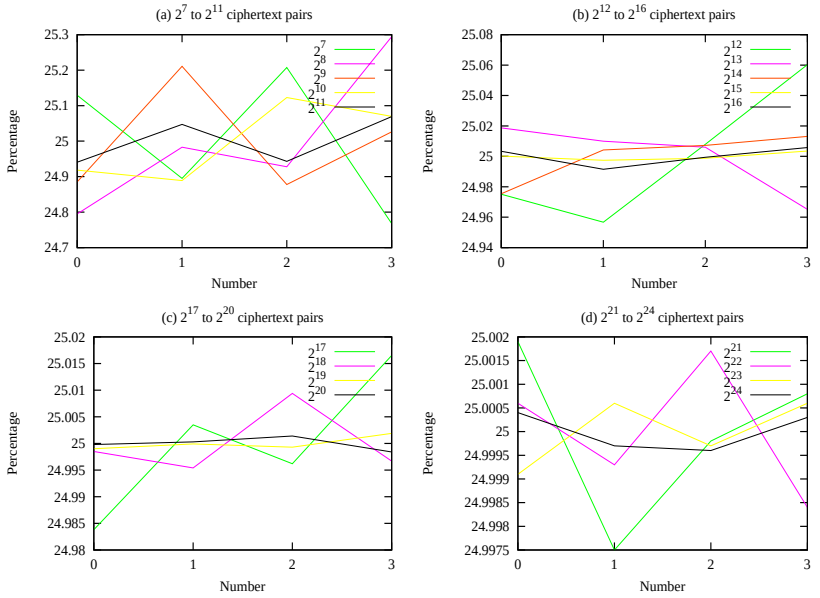

Figure E.54: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 7-round KASUMI for 10 trials (Autofeeding)

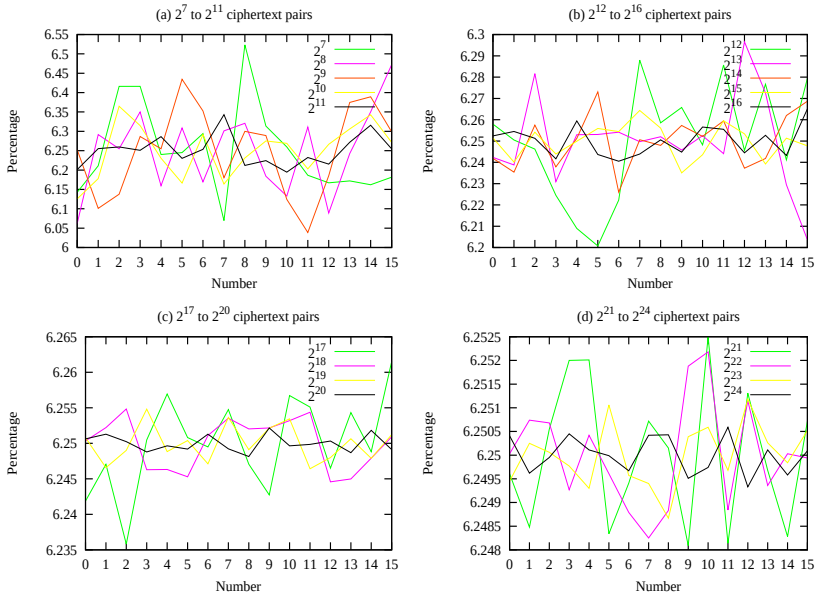

Figure E.55: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 7-round KASUMI for 10 trials (Autofeeding)

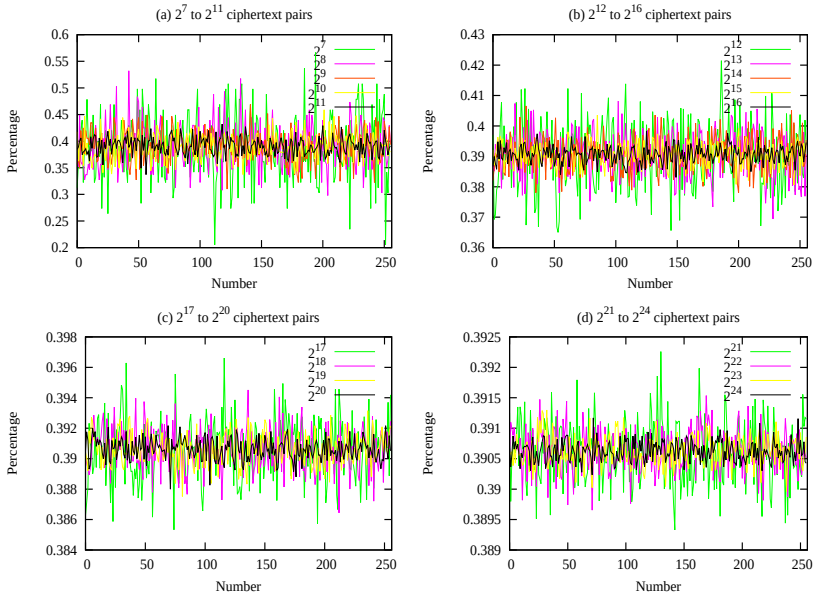

Figure E.56: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 7-round KASUMI for 10 trials (Autofeeding)

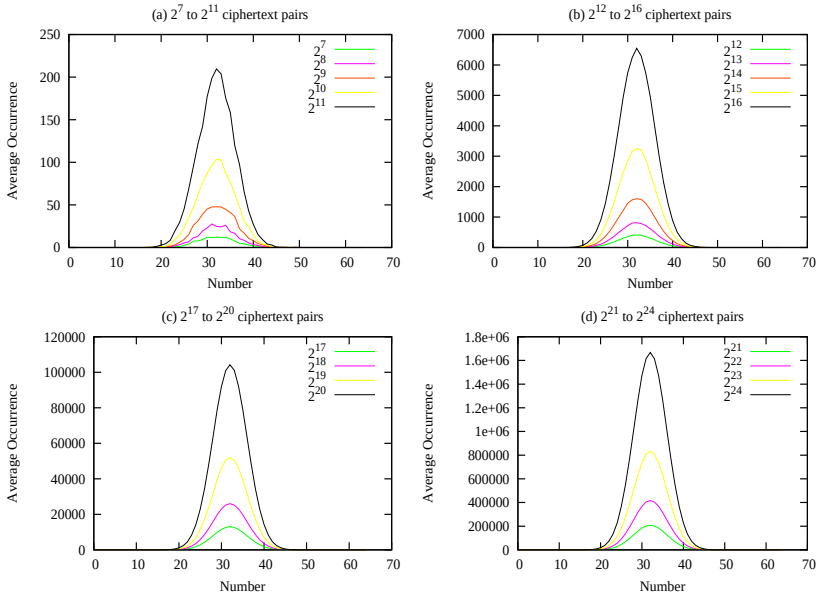

Figure E.57: Average Hamming Weight Distribution between ciphertext pairs for 8-round KASUMI for 10 trials

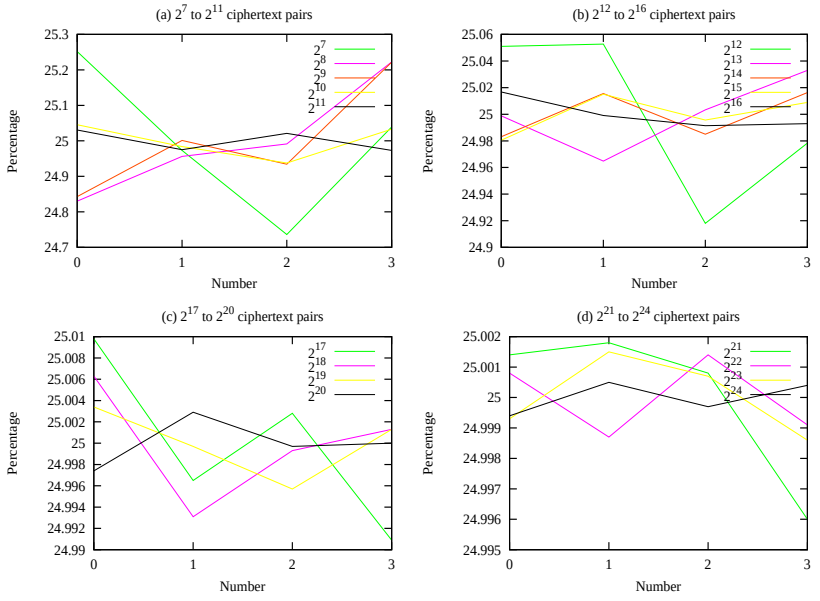

Figure E.58: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 8-round KASUMI for 10 trials

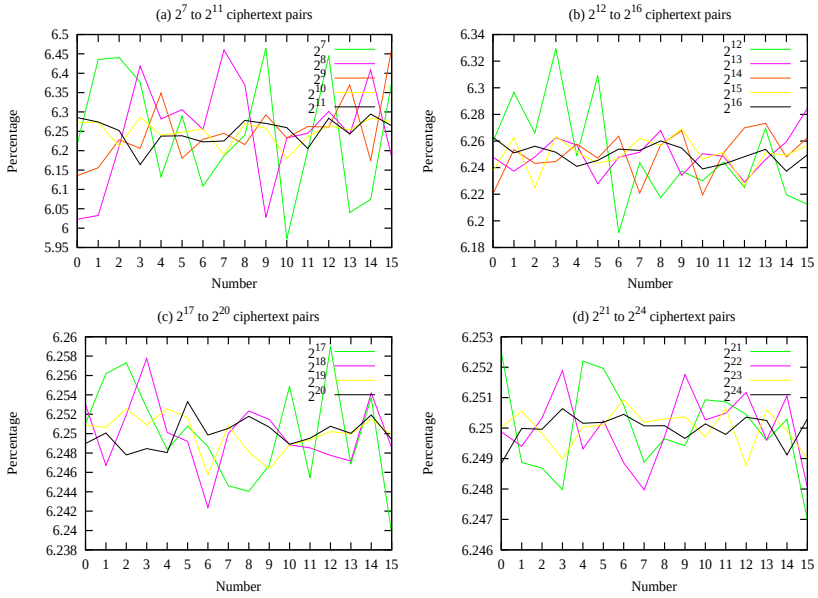

Figure E.59: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 8-round KASUMI for 10 trials

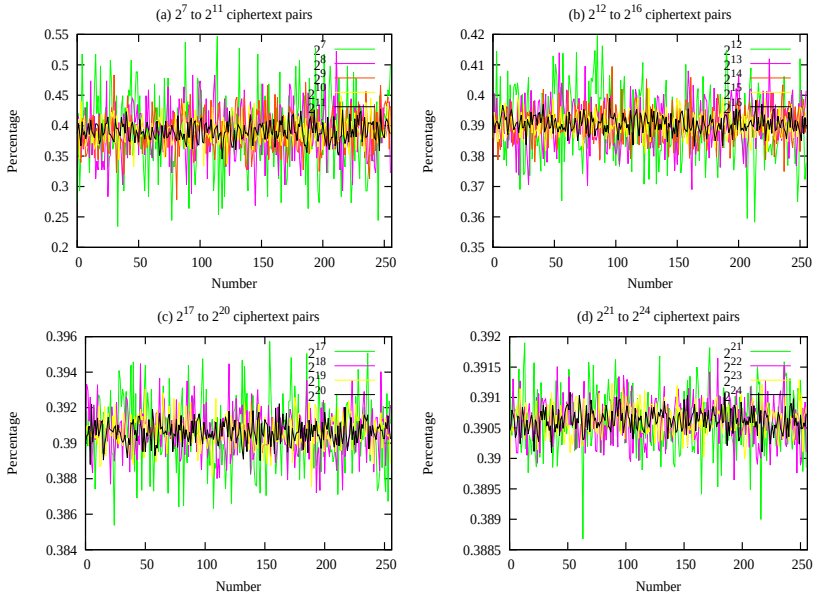

Figure E.60: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 8-round KASUMI for 10 trials

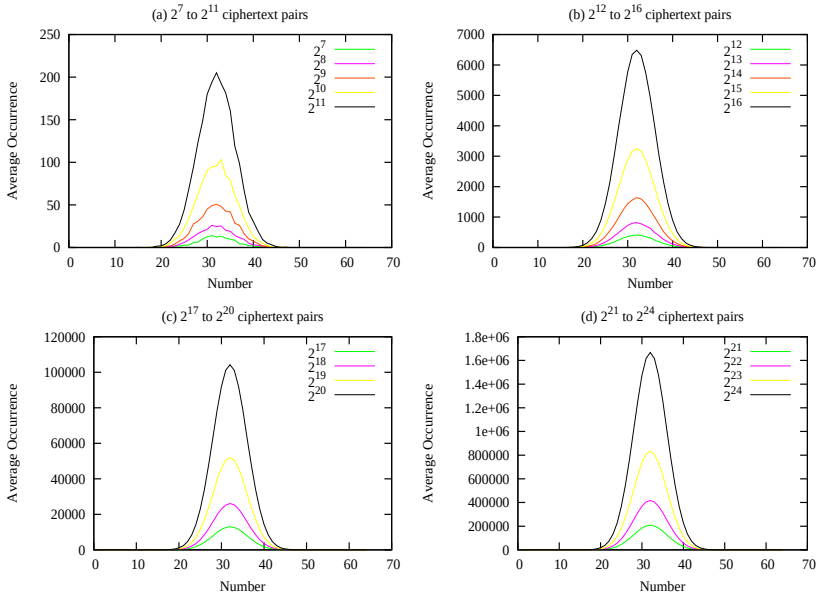

Figure E.61: Average Hamming Weight Distribution between ciphertext pairs for 8-round KASUMI for 10 trials (Autofeeding)

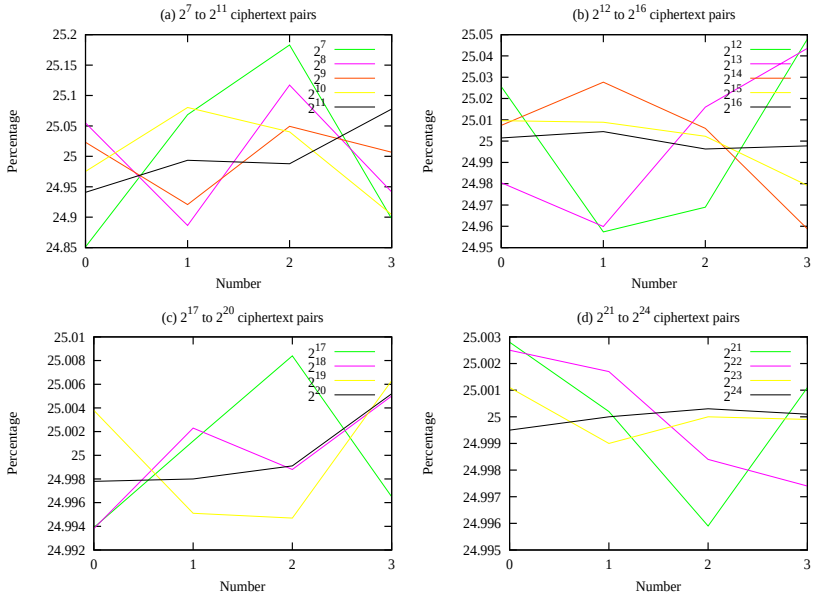

Figure E.62: Percentage of Occurrence of 2-bit substrings for ciphertext pairs for 8-round KASUMI for 10 trials (Autofeeding)

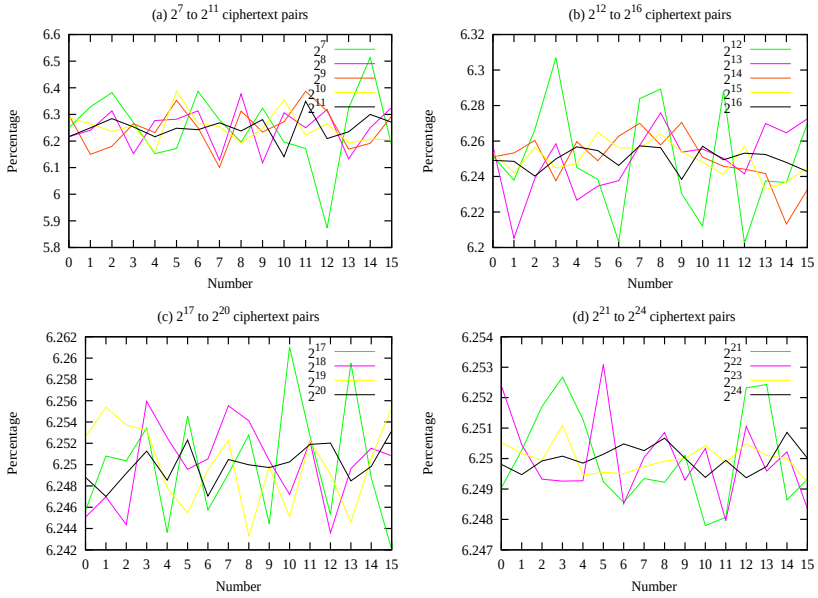

Figure E.63: Percentage of Occurrence of 4-bit substrings for ciphertext pairs for 8-round KASUMI for 10 trials (Autofeeding)

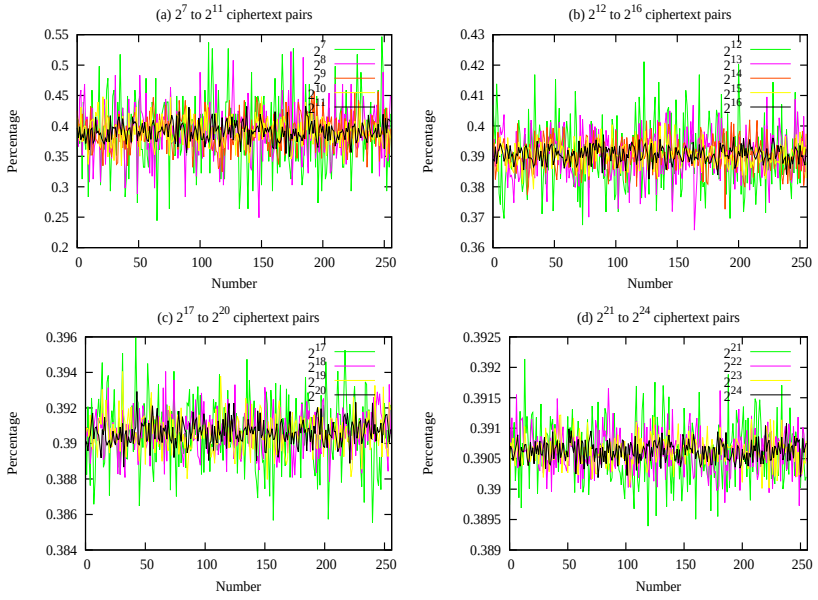

Figure E.64: Percentage of Occurrence of 8-bit substrings for ciphertext pairs for 8-round KASUMI for 10 trials (Autofeeding)

F Snapshots of Experimental Test Results

F.1 Differential Cryptanalysis of 8-round DES snapshots

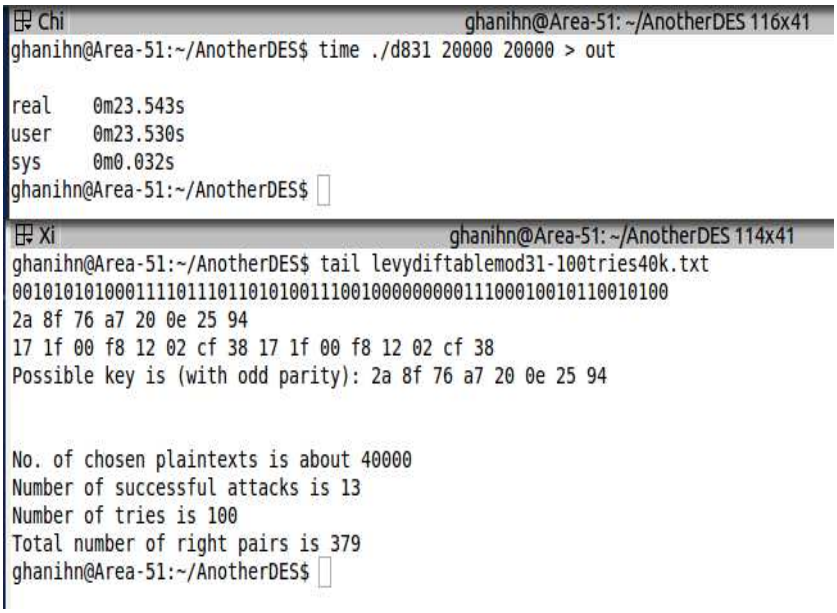

Figure F.1.1: Snapshot of No. of Successful Attacks and time taken for Differential Cryptanalysis of 8-round DES with 40000 chosen plaintext pairs

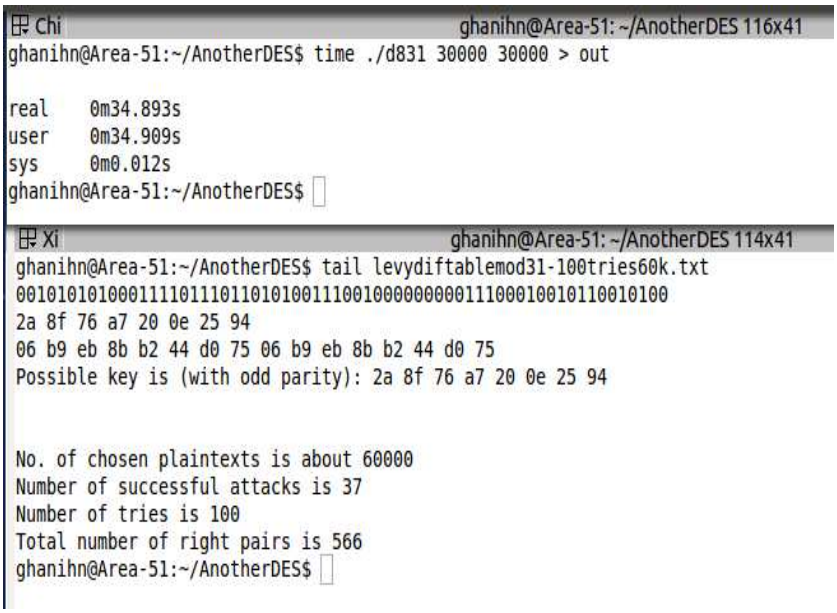

Figure F.1.2: Snapshot of No. of Successful Attacks and time taken for Differential Cryptanalysis of 8-round DES with 60000 chosen plaintext pairs

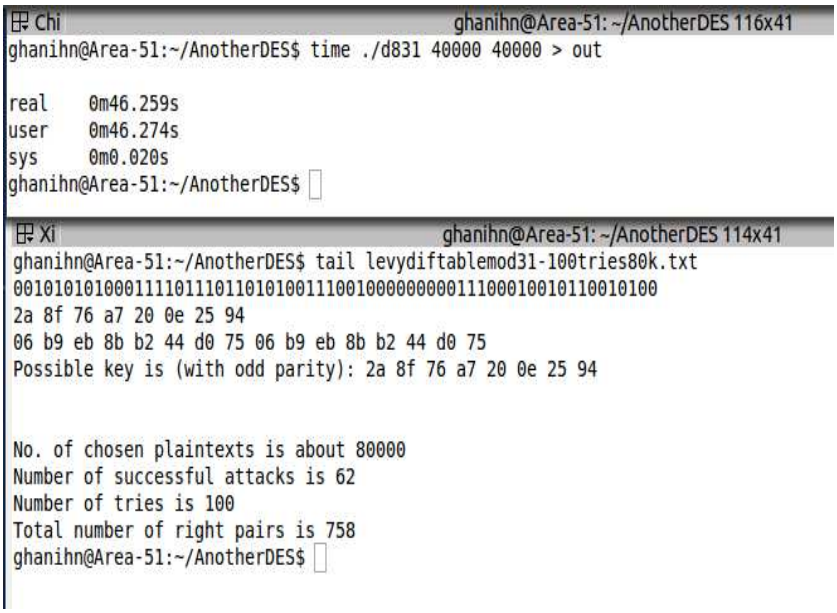

Figure F.1.3: Snapshot of No. of Successful Attacks and time taken for Differential Cryptanalysis of 8-round DES with 80000 chosen plaintext pairs

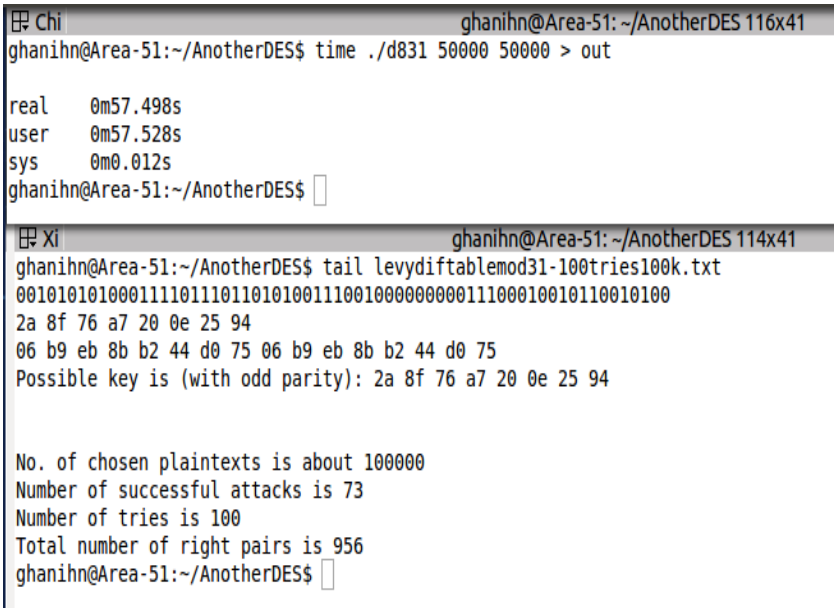

Figure F.1.4: Snapshot of No. of Successful Attacks and time taken for Differential Cryptanalysis of 8-round DES with 100000 chosen plaintext pairs

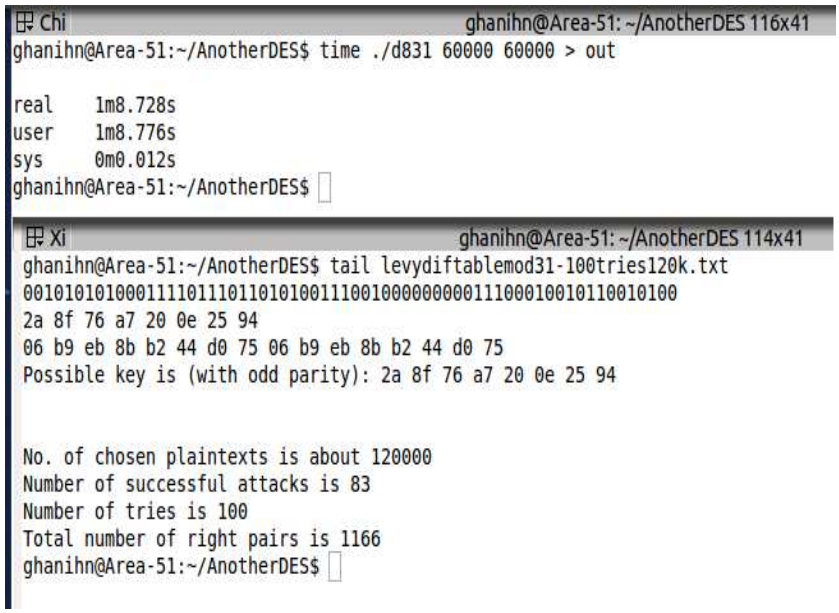

Figure F.1.5: Snapshot of No. of Successful Attacks and time taken for Differential Cryptanalysis of 8-round DES with 120000 chosen plaintext pairs

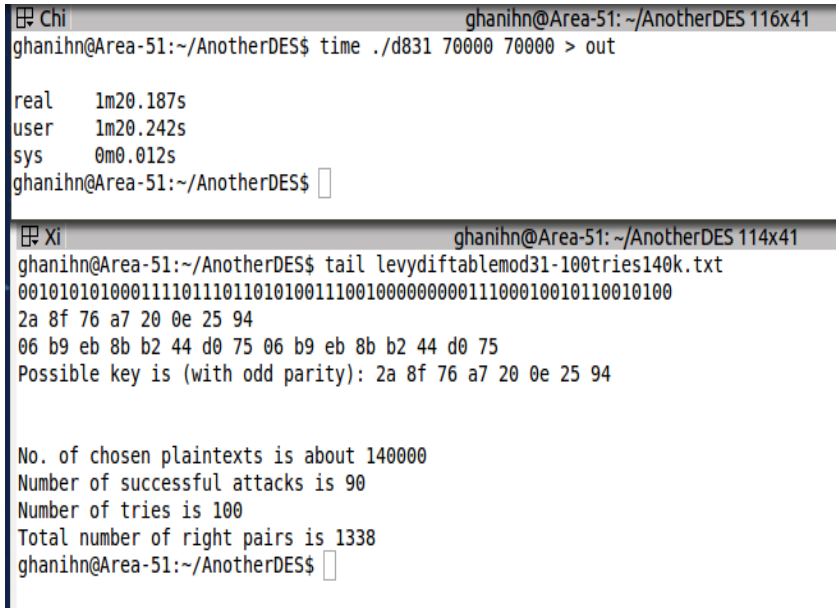

Figure F.1.6: Snapshot of No. of Successful Attacks and time taken for Differential Cryptanalysis of 8-round DES with 140000 chosen plaintext pairs

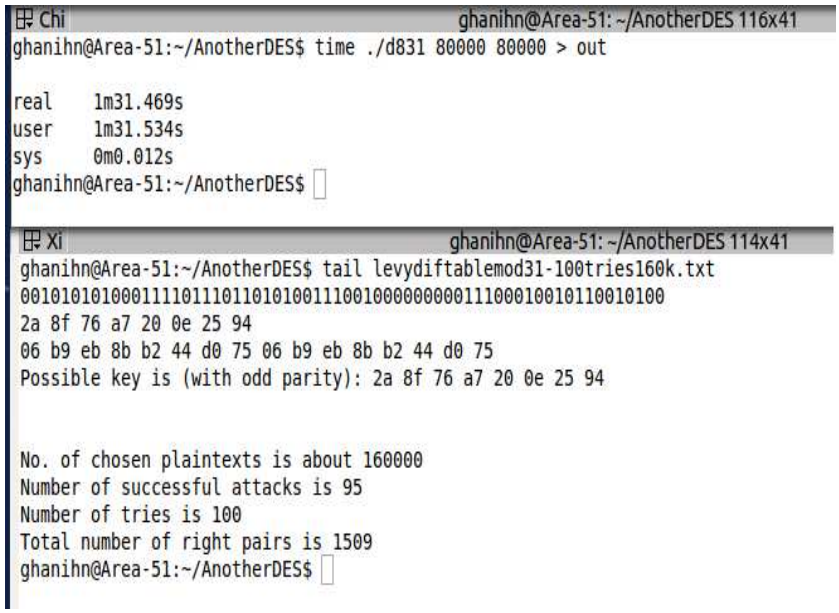

Figure F.1.7: Snapshot of No. of Successful Attacks and time taken for Differential Cryptanalysis of 8-round DES with 160000 chosen plaintext pairs

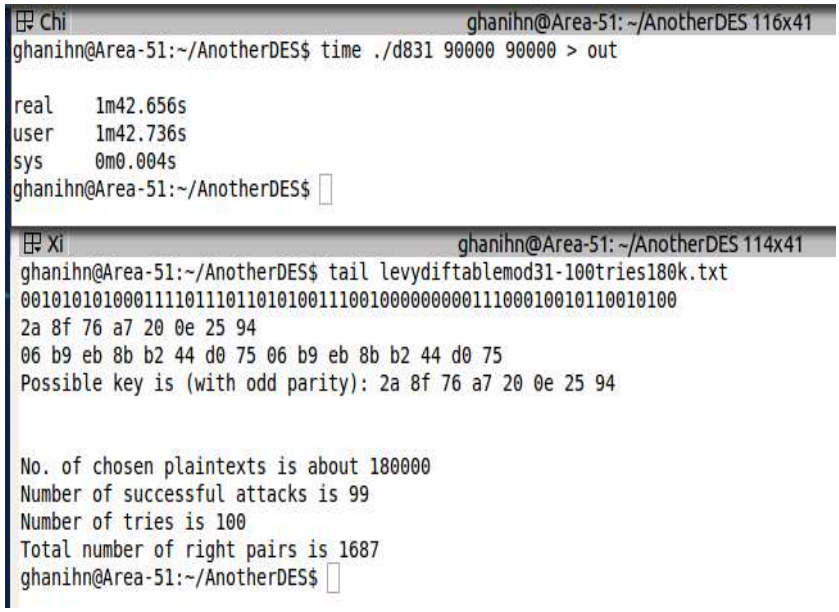

Figure F.1.8: Snapshot of No. of Successful Attacks and time taken for Differential Cryptanalysis of 8-round DES with 180000 chosen plaintext pairs

```
Chi                                                    ghanihn@Area-51: ~/AnotherDES 116x41
ghanihn@Area-51:~/AnotherDES$ time ./d831 100000 100000 > out

real    1m53.954s
user    1m54.025s
sys     0m0.024s
ghanihn@Area-51:~/AnotherDES$
Xi                                                     ghanihn@Area-51: ~/AnotherDES 114x41
ghanihn@Area-51:~/AnotherDES$ tail levydiftablemod31-100tries200k.txt
00101010100011110111011010100111001000000000111000100101100101000
2a 8f 76 a7 20 0e 25 94
06 b9 eb 8b b2 44 d0 75 06 b9 eb 8b b2 44 d0 75
Possible key is (with odd parity): 2a 8f 76 a7 20 0e 25 94


No. of chosen plaintexts is about 200000
Number of successful attacks is 99
Number of tries is 100
Total number of right pairs is 1897
ghanihn@Area-51:~/AnotherDES$
```

Figure F.1.9: Snapshot of No. of Successful Attacks and time taken for Differential Cryptanalysis of 8-round DES with 200000 chosen plaintext pairs

```
Chi                                                    ghanihn@Area-51: ~/AnotherDES 116x41
ghanihn@Area-51:~/AnotherDES$ time ./d831 110000 110000 > out

real    2m5.298s
user    2m5.395s
sys     0m0.004s
ghanihn@Area-51:~/AnotherDES$
Xi                                                     ghanihn@Area-51: ~/AnotherDES 114x41
ghanihn@Area-51:~/AnotherDES$ tail levydiftablemod31-100tries220k.txt
00101010100011110111011010100111001000000000111000100101100101000
2a 8f 76 a7 20 0e 25 94
06 b9 eb 8b b2 44 d0 75 06 b9 eb 8b b2 44 d0 75
Possible key is (with odd parity): 2a 8f 76 a7 20 0e 25 94


No. of chosen plaintexts is about 220000
Number of successful attacks is 100
Number of tries is 100
Total number of right pairs is 2105
ghanihn@Area-51:~/AnotherDES$
```

Figure F.1.10: Snapshot of No. of Successful Attacks and time taken for Differential Cryptanalysis of 8-round DES with 220000 chosen plaintext pairs