Overview of Modern Symmetric-Key Cipher Cryptanalysis Techniques

Author: Sylvain Martinez
Version: 1.1

New versions of this document are based on the version I submitted as part of the requirements for the award of my MSc in Information Security at Royal Holloway, University of London.

CHANGES HISTORY

Version	Date	Changes
1.1	2010/03/22	Page 53: In the Seeded Ciphertext Shuffled box, sC_i and sC_j have been replaced with sC_{n-1} and sC_n Page 64: In the Seeded Ciphertext Shuffled box, sC_i and sC_j have been replaced with sC_{n-1} and sC_n
1.0	2009/08/28	This version was submitted as part of the requirements for the award of the MSc in Information Security at Royal Holloway, University of London.

.

FOREWORDS

I have been interested in cryptography since 1995, which lead me to create my own symmetric-key block cipher algorithm for my BSc project in 1998. Although I have some basic knowledge on how symmetric-key ciphers work, I have never studied the cryptanalysis side of it. Indeed, it is easier to claim a cryptography algorithm is secured than to prove it actually isn't.

I have always looked for and welcomed new challenges; the topic of this thesis was no exception. Furthermore, because this is an area of personal interest, it has been a very time demanding project due to my thirst for knowledge and tendency of wanting to understand the details. As a result I researched a lot of materials and learnt a lot on a field which at times requires a lot of dedication.

Although an introduction will be provided to the different concepts discussed in this thesis, previous foundation knowledge of mathematics and cryptography will be beneficial to fully understand its content.

I would like to thanks the following people who, in some form or another, had an impact on my work. Thank you to my current employer for sponsoring my MSc at Royal Holloway; to my supervisor C. Cid for answering all those endless questions and guiding me along the way; to F. Piper for having re-ignited my flame for cryptology; to R. Walliker for seeding the idea of an MSc in my mind; to D. Mitchell who always like to challenge my thoughts even when wrong!; to T. Tippins for once more providing some valuable feedback; to my work colleagues R. Vekaria and C. Calhoun for taking the time to look at this thesis; to my line managers for being flexible on my working schedule; to my parents for always being supportive and believing in me; to my brother F. for always reminding of what is important; to my brother T. for helping me with the cipher's statistics scripts and correcting my mathematical terminologies.

And finally to my fiancée, for supporting me through the numerous late evenings spent working on this thesis and for everything!

ABSTRACT

The aim of this thesis is to understand the general concept of cryptanalysis and to discuss the main modern symmetric-key cryptanalysis techniques.

Linear Cryptanalysis will be described in detail as it is commonly used and provides the basis for other real and conceptual attacks such as those based on Differential Cryptanalysis. Though an effort has been made to introduce other modern cryptanalysis techniques and related progress in research, the detail on techniques other than Linear Cryptanalysis are beyond the scope of this thesis and can be researched by following guidance in the referenced material.

To get an appreciation on how those techniques can be used, examples of old successful and new emerging attacks will be provided for different types of symmetric-key ciphers. Their real world implications will also be briefly discussed.

To apply what has been learnt while researching this thesis, a cryptanalysis overview will be conducted on the cipher I have created years ago and entitled BUGS. It will provide a description if its concept, a normalisation of the algorithm, a list of relevant attacks and how some of them could be applied to break the cipher.

This thesis consists of eight main chapters and each chapter builds upon the last, allowing the reader to increase knowledge on the subject.

The first chapter is an introduction to this thesis.

The second chapter defines the cryptography terminology used and an overview of symmetric-key ciphers.

The third chapter explains the general concept of cryptanalysis by briefly introducing some foundation techniques.

The fourth chapter introduces modern cryptanalysis techniques and focuses on linear cryptanalysis

The fifth chapter presents four ciphers cryptanalysis and their real world implications.

The sixth chapter gives an overview of the BUGS cipher

The seventh chapter gives a high level cryptanalysis of the BUGS cipher and introduces five potential attacks on that cipher.

The eighth chapter concludes this thesis by summarising what has been discussed, the future of cryptanalysis and my personal learning through the thesis.

TABLE OF CONTENTS

1. Introduction

Cryptanalysis is a topic that many people in the security world would have heard of, but would also not fully understand. It is known by many, because it has been the underlying of most attacks on data and communication security for centuries. It is not understood by many, because it is a complex topic which has evolved to require an increasingly advanced mathematical knowledge. While a large number of technical papers on cryptanalysis are publicly available, only a few books attempting to summarize this topic have been written and are either aimed at cryptanalysts or not detailed enough.

This thesis aims at providing a comprehensive overview of modern symmetric-key ciphers cryptanalysis techniques and introducing, for the first time, some applied cryptanalysis attacks on the BUGS cipher.

Cryptanalysis is in essence the counter part of cryptography; the latter having evidence of its usage dating back thousands of years[1], as greatly explained and summarized in S. Singh's book [1]. Two different examples from that book can be used to illustrate the impact of both cryptography and cryptanalysis in history.

S. Singh relates the ill fated story of Mary Queen of Scots who while being kept captive in England was using cryptography to protect hidden messages related to her planned rescue and conspiracy to assassinate the Queen of England. Her messages were intercepted, meaning that her life depended on the strength of the encryption used to hide her treasonous messages. Unfortunately for her, they were decrypted and she was beheaded on the 8th of February 1587.

However, cryptanalysis is not always successful. S. Singh describes a fascinating story about an ongoing treasure hunt started in the 19th Century. Although with less dramatic impact it illustrates the difficulties related to cryptanalysis. In 1885 a pamphlet, *The Beale Papers*, was published describing the existence of a buried treasure in the USA. Its location, content and the names of those who buried it were each protected by a different cipher. The three ciphertexts were given to someone for safe keeping who never received the key to decipher them. The pamphlet was the result of years spent conducting their cryptanalysis and only with the second cipher, related to the treasure's content, broken. The difficulty only contributed and still contributes to raise interests among those who believe that story to be true.

The above examples illustrate the two extremes with the motivations and impact behind cryptology[2]. Although cryptography can be surrounded by controversy because it protects the privacy of both the innocent and the guilty, cryptanalysis usually isn't, because it is regarded as much as a gift to the attacker as it is a means to drive better encryption schemes.

---

[1] The Scytale cipher is as an example of early cryptography with some research suggesting its usage could be as early as 700BC.
[2] One may speculate where academics fit between those extremes!

2. Cryptography Concepts

2.1 Terminology

- *Cryptography*, is the science related to hiding information. With the aim to provide a combination or all of the following: Confidentiality, Integrity, Authentication and Non-repudiation. This can be achieved through the design of cipher algorithms.

- *Cryptosystem*, designates anything related to the encryption and decryption process and can thus impact its overall security; such as the cipher itself, the random number generators it uses, how the keys are exchanged between two parties and how they are stored, etc

- *Cryptographic primitives*, are low level specific components of a cryptosystem designed to do one specific task and are related to the definition of type of cipher algorithms, pseudo random functions, etc.

- *Cryptanalysis*, is the study of ciphers, ciphertext and cryptosystems with the aim of finding weaknesses to decipher data, without necessarily knowing the secret keys, plaint texts or algorithms used. As long as people have used ciphers to protect their data, others have tried to break that protection to alter its confidentiality, integrity or authenticity.

- *Cryptology*, is the study of both cryptography and cryptanalysis.

- *Plaintext*, noted *P*, is data which in the context of cryptography is not hidden, thus unprotected. Although historically it would have mainly been referring to information such as a plain English text, in the computer era it can refer to anything in numeric form such as a text file, a picture, an executable, etc.

- *Ciphertext*, noted *C*, is data that has been transformed as a result of a plaintext encryption. It is protected in the sense it should be hidden to anyone who hasn't got the corresponding encryption key.

- *Encryption*, noted *E()*, also called encipher; it is the process used to transform a plaintext into a ciphertext. This requires a secret key which will be used to configure and change that transformation.

- *Decryption*, noted *D()*, also called decipher; it is the process used to transform a ciphertext back into a plaintext. This requires the same or a related secret key to the one used during the encryption process.

- *Cipher*, it refers to the encryption and decryption algorithms. Figure 1 inspired by L. Keliher [2] illustrates the concept of a cipher operation where a sender encrypts a plaintext ***P*** with an encryption key K_e into a ciphertext ***C*** and sends it to a recipient who decrypts ***C*** back into ***P*** with a decryption key K_d

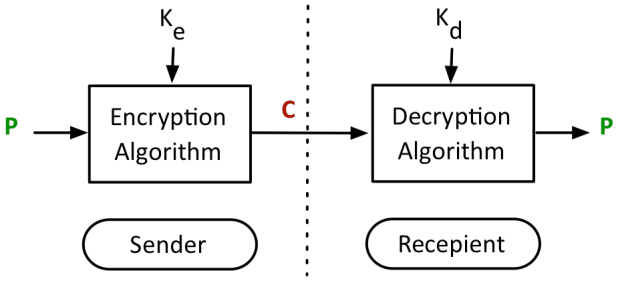

Figure 1. Operation of a cipher.

- *Key Scheduling*, is the process to derive subkeys from an original key.

- *Keyspace*, is the set of all the possible keys available to a cipher algorithm. It is related to the complexity of a brute force attack.

- *Pseudo Random function*, is a function used to generate numbers with statistical properties of randomness. Because they are not truly random they are called *pseudo* random.

- Initialisation Vector, noted *IV*, is a block of bits that can be randomly generated and is independent from the key used for encryption. It is used to produce different ciphertexts while using the same key to encrypt the same plaintext.

- Π: This is the symbol for n-ary multiplications, i.e.: $\prod_{i=1}^{n} a_i = a_1 \times a_2 \times \ldots \times a_n$

- Σ: This is the symbol for n-ary additions, i.e.: $\sum_{i=1}^{n} a_i = a_1 + a_2 + \ldots + a_n$

2.2 Types of cryptography

There are two main types of cryptography, one related to *symmetric-key* ciphers and one to *public-key* ciphers. The following definitions are based on the descriptions written by A. Menezes et al [3].

If for a given cipher it is relatively easy to derive K_e from K_d and vice versa, then it is called a *symmetric-key* cipher. Because of that relationship between the two keys, they must be kept secret/private between the sender and recipient. The term "symmetric" is used because in most implementation $K_e = K_d$

In contrast, if for a given cipher it is impossible to derive K_e from K_d and vice versa, then it is called a *public-key* cipher. The two keys are mathematically related and form a specific pair. One key will be used for encryption and be public while the other key, used for decryption, will be kept private.

Only *symmetric-key* ciphers are relevant for this thesis.

2.3 Symmetric-key Ciphers Overview

2.3.1 Stream Ciphers

A stream cipher breaks a plaintext into a stream of bits which are encrypted sequentially with a stream of key bits, called a keystream. The encryption process is usually very simple and has the property of having no error propagation. The keystream is generated by a keystream generator which uses a seed, such as a random initialisation vector, to initialise the keystream's sequence. As we will discuss in section 5.2.1, a keystream should be reused with the least possible frequency.

A basic and common component of a keystream generator is a Linear Feedback Shift Register (LFSR) which is designed to "stretch" a key by combining some of its bits to generate new ones. This should have the property of diffusing the effect of repetition. An example of a LFSR is illustrated in Figure 2 where a register holds 4 initial bits. The bits 1 and 3 are combined together, all the bits "shift" on the right with the effect of producing the bit 0 as an output. Then, the register where the bit 4 was located before the "shift", will be populated with the bits combination just calculated. The process is repeated as long as an output bit is requested from the LFSR.

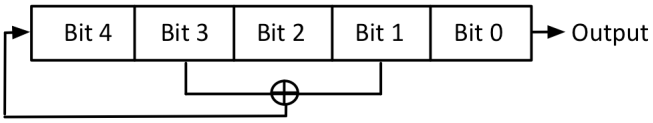

Figure 2. Linear Feedback Shift Register Example

2.3.2 Block Ciphers

A block cipher breaks a plaintext into blocks of bits which are then encrypted into a ciphertext block using a secret key. Usually a n-bits key will be used to encrypt a n-bits plaintext block.
This type of cipher has different modes of operation; the two which are relevant to the cipher studied later in Chapter 6 are the Electronic Codebook mode (ECB) and the Cipher Block Chaining mode (CBC). Both modes are illustrated in Figure 3 and 4 respectively; for N Blocks, where the cipher stops when i=N. Those figures were taken and modified from the A. Menezes et al. book [4] where the description of other modes of operation is also available.

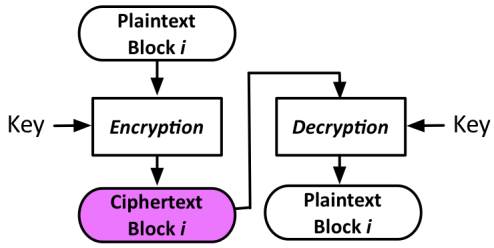

Figure 3. ECB for Plaintext Block 1≤ i ≤ N

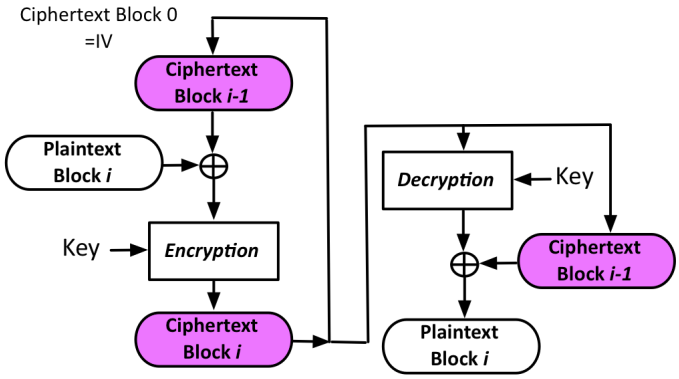

Figure 4. CBC for Plaintext Block 1≤ i ≤ N, using an IV for the first ciphertext input

2.3.3 Block Ciphers Structures

Most of the block ciphers have either a *Feistel* or *Substitution-Permutation Network* (SPN) structure.
The *Feistel* structure was designed by H. Feistel [5] where a plaintext block is split in two halves that are encrypted according to the process described below, taken from M. Stamp and R. Low's book [6] and illustrated in Figure 5.

Let L_0 be the left half of the plaintext and R_0 the right half of the plaintext, then
$\boldsymbol{P = (L_0,R_0)}$
Let *N* be the number of Rounds, *i* the round number such as i = 1, 2, 3, … , n
Let *F()* be a round function transforming an input block using a key, K_i, generated with a key scheduler, then *Li* and *Ri* are generated as follow:
$\boldsymbol{L_i = R_{i-1}}$
$\boldsymbol{R_i = L_{i-1} + F(R_{i-1}, K_i)}$
Let C be the final round output, then
$\boldsymbol{C = (L_n,R_n)}$

The decryption process simply takes this concept backwards with i = n, n-1, …, 3, 2, 1
$\boldsymbol{R_{i-1} = L_i}$
$\boldsymbol{L_{i-1} = R_i + F(R_{i-1}, K_i)}$
The final round output produces *P*, the original plaintext:
$\boldsymbol{P = (L_0,R_0)}$

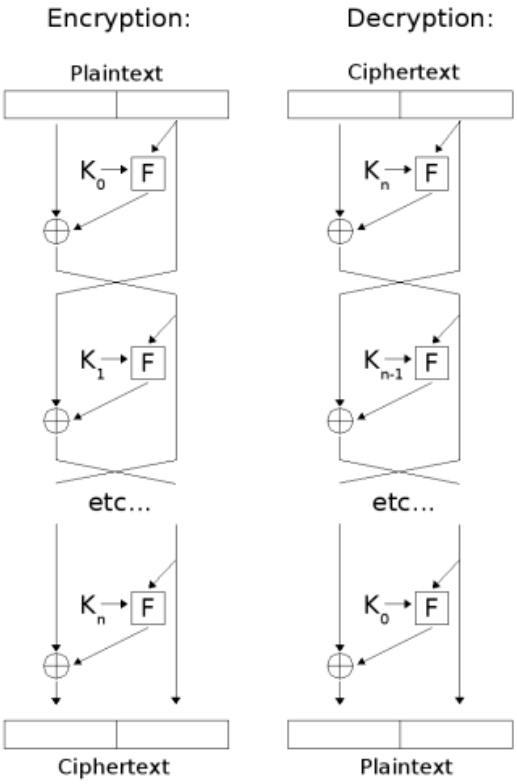

Figure 5. Feistel Cipher diagram from Wikipedia.com

The *SPN* structure has two components, a *substitution* function and a *permutation* function. The *substitution* function, also called an S-BOX, takes input bits and transforms them into different output bits. The transformation process may vary, some ciphers such DES [7] have static S-BOX values as illustrated in Table 1, where the output value depends on the input value. In DES, a 6 bits block is used as input to one of eight S-BOX where the first and last bit will be used to select a row and the remaining four bits will be used to find a column. The value at the intersection of the table will be the four bits long output of the S-BOX. For example, with an input of 18, the row will be 0, the column 9 and the output 10. Other algorithms can also implement key dependant S-BOX, where the key used to encrypt a file will change the behaviour of the S-BOX.
The *Permutation* function acts as a "mixer" or "shuffler", the bits themselves are not transformed but their positions in the block are; hence the output value will also be different from the input value.
An SPN structure combines those two functions one after the other for a number of *Rounds*. For each round the output can be combined with a Key before being used as an input to the next round, as illustrated in Figure 6 for a 3 rounds SPN using four S-Boxes and a Permutation function labelled P.

Row	Column Number															
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
[0]	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
[1]	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
[2]	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
[3]	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Table 1. DES First S-BOX – From A. Menezes et al Book

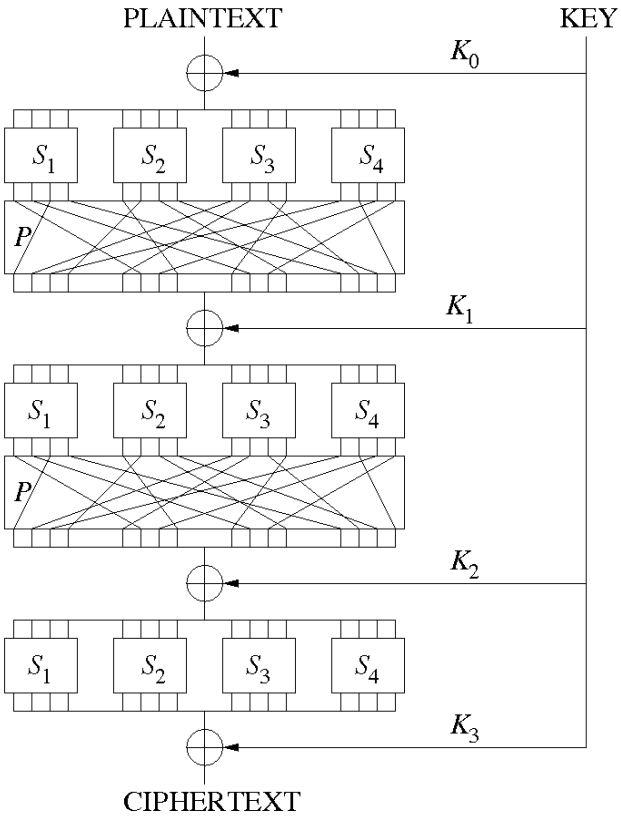

Figure 6. SPN cipher diagram from Wikipedia.com

2.3.4 Diffusion and Confusion

In 1949, C. Shannon [8] introduced two security properties, *diffusion* and *confusion*, on symmetric-key ciphers operation which are still used today as a guideline for new symmetric-key cipher designs. The aim of those properties is to mask any relationships between the plaintext, the ciphertext and the key.

Confusion, aims at masking plaintext and key characteristics, where no information about the original plaintext and the key used for the encryption should be found by studying the ciphertext. For example, studying the ciphertexts of large number of plaintexts encrypted with the same key should not provide information on the key that was used.
Diffusion, aims at providing ciphertext statistical homogeneity, where no ciphertext blocks should repeat significantly more than others nor should their repetition frequency be constant.

A cipher using an SPN network, as described in the previous section, uses substitution and permutation functions to create *confusion* and *diffusion*.

3. Cryptanalysis Concepts

3.1 Cryptanalysis and Cryptosystem

When attempting to provide data security, implementing a cipher algorithm is only one part of the equation. One needs to consider the whole cryptosystem to ensure the level of security is adequate. This can be illustrated with what is called Quantum cryptography. It has been arguably held as the holy grail of cryptography for some time; however, even if this is true it relies on the whole cryptosystem to be "quantumly" secured [9]. One aspect of Quantum cryptography security is that no one can eavesdrop without being detected[3]. This relies on some quantum physics properties that are not practical if the secured data is moved on, say, my iPhone which although being able to do many things, is not yet quantum ready!
The point is that cryptography is only one piece of the data security puzzle, and the full lifecycle of the data being secured needs to be considered, as it will certainly be by any serious attacker.

When we speak about cryptanalysis, we especially look at the security and ways of attacking one part of the cryptosystem. Hence when evaluating the security of a cryptosystem, cryptanalysis is also only one part of the equation.

In this thesis, we will focus on cryptanalysis.

3.2 Historical Cryptanalysis

Most ciphers designed before the computer age era, are vulnerable to three main types of attacks, secrecy leaks, brute force and frequency analysis.

According to Kerckhoffs' Principle the security of a cipher algorithm should never be based solely on its secrecy and should, instead, only be based on the secrecy of the key. Although not knowing a cipher's algorithm details makes its cryptanalysis harder, especially for modern ciphers, early ciphers security were mainly based on the fact only a few people knew how they worked. However, once understood, the likes of the Scytale or Caesar ciphers were trivial to break. C. Shannon later built on this principle and formulated what is known as the Shannon's maxim: "the enemy knows the system". For those reasons secrecy based security is still a debatable concept and not usually recommended.

Brute force, which is also referenced as an exhaustive key search, is when an attacker will try all possible combinations of keys against a ciphertext until the result is satisfactory, i.e.: recovering an English text. Ciphers with small key space are especially vulnerable to this attack, as it would require less time to try all possible combinations. In practice, the average number of keys to try before finding the correct one is the number of possible keys divided by two.

[3] Quantum physic properties mandate that any attempts to measure a photon state will introduce detectable anomalies.

Frequency analysis of those early ciphers allowed deducing the plaintext from the cipher text by looking at statistical language characteristics that are reproduced into the ciphertext. For example, in the English language the letter e is the most commonly used followed by the letters t, a and so on. Although different techniques were used to hide such characteristics with the likes of nulls[4] and polyalphabetic ciphers; frequency analysis attacks could be adapted to remain successful. The most notorious examples are the use of digraph[5] which was the starting point on how a 200 years old French cipher, The Great Cipher [10], was broken as well as a British Cipher called Playfair [11]. Also, the famous Vigenère [12] cipher was broken by combining frequency distribution analysis and reducing the cipher to a monoalphabetic one; exploiting its cyclical nature.
Other techniques can be used to help analysing the type of cipher used; such as the index of coincidence [13], which measures the frequency characters should appear next to each other.

With the advance in computer technology the attacks described above have been optimised and automated. The ciphers once vulnerable to those attacks became easily breakable and what was once considered as extremely secure became obsolete. As a result, modern ciphers became more complex and rely on more advanced mathematical concepts. However, if history is to keep repeating itself, and it has done many times in cryptography, what is now considered as unbreakable may soon be unsecure with some type of mathematical breakthrough. Before this time arrives though, there are some modern techniques that can already be used to identify implementation and algorithm weaknesses. Maybe not surprisingly, statistics still play a major role but this time combined with probability characteristics.

We will now introduce 6 main types of attacks, used in modern cryptanalysis techniques.

3.3 Algorithm and implementation attacks

The first step when doing the cryptanalysis of a new cipher is to study its algorithm to find security flaws and known weaknesses for which some attacks might be applicable. Those known weaknesses may be inherent to the underlying cryptography primitives used, for example algorithms based a Feistel cipher [14] may be vulnerable to linear and differential cryptanalysis attacks described later in this chapter.

What may also start as a known secured cipher algorithm may end up being badly implemented and security flaws introduced into the overall algorithm. This could be referenced as an information technology “translation” issue. A cipher algorithm will first be written into some pseudo code, which will then be translated into a

[4] Nulls: Symbols or letters used as blanks and not to represent anything meaningful.
[5] Digraph: Symbols or letters used to represent a pair of letters. The term Trigraph is used when representing 3 letters.

programming language computers understand. This translation process can leave way of interpretation with disastrous effects on the implementation's security. In addition to this, security flaws might be introduced by the author's ignorance or desire to "cut corners" as we will see later when looking at the WEP algorithm and its Initialisation Vector. Another common example is related to random generators as they usually play a very important role in modern cryptography. They are used to hide ciphertext patterns when encrypting different or similar plaintexts with the same key. If the algorithm's author decided to use a basic computer built-in random generator based on timestamps to generate pseudo random numbers, it will be possible to guess those numbers based on the time the encryption took place and the computer used. Although not a direct attack aimed at the cipher, its indirect effect can reduce its security strength.

3.4 Known ciphertext attacks

Known ciphertext based attacks are the underlying of most practical attacks because it is relatively easy to intercept ciphertexts. If the cipher used to generate the ciphertexts is not known, studying the intercepted information for certain characteristics may help to identify which cipher algorithm was used and which attacks may be more efficient against it. Furthermore, ciphertexts are often used to identify statistical characteristics which may reveal cipher's weaknesses will be explained later.

3.5 Known plaintext attacks

This type of attack requires access to ciphertexts as well as the corresponding plaintexts. This information can be used to identify how an unknown cipher works and also any relationship between the plaintext/ciphertext pairs which may help recovering parts of the key used.
Even if only having one of such pairs is of great help, a large number of pairs is often required to deduce anything of interest. Also, the more pairs encrypted with the same key the more vulnerable a key may become.

A less academic variant of this type of attack is the probable plaintext attack. This relies on the attacker to guess some or part of the plaintexts. For example, most windows users capture information into Microsoft Word Documents which may be encrypted if sensitive. This type of document will always start with the same or similar header identifying the document as a word document, those standard applications "headers" can be found at the start of many different type of document: HTML, PDF, Emails, etc. A parallel could be drawn here with the "cribs[6]" used to crack enigma during the Second World War [15].

[6] Cribs are parts of a plaintext that can be associated to parts of the related ciphertext

3.6 Chosen plaintext or ciphertext attacks

Certain plaintexts may help identifying some characteristics of a key and identify cipher weaknesses. One could use a series of plaintexts first made entirely of 0's bits, and progressively introduce 1's bits to study the effect this has on the resulting ciphertext. This attack can also be extended to a chosen key attack to study the effect of weak keys on the ciphertexts.

Again, one less academic variant is a "forced" plaintext attack, where the attacker forces the victim to encrypt a chosen plaintext. A famous, if maybe a little unknown, example of such attack happened during the Second World War when on some occasions the allies in need of new "cribs" would purposely lay mines in the sea at specific coordinates, for the German boats to find them and warned other boats. Indeed, once those mines were found, they would send those coordinates encrypted [16].

This type of attack can also be adapted to a chosen ciphertext attack where chosen ciphertexts will be generated and the resulting plaintexts pairs studied to identify potential weaknesses.

3.7 Adaptive chosen plaintext or ciphertext attacks

This is a variant on the previous attack where the attacker may change its chosen plaintext based on information gathered during the attack. Again, this can also be applied to the use of adaptive chosen ciphertexts.

3.8 Related keys attacks

Most ciphers use different rounds to encrypt/decrypt data and to increase security they use "different" keys in each round using what is called, a key scheduling process. Those keys, called subkeys, are derived from an original key in some ways, with DES the master key is split in half and each subkey is then rotated by one bit at each round, with AES [17] a one-way function is used. However different those methods are, deriving keys implies a relationship between all the keys generated. Thus, using two or more keys that are related in some special way may allow an attacker to find the original key. Eli Biham [18] popularized this concept by explaining that most key scheduling algorithms can be seen as a set of algorithms which use previously generated keys, or subkeys, to generate new ones. When those algorithms are the same, then the same operations are repeated for each key generated and it is possible in some circumstances to run this process backward and find the original key. This attack is also related to the chosen plaintext attack described previously.

Those 6 types of attacks can now be used as part of the specific attacks we will now describe below. A more in-depth explanation will be given to the most commonly used attacks nowadays, linear and differential cryptanalysis.

4. Modern Cryptanalysis Techniques Overview

4.1 Statistical and probabilistic attacks

As modern ciphers have now evolved into more mathematically complex algorithms, traditional frequency analysis has become obsolete and had to evolve as well. The same principle remains but introduces the concept of probability and statistic where *Probability* relates to the prediction of the likelihood of future events and *Statistic* relates to the analysis of the frequency of past events.

Applying this in cryptanalysis means guessing potential weaknesses in a cipher and analysing how many times those guesses were true in order to improve their accuracies.

As described later in this chapter, a combination of probabilities and statistics now underpin most modern cryptanalysis techniques; in linear and differential cryptanalysis for example, we first look at probabilities to identify potential cipher characteristics, then select the characteristics with the highest probabilities and finally look at the statistics of those probabilities holding true to extract information on the key being attacked.

4.2 Slide Attacks

The outline of this *Known/Chosen plaintext* attack was first described in 1977 [19] but only came to maturity in 1999 [20] and was labelled a Slide-Attack by B. Schneier. It works especially well against the wrong pre-conception that just increasing the number of rounds in a cipher helps making it more secure, even if it was weak to start with. This is because it attempts to find weaknesses in the key schedule and not into how the different rounds impact the plaintext.

The attack requires a large number of plaintext/ciphertext pairs with the aim of finding two pairs for which the first round encryption of one plaintext is equal to the second plaintext. Such pair is called a slide pair and if the round function is weak, we can use that slide pair to extract information on the key being attacked. This is illustrated in Figure 7, where for two pairs (P, C) and (P', C'), if F() is the function used for each round and 'r' the number of round in the cipher, then the slide pair we are looking for should satisfy the following condition: P'= F(P) and C'= F(C)

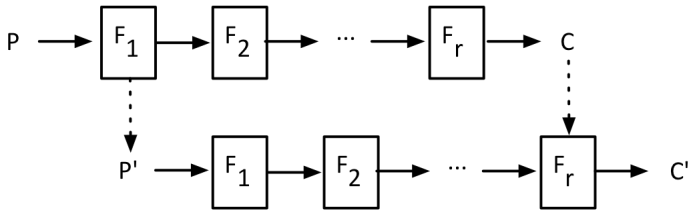

Figure 7. A Slide Pair Representation

For a cipher using plaintext blocks of n-bit size, according to the birthday paradox [21] a slide pair can be found with $2^{n/2}$ plaintext/ciphertext pairs. Extensions of this

type of attack have also been designed against ciphers with more complex round functions: *complementation slide attack*, *sliding with a twist* and a combination of the two have been described by A Biryukov and D. Wagner [22]

4.3 Correlation Attacks

This attack is aimed against stream ciphers where a "Divide and Conquer[7]" approach is taken to exploit potentially weak keystream generator's Boolean functions by studying probabilities on bits correlations between the key and the keystream.

We will explain this attack by using the Geffe Generator as illustrated in Figure 8 This is a weak stream cipher using three Linear Feedback Shift Registers (LSFR) to feed a non-linear Boolean function, F(), which combines the input bits to generate a keystream. The aim is to identify correlation(s) between the bits of the shift registers used in the LFSR and the output bits of the keystream.

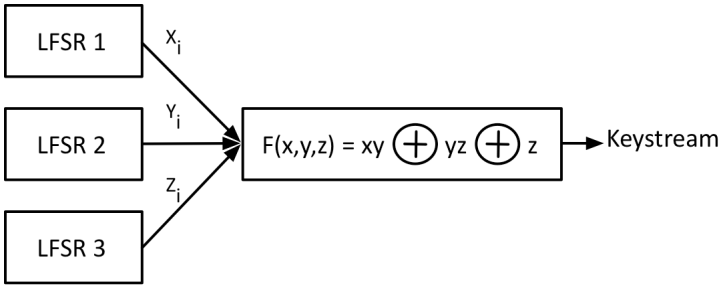

Figure 8. Simple Geffe keystream generator

The key used to generate the keystream will be composed of each LFSR shit register bits; the X, Y and Z bits. For example LFSR1 may be using 3 "x bits", LFSR2 may be using 4 "y bits" and LFSR3 may be using 5 "z bits"; giving a total of a 12 bits key.
The first step of the attack is to list the probabilities for a an input bit of the key (i.e.: X_i) to be equal to the output bit of the keystream. The results are listed in Table 2 where each line is calculated the same way as with the line highlighted in green and described below:
F = xy ⊕ yz ⊕ z = 0 x 0 ⊕ 0 x 1 ⊕ 1 = 0 ⊕ 0 ⊕ 1 = 1

[7] Divide and Conquer: Breaking down a complex problem into less complex sub-problems in order to find a solution more easily

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 2. Boolean function output table

From the above table we can see that 6 times out of 8 we have F=x giving us a probability of ¾ which is higher than the expected truly random probability of ½. This is called a *correlation*. We can exploit this information to find the correct value of the X bits part of the key. To do that, we combine this attack with a known plaintext attack. Because a stream cipher is of the form: $P_i \oplus Keystream_i = C_i$, knowing the plaintext and the corresponding ciphertext will also give us the corresponding keystream. The idea is to initialise the targeted LFSR with all possible shift register values, generate of stream of X_i bits and compare it with the keystream. Once we found a match ¾ of the time we can assume the guessed initial X bits were correct. We can then either try to find other correlations, this time for the Y and Z bits or brute force the rest of the key.
When designing a keystream generator it is therefore very important for it not to leak information of the individual shift registers used by the LFSR. This is called *correlation immunity*.

4.4 Time-Memory Trade-Off

A Time-Memory or Time-Space Trade-Off consists in finding the balance between the data storage space and time performance. It is usually referenced as *TMTO*. In some computer science operations increasing the storage of pre calculated data can increase the speed of execution. However, when those operations occur either on extremely large fields or infinite fields, the data storage required may force the use of slow storage medium such as using a Hard-disk instead of memory, or may even just be too large to be feasible.

Below are some cryptanalysis techniques based on this concept.

4.4.1 Meet in the middle attack

This existence of this attack, which was first described by W. Diffie and M. Hellman [23], is the reason why Double DES (2DES) is not a strong algorithm. This is because although using two different keys of size n to encrypt a plaintext does indeed mean an exhaustive key search would require 2^{2n} keys; it does not however mean that the

complexity of this attack is of the same order, instead it is only of 2^{n+1} thanks to the birthday paradox[8].
We can demonstrate this by attacking the 2DES concept where a plaintext P is encrypted twice with two different keys, K1 and K2, we then have $C = E_{K2}(E_{K1}(P))$. The attack is a *known plaintext attack* and breaks the algorithm into two parts, an encryption and a decryption. The ciphertext will first be decrypted with all possible values of K2 to generate all possible $E_{K2}()$ values representing $E_{K1}(P)$ and the results stored into a "table". The second part will use the plaintext and encrypt it with all possible values of K1 to generate all possible $E_{K1}()$ values also representing $E_{K1}(P)$. We will then have to lookup where there is a match of $E_{K1}(P)$ values for both K1 and K2, when this occurs we would have "meet in the middle" and found the correct values for both keys.

4.4.2 Hellman's TMTO

M. Hellman extended the previous attack into what is called the Hellman Time-Memory Trade-Off Attack [24]. It was first designed against DES but can be adapted against any block cipher. It is a *chosen plaintext attack* in the sense the same plaintext has to be encrypted by different keys a number of times.

The aim of this attack is to find the key used to encrypt a known plaintext *P* and for which we have intercepted the corresponding ciphertext *Z*. The main concept is to conduct some sort of recursive encryption $E_i(P)$ where the resulting ciphertext, C_i, will be used as a key for the next iteration. Therefore, the key has to be the same size as the ciphertext but this is not always true. For example, DES uses a 56 bit key to produce a 64 bit block; we can is this case use a simple reduction function *R[]* to remove the last 8 bits of the ciphertext[9].

This technique can be described with the 4 steps below:

Step 1
We randomly choose *M* number of keys from the cipher's key space as Starting Points (SP)

Step 2
We conduct *T* number of encryption iterations, $R[E_X(P)]$, as described below:
With $\overset{M}{\underset{i=1}{X}} i,0 = SP_i$ we calculate $\overset{T}{\underset{j=1}{X}} i,j = R[E_{Xi,j-1}(P)]$
Where *i* relates to the Starting Point key and *j* relates to the number of encryption iterations.

[8] Birthday paradox: It gives us a complexity of 2^n but because we are conducting two operations in this attack, an encryption and a decryption, the complexity is of 2^{n+1}
[9] Reduce function: Any bits can be removed and the number of bits to remove depends of size on the key/ciphertext

With *EP* standing for Ending Point, the attack workflow is:

$SP_1 = X_{1,0} \rightarrow X_{1,1} \rightarrow X_{1,2} \rightarrow X_{1,3} \rightarrow \ldots \rightarrow X_{1,T} = EP_1$

$SP_2 = X_{2,0} \rightarrow X_{2,1} \rightarrow X_{2,2} \rightarrow X_{2,3} \rightarrow \ldots \rightarrow X_{2,T} = EP_2$

...

$SP_M = X_{M,0} \rightarrow X_{M,1} \rightarrow X_{M,2} \rightarrow X_{M,3} \rightarrow \ldots \rightarrow X_{M,T} = EP_M$

The (black) intermediate points will be discarded and only $\{SP_i, EP_i\}_{i=1}^{m}$ will be recorded into a table sorted on End Points. From the workflow above we can see that *M* refers to the Memory/Space and *T* to the Time/Number of operations required. Defining the values of those variables will be explained in the next section.

Step 3

To attack the intercepted ciphertext *Z* we have to compare its reduced form *R[Z]* with the EP_i ciphertexts to find a match. If we do, then we know the key used was the second last ciphertext generated for that EP_i which is $\boldsymbol{X_{i,T-1}}$.However, this is an intermediate point which has not been stored, hence we will have to calculate the chain again using the corresponding SP_i to recover the key $\boldsymbol{X_{i,T-1}}$.

Step 4

If we did not find a match we then use the *R[Z]* as the only Starting Point key and follow a similar process described in the previous steps:

With $X_0 = E_{R[Z]}(P)$ we calculate $\overset{T}{\underset{j=0}{X}} j = R[E_{Xj}(P)]$

We stop if either X_j matches one of the EP_i ciphertexts to recover the key $\boldsymbol{X_{i,(T-j-1)}}$ or if no matches were found after *T* iterations.

4.4.3 Complexity and Success of Hellman's TMTO

When defining the values of *M* and *T*, a trade-off is needed between the *M* number of keys to be pre-calculated as Starting Point and the *T* number of encryptions to do before reaching an End Point.

Also, there is a number of false positives to consider due to key collisions and the reduction function *R[]*. Indeed, that function has an impact on the '*r*' number of tables required by this attack.

Hellman suggests that the probability of success can be defined as:

$P(\text{success}) = 1 - e^{-MTr/2^k}$

And with *k* being the keylength used, $M = T = r = 2^{k/3}$

Explaining why this is true is beyond the scope of this thesis and a more approachable explanation than Hellman's original can be found in M. Stamp and R. Low's book on cryptanalysis [25] where it is concluded that this technique, which can be applied to any block cipher, has the particularity of requiring "no knowledge of the internal workings of the underlying cipher".

4.4.4 Extensions of Hellman's TMTO

This technique was improved by R. Rivest shortly after it was first published, it introduces the notion of *distinguished points* [26] to optimize the size of the stored values by allowing for a dynamic value of *M.*

It was further improved in 2003 by P. Oechslin [27] who introduced the notion of *Rainbow Tables*. It uses a different reduction function for each encryption iteration and thus does not need distinguished points or multiple tables. However, the latter can still be used to improve on the key recovery probability of success but at a Memory and Time cost. This technique was successfully implemented against the Microsoft LAN Manager Password Hash as described in Chapter 5.

Finally, the *Distinguished Rainbow Points* method [28] is an ongoing project based on a combination of the two previously mentioned extensions. It is interesting to note that this new technique is being designed for COBACOBANA [29] which is a hardware cryptanalysis framework primarily aimed at DES.

4.5 Linear Cryptanalysis

4.5.1 Introduction

In Mathematics a Linear mapping is a function between two vector spaces where the vector addition and scalar multiplication structures of the two are preserved.

The main issue for the cryptanalysis of an S-box used in a cipher is that it is using a nonlinear mapping, which means if x represents the input bits and y the output bits, there is no linear function of x solving y. Linear cryptanalysis attempts to find linear functions/equations between input and output bits, this is done by guessing linear approximations of S-boxes.
This type of attack was first described by M. Matsui and A. Yamagishi in 1993 [30] with 2 different algorithms; but only the second algorithm offers real cryptanalysis value. Although it was first designed to work against the FEAL[31] cipher, it was also later successfully adapted to work on DES and other type of ciphers using structures such as Feistel, SPN, etc

4.5.2 Linear equations

In this type of attack, a linear equation is defined as follow:
Let each element A, B, C and X be binary elements with a value of either 1 or 0
Let $\oplus$ represent an eXclusive OR (XOR)[10].
If $A \oplus B \oplus C = \underline{Z}$
Then due to the commutativity of XOR we also have
$A \oplus B \oplus C \oplus \underline{Z} = 0$
For example: $1 \oplus 0 \oplus 1 = \underline{0}$ because $(1 \oplus 0) = 1$ and then $1 \oplus 1 = 0$
and we also have: $1 \oplus 0 \oplus 1 \oplus \underline{0} = 0$

Let X represents an input element from the plaintext, Y represents a group of output bits from the key/subkeys and let "i" be a specific bit of those elements. For example, X_i, represents the bit number i of the X element and X_3 would therefore represent the 3rd bit. The input element is also usually broken up into several blocks of bits, say 4 bits, hence for an input element of 16 bits the following blocks of bits will be defined: $[X_1, X_2, X_3, X_4]$, $[X_5, X_6, X_7, X_8]$, $[X_9, X_{10}, X_{11}, X_{12}]$, $[X_{13}, X_{14}, X_{15}, X_{16}]$
The bit length of an element will be represented by "n". In the example above, n=16

Those linear equations will relate Input and Output bits, but they do not have to be "paired", or equally represented. Hence the following equations could be selected:
$X_1 \oplus Y_1 = 0$ (Paired)
$X_1 \oplus Y_5 = 0$ (Not Paired)
$X_2 \oplus X_4 \oplus Y_1 = 0$
$X_2 \oplus X_4 \oplus Y_1 \oplus Y_2 \oplus Y_3 = 0$
Also, the opposite equations, called "affine" can also be considered: $X_1 \oplus Y_1 = 1$

[10] XOR: $0 \oplus 0 = 0$; $1 \oplus 1 = 0$ and the opposite, $0 \oplus 1 = 1$; $1 \oplus 0 = 1$

4.5.3 Linear Cryptanalysis Concept

Linear cryptanalysis is a *known-plaintext, probabilistic and statistical* attack using M. Matsui second algorithm to focus on the last subkeys that are generated and can be used to gain information on the key itself, on which those subkeys are based.

The main concept is to approximate the cipher algorithm into a system of linear equations, of the form shown above, where the inputs and outputs bits are somewhat related with the aim of recovering some of the bits from the key used for encryption. It can be achieved because for a cipher to be "perfect" it would require to provide a perfect diffusion and confusion; no cipher being perfect this type of attack attempts to exploit those weaknesses.

Those cipher algorithm imperfections mean that for a given input bit, the related output bit probability of being 0 or 1 isn't exactly ½. That difference, called a *bias*, between a perfect probability of ½ and the measured real probability of a linear expression is what linear cryptanalysis is focusing on, the bigger the difference and the more likely we are to find information on the original key used. The bias of a linear equation/expression, represented as "e", is the absolute value of the probability of that expression being true minus ½:
$e=|p - ½|$

4.5.4 Step 1 - Linear approximation of an S-Box

The first step is to build a full linear approximation of all the S-Boxes used in the cipher by trying all possible linear equations and record the results, that is, how many times they hold true. This will produce a Linear approximation table for the S-Boxes. This means we will be calculating all the probability values and biases for each possible combination of input and output bits.

The rest of this section is based on the excellent example given in H. Heys' Linear and Differential Cryptanalysis tutorial [32]. We will consider a basic 4 rounds SPN cipher using the same S-Box through. Each S-Box will be taking 4 bits as input to produce 4 output bits and will be using the first line of the first DES S-Box as defined in Table 3 There will be a total of 4 S-Boxes per round and the output of those S-Boxes will be XORed with a subkey derived from a master key used to encrypt the plaintext.

Input	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Output	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7

Table 3. S-Box Representation of the 16 possible output values.

Because we are using the same S-Box through the cipher we only need to look at the linear approximation of the one S-Box Using all possible input values we will be recording each time linear equations such as $X_1 \oplus X_4 = Y_2$, which is graphically represented in Figure 9, are true and we will have to do this for all possible linear equations.

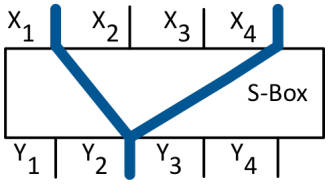

Figure 9. A graphical representation of $X_1 \oplus X_4 = Y_2$ on a 4-bit S-Box

The Table 4 below illustrates a manual process to record the findings of one linear equation. The columns shaded in blue, highlight the bits that have been used while the cells shaded horizontally in green, show when that equation is true.

X_1	X_2	X_3	X_4	Y_1	Y_2	Y_3	Y_4	$X_1 \oplus X_4$	Y_2
0	0	0	0	1	1	1	0	0	1
0	0	0	1	0	1	0	0	1	1
0	0	1	0	1	1	0	1	0	1
0	0	1	1	0	0	0	1	1	0
0	1	0	0	0	0	1	0	0	0
0	1	0	1	1	1	1	1	1	1
0	1	1	0	1	0	1	1	0	0
0	1	1	1	1	0	0	0	1	0
1	0	0	0	0	0	1	1	1	0
1	0	0	1	1	0	1	0	0	0
1	0	1	0	0	1	1	0	1	1
1	0	1	1	1	1	0	0	0	1
1	1	0	0	0	1	0	1	1	1
1	1	0	1	1	0	0	1	0	0
1	1	1	0	0	0	0	0	1	0
1	1	1	1	0	1	1	1	0	1

Table 4. Sample linear approximation – Modified from H.Heys' Tutorial

According to the results, that linear equation is true 8 times out of 16, hence the probability bias is $^{8}/_{16} - ½ = 0$. This means that equation is not very interesting as we are looking for large biases.
Instead of manually trying all possible linear equations until we find the ones with a large probability bias, this process is automated/programmed. A table summarizing the number of times all linear approximations are true for of a given S-Box is produced, as shown in Table 5

Output SUM

Input SUM	0	1	2	3	4	5	…	15
0	8	0	0	0	0	0	…	0
1	0	0	-2	-2	0	0	…	0
2	0	0	-2	-2	0	0	…	2
3	0	0	0	0	0	0	…	-2
4	0	2	0	-2	-2	-4	…	0
5	0	-2	-2	0	-2	0	…	0
6	0	2	-2	4	2	0	…	-2
7	0	-2	0	2	2	-4	…	2
8	0	0	0	0	0	0	…	-6
9	0	0	-2	-2	0	0	…	-2
…	…	…	…	…	…	…	…	…
15	0	-2	-4	-2	-2	0	…	…

Table 5. Linear Approximation Results Table – Modified from H.Heys' Tutorial

Each linear equation is represented by using numbers for the input and output sum parts of the equation. Looking at the linear equation we used previously, $X_1 \oplus X_4 = Y_2$, the result has been highlighted in shaded blue in the above table and it is indexed with:
Input sum = $X_1 \oplus X_4 = 1001 = 9$
Output sum = $Y_2 = 0100 = 4$
Because we are interested in biases which are expressed as $e = p - ½$ and we are using a 4-bit S-Box with 16 possible values, the biases can be calculated as
$e = {}^{n}/_{16} - ½ = {}^{(n-8)}/_{16}$
Hence, the results in the above table are the number of times each linear equation was true minus 8; to calculate the bias that value then needs to be divided by 16. This allows to quickly identify linear approximations of interest, those with large biases, and eliminate the ones with no or small biases. I.e., a result of 0 in the above table means the bias is also equal to 0 as 0/16 = 0.

4.5.5 Step 2 - Linear approximations for the complete cipher

The second step is to study the linear approximations of the full cipher, from start (plaintext) to finish (ciphertext) through all the rounds of the cipher up to the second last round. This is achieved by concatenating selected linear approximations of S-Boxes from each round, which we call linear approximation scenarios. The aim is to find scenarios with large linear probability bias.
Such scenario has been graphically represented in Figure 10 with the following linear approximations:
S11: $X_{1,1} \oplus X_{1,3} \oplus X_{1,4} = Y_{1,2}$ (the index represents a round number and bit position)
S21: $X_{2,2} = Y_{2,2} \oplus Y_{2,4}$
S31: $X_{3,2} = Y_{3,2} \oplus Y_{3,4}$
S32: $X_{3,6} = Y_{3,6} \oplus Y_{3,8}$

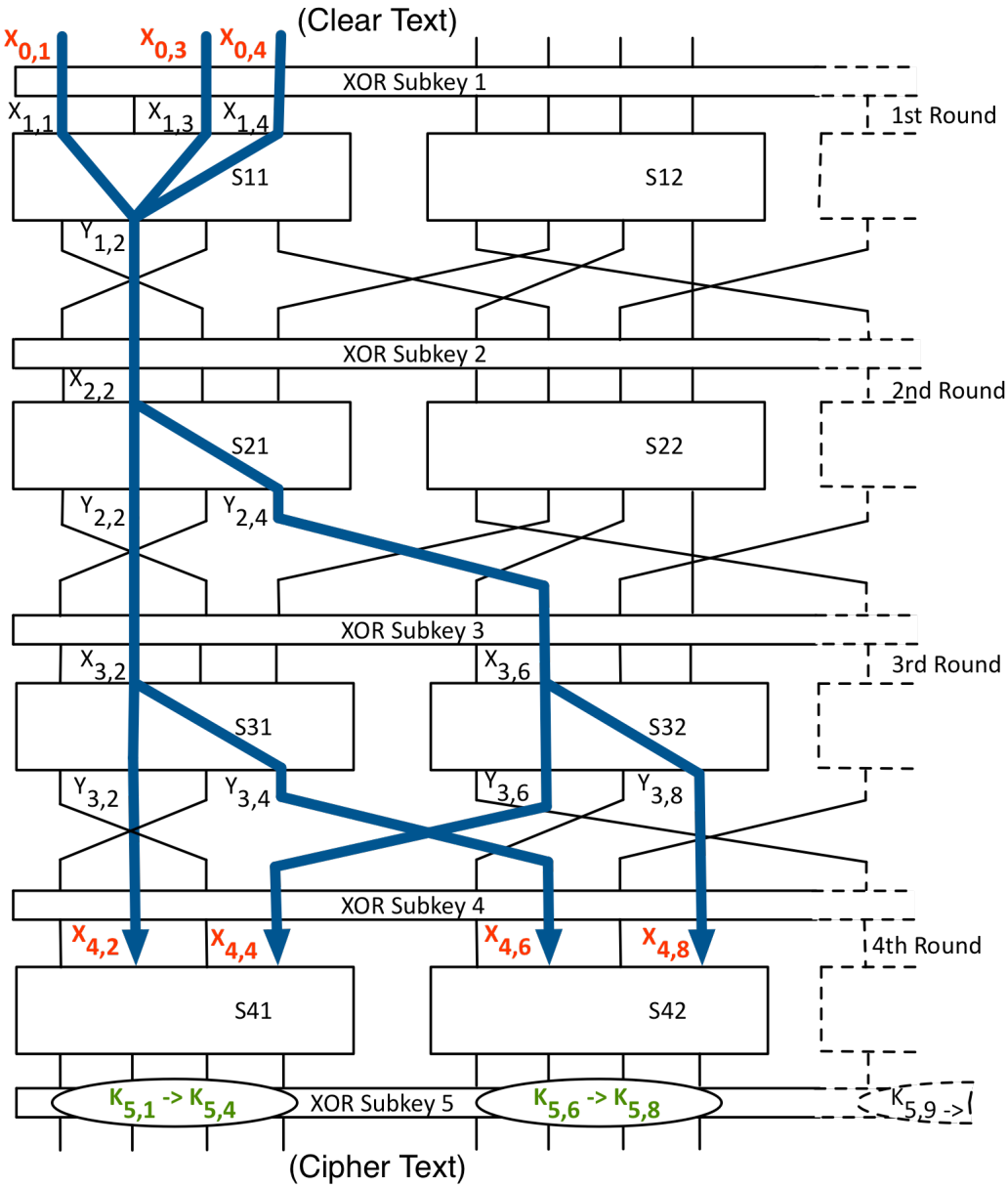

Figure 10. Sample Linear Approximation – Modified from H.Heys' Tutorial

Highlighting the "extremities" in red, we have the following linear equation:

$X_{0,1} \oplus X_{0,3} \oplus X_{0,4} \oplus X_{4,2} \oplus X_{4,4} \oplus X_{4,6} \oplus X_{4,8} =$

$X_{1,1} \oplus X_{1,3} \oplus X_{1,4} \oplus Y_{1,2} \oplus X_{2,2} \oplus Y_{2,2} \oplus Y_{2,4} \oplus X_{3,2} \oplus Y_{3,2} \oplus Y_{3,4} \oplus X_{3,6} \oplus Y_{3,6} \oplus Y_{3,8}$

The right side of the equation is made of the subkey bits in between the extremities used to construct the linear approximation. Their values will depend of the key and derived subkeys used and will therefore be unknown, their sum will either be equal to 0 or 1. However, the linear approximation scenario required must have its input and output bits equal such as: $X_{0,1} \oplus X_{0,3} \oplus X_{0,4} = X_{4,2} \oplus X_{4,4} \oplus X_{4,6} \oplus X_{4,8}$

Which means we must also have: $X_{0,1} \oplus X_{0,3} \oplus X_{0,4} \oplus X_{4,2} \oplus X_{4,4} \oplus X_{4,6} \oplus X_{4,8} = 0$

Therefore if the subkey bits sum is equal to 0 then this approximation holds true and the sign of its bias is the same as the subkey bits sum's bias. If it was equal to 1, then the sign of its bias would be the opposite. This, however, has no overall influence as we are only interested in the absolute value of the bias.

To calculate the bias of those combined linear expressions we cannot just multiply each linear expression's bias together. For this, we need to use the piling-up lemma principle as described by M. Matsui.

4.5.6 Step 3 - Pilling-up lemma

Let X_i be a binary element of an S-Box. If p_1 is the probability of $P(X_1 = 0)$ being true, then $1-p_1$ is the probability of $P(X_1 = 1)$ being true and the piling up lemma for 2 elements is:

$$P(X_1 = i) = \begin{cases} p_1 & \text{for } i = 0 \\ 1-p_1 & \text{for } i = 1 \end{cases}$$

$$P(X_2 = i) = \begin{cases} p_2 & \text{for } i = 0 \\ 1-p_2 & \text{for } i = 1 \end{cases}$$

$$\begin{aligned} \text{And } P(X_1 \oplus X_2 = 0) &= P(X_1 = 0)P(X_2 = 0) + P(X_1 = 1)P(X_2 = 1) \\ &= p_1p_2 + (1 - p_1)(1 - p_2) \\ &= p_1p_2 + (1 - p_1 - p_2 + p_1p_2) \\ &= 2p_1p_2 - p_1 - p_2 + 1 \end{aligned}$$

If we now replaces p_1 with $½ + e_i$ as explained in section 2.6.3, we have:

$$\begin{aligned} P(X_1 \oplus X_2 = 0) &= 2(½ + e_1)(½ + e_2) - (½ + e_1) - (½ + e_2) + 1 \\ &= ½ + 2e_1e_2 \end{aligned}$$

The probability bias for $X_1 \oplus X_2 = 0$ is therefore $2e_1e_2$ and is noted as $e_{1,2} = 2e_1e_2$
The same principle can be used for a linear expression with n elements and the following deducted:

$$e_{1,2,\ldots,n} = 2^{n-1} \prod_{i=1}^{n} e_i$$

In the previous Figure 10 the linear equations had the following probabilities and biases:
S11 Linear equation had a probability of $^{12}⁄_{16}$ and a bias of +¼
S21 Linear equation had a probability of $^{4}⁄_{16}$ and a bias of -¼
S32 Linear equation had a probability of $^{4}⁄_{16}$ and a bias of -¼
S33 Linear equation had a probability of $^{4}⁄_{16}$ and a bias of -¼

Applying the pilling up lemma described above, the concatenated linear expression bias is therefore:
$e = 2^3 (¼ \times -¼ \times -¼ \times -¼) = -^{8}⁄_{256} = -^{1}⁄_{32}$

4.5.7 Step 4 – Target partial subkey attack

Once large biases are found, the fourth and final step is to apply those scenarios to a number[11] of plaintext/ciphertext pairs encrypted with the same key that is being attacked and record when they hold true.

The last subkey bits involved in the linear approximation will be the target of the attack, they are called the *target partial subkey* and have been highlighted green in Figure 10. All possible values, meaning combination of bits, of this target partial subkey will be used against each plaintext/ciphertext pair. Each time, the ciphertext will be partially decrypted using the target partial subkey to generate the lower end of our linear approximation scenario extremity ($\mathbf{X_{4,2}} \oplus \mathbf{X_{4,4}} \oplus \mathbf{X_{4,6}} \oplus \mathbf{X_{4,8}}$), this will then be compared to the upper end extremity ($\mathbf{X_{0,1}} \oplus \mathbf{X_{0,3}} \oplus \mathbf{X_{0,4}}$) which we know, as they are bits of the plaintext. If the two extremities are equal then the scenario is true for that value of the target partial subkey and a "True" count on that value is incremented.

Once this is finished, the value of the target partial subkey with the highest "True" count can be assumed to be the correct one. It can be interesting to calculate what H. Heys describes as a bias magnitude for each "True" count, because you would expect such number to be very close to the linear approximation scenario bias (i.e.: $|-1/32|$). For 'N' number of plaintext/ciphertext pairs, this can be calculated as:
|bias| = |"True" count – N/2| / N

4.5.8 Search Algorithm

One of the most difficult parts of linear cryptanalysis is to define good or best linear approximations. Different algorithms have been designed to help find those preferred approximations, the 2 most notable ones are the Matsui Linear Expression search algorithm [33] and an improved version published a year later [34].

4.5.9 Success and Complexity of Attack

For linear cryptanalysis to be successful a number of factors are required:

- A large number of plaintext/ciphertext pair is required
- The linear approximations of each S-Box should be independent. The reality is that they are not, thus impacting probabilities calculation accuracy.
- The function deriving subkeys from the encryption key is not a one-way function, so once parts of the last subkey are found it is possible to reverse back to some original encryption key bits. If a one-way function is used, there are still two less efficient ways to use this type of attack. The first option is to make your way up to the original key by attacking the previous subkey once one has been found. The second option is to switch to a brute force attack on the key once one or X number of subkeys have been found as it still reduces

[11] Number of plaintext/ciphertext: it depends of the scenarios' biases and is explained later in that section.

the overall time required to brute force the encryption key by removing one or X number of round of complexity.

- The larger the linear approximation scenarios bias and the better the chances to recover part of the last subkeys. This means it is preferable to involve Linear Approximations of S-Boxes with large biases. It also means the number of S-Boxes involved overall needs to be kept to a minimum while designing the scenarios as S-Boxes probabilities are multiplied by one another and thus reducing the overall scenario bias (See The Pilling Up Lemma).
- This type of attack is mainly used against block ciphers even if it can and has been adapted against stream ciphers. J. Golic has discussed in different papers its generic concept [35] as well as a specific attack on the Bluetooth stream cipher [36]

The complexity of the attack will depend on a number of factors as well:

- The number of plaintexts required for a decent attack success rate is proportional to the Linear approximations scenarios biases and approximately equal to $r \times 1/e^2$, where *r* is a ratio which does not affect the overall complexity of the attack, hence the number of required plaintexts is often only referenced as e^{-2}. In our previous example where the scenario's bias was $-1/32$, the number of plaintexts required was therefore: $(-1/32)^{-2} = (2^{-5})^{-2} = 2^{10} = 1{,}024$ which can be approximated to 1,000.
- It is important to note that depending of the cipher being attacked, that ratio might increase, for example Matsui describes in [37] a number of $8e^{-2}$ plaintexts to provide a 96.7% success rate against DES and E. Biham uses the same ratio in [38] to describe a 78% success rate against FEAL-8. Although this does not affect the complexity of the overall cryptanalysis, it can have a non negligible impact on the number of plaintexts required and its real world feasibility; i.e. 8 more times of plaintext to be intercepted.
- For this attack to work, its concept has to be adapted to most ciphers being targeted. The complexity of a cipher has an impact on the difficulty of approximating linear equations out of non linear equations.
- Only parts of the key bits are recovered and it is usually combined with a brute force attack on the remaining part of the subkey. Furthermore, this attack is based on guesses, of which some can be wrong; however, correctly guessing a number of subkeys will reduce the difficulty of the brute force attack and can save valuable time.

4.6 Linear Cryptanalysis Variants

4.6.1 The Linear Hull Effect

Linear Cryptanalysis is a powerful attack that has been widely used for some years; however, because it is using a lot of bias approximations, those biases can sometimes be inaccurate. This is called the Linear Hull effect and was first described by K. Nyberg [39]. It occurs when some linear approximation scenarios involving the same initial input and final output bits can be achieved using different linear S-Box approximations, thus different subkey bits. Scenarios that individually have small biases may be combined to produce a new scenario with a much larger bias and less complex attack. A more in-depth study on the applicability of this concept can be found in L. Keliher PhD thesis [40].

4.6.2 Multiple Linear Approximations

This variant was first presented by B. Kaliski and M. Robshaw [41]. The main concept is that instead of working on a large number of plaintext/ciphertext pairs, the focus is on a limited number of such pairs but with a much larger number of linear approximations used against the same targeted key bits.

For each round, different linear approximations involving the same subkey bits will be used and their results, meaning the combined total counts of when they hold true will be weighted as follow:
Let U represents that new combined total counts. Let T represents the number of time each linear approximation holds true as explained in the 4.5.4 section, Table 5
Let *e* represents the biases and let *N* represents the total number of plaintext/ciphertext pairs.

$$U = \sum_{i=1}^{N} T_i \times \frac{e_i}{\sum_{j=1}^{N} e_j}$$

'U' will have a similar bias to the T's but with a reduced variance resulting in the need of less plaintext/ciphertext pairs for a successful attack. Although this technique never really saves the cryptanalyst much time and is mainly used to optimise attacks when limited plaintext/ciphertext pairs were available, recent progress seem to show a possible speed improvement over a simple linear cryptanalysis. Such framework has recently been described in by A. Biryukov, C. De Canniere and M. Quisquater's paper [42].

4.6.3 Other Variants

The following other variants which may be of interest were found in the Encyclopaedia of Cryptography and Security [43]:

- Key-ranking which is a trade-off between data and time analysis [44]
- Partitioning cryptanalysis which "studies correlation between partitions of the plaintext and ciphertext spaces" [45]
- Chi-square cryptanalysis which is a form of side channel attack [46, 47]
- Non-linear approximations [48]

4.7 Differential Cryptanalysis

4.7.1 Introduction

This type of attack is often associated and explained in conjunction with Linear Cryptanalysis of which it is pre dating. It was first published by E. Biham and A. Shamir in 1990 as an attack on DES [49] and is one of the most important discoveries in modern cryptanalysis. However, it was also independently discovered as early as the mid seventies by the NSA[12] and IBM. Because of the strategic advantage of knowing such technique against many ciphers it was never publicly acknowledge until 1994 [50]. Some recently declassified top secret NSA documents about American Cryptology during the cold war offer an interesting insight on secrecy of knowledge and has been comprehensively summarized by T. Johnson [51]

4.7.2 Differential

This attack focuses on *differential* defined as follow.
For a set of N input bits $X = [X_1 X_2 \dots X_n]$ and N output bits $Y = [Y_1 Y_2 \dots Y_n]$, we consider two sets of inputs noted as X' and X'' and the related two outputs Y' and Y''.

The input difference is $\Delta X = [\Delta X_1 \Delta X_2 \dots \Delta X_n]$ where $\Delta X_i = \Delta X'_i \oplus \Delta X''_i$
And the output difference is $\Delta Y = [\Delta Y_1 \Delta Y_2 \dots \Delta Y_n]$ where $\Delta Y_i = \Delta Y'_i \oplus \Delta Y''_i$

A *differential* is a pair (ΔX, ΔY), where for a given set of input differences, ΔX, the corresponding output differences, ΔY, have also been calculated. It should be noted that although a difference calculated using a XOR is the most common, other forms of operation is also possible.

4.7.3 Differential Cryptanalysis Concept

Differential cryptanalysis is a *chosen-plaintext, probabilistic and statistical* attack; it uses a similar structure to Linear cryptanalysis but instead of focusing on the linear approximations of input and output bits of S-Boxes it focuses on the relationships between the differences of a pair of input bits set and the differences of the corresponding pair of output bits set.

The main concept is to carefully select plaintexts and study the generated ciphertexts to extract information of the key used. This is achieved by defining full cipher differential characteristics where for a set of plaintext pairs their differential inputs should generate specific differential outputs. Those pairs will be encrypted and when a differential characteristic holds true, the related last subkey bits will then become the target of the attack; likewise with Linear Cryptanalysis this will be called the *target partial subkey* and a similar attack process will be followed: a partial decryption by using all the possible target partial subkey values combined with a count of when they hold true.

[12] NSA: The USA National Security Agency may have known about this technique much earlier than in the seventies.

4.7.4 Step 1 – S-Box Differential

The first step defines the differential characteristics for each round; this is when the output differences of one round are used as the input differences of the next round.

For each S-Box with input X and output Y, all possible values of X will be used to calculate all possible values of Y. With this information and by selecting some differential input values, ΔX, it will be possible to deduce the corresponding differential output values, ΔY, and a count of those differential occurring will be recorded into a difference distribution table/list similar to the one used in Linear cryptanalysis. However, unlike Linear cryptanalysis we are not looking for large biases, hence there is no need to deduce 8 to the count of occurrences but to calculate the differential probability it is still required to divide by 16 when using 4 bits S-Boxes. What we are looking for is large probabilities, if the S-Box was ideal each differential would be equal to 1/16 but this is mathematically impossible and for N number of input bits, a large probability for (ΔX, ΔY) is a probability which is much larger than $1/2^n$.
With most cipher the keys have no effect on the differential, but in some it does and this mean a more complex differential calculation. This is the case with the blowfish cipher for example where the S-boxes behaviours are determined by the key themselves.

4.7.5 Step 2 – Differential characteristics for the complete cipher

The second step focuses on differentials with a high probability and involving a small number of subkey bits.
The most highly probable differential characteristics will be combined together to define differential characteristics for the entire cipher with an initial ΔX, from the plaintext, and a final ΔY from the ciphertext. Such characteristics define the values of (ΔX, ΔY) for which they hold true. The S-Boxes which are not impacted by the differential characteristics will use zero differentials as inputs and outputs. Those impacted will have non-zero differentials as input and be called *active* S-Boxes. As with Linear Cryptanalysis, it helps to represent those differential characteristics into a similar diagram to the one show in Chapter 4.5.5, Figure 10.

4.7.6 Step 3 – Target partial subkey attack

The final step, using the previous information, involves generating pairs of plaintexts for which the initial differential input holds true. Those plaintexts will be encrypted and each time a full cipher differential characteristic holds true for a given input pair, such pair will be identify as a *right pair* as opposed to a *wrong pair* when this is not the case.

The subkey bits related to the most highly probable *right pair* will be used as target, likewise with Linear cryptanalysis this will be called the *target partial subkey*. All possible values for that partial subkey will be used to partially decrypt the ciphertext and a count will be kept of the subkey values for which the corresponding input differential is true. The subkey value with the highest count will have a high probability to be correct and the rest of the key can then be brute force.

4.7.7 Success and Complexity of Attack

For differential cryptanalysis to be successful a number of factors are required:

- A large number of *chosen* plaintext/ciphertext pair is required; this makes this attack less likely to succeed in the real world as it no longer only relies on intercepting data but also modifying or influencing it.
- Each S-Box differential characteristic probability should be independent. The reality is that they are not, thus impacting probabilities calculation accuracy.
- The larger the differential probabilities and the smaller number of *active* S-boxes, the better the chances to recover part of the last subkeys.
- As with Linear Cryptanalysis not using a one way function to generate subkeys is also a requirement and the right balance needs to be found between the number of subkey bits to extract and the number to brute force.

The complexity of the attack will depend on a number of factors as well:

- Exactly defining the number N of chosen plaintexts required is very complex but a good approximation is $N = 1/P_d$ where P_d is the probability of the second to last round differential characteristic. That probability, P_d, is calculated by the product of each active S-Box differential probability as defined in the full cipher differential characteristic.
- Some ciphers have been designed to resist this type of attack by purposely increasing the number of active S-Boxes and keeping the S-Boxes difference pair probability to a minimum.
- The right balance needs to be found between the number of subkey bits to extract and the number to brute force.

4.8 Differential Cryptanalysis Variants

A number of new attacks based on Differential Cryptanalysis have been designed since the original concept was first made public, the following variants overview can be found in more details in C. Swenson's recent book on Cryptanalysis [52]

4.8.1 Differential-Linear

The first variant combines the 2 main attacks previously described, Linear and Differential Cryptanalysis. It was first described by S. Langford and M. Hellman [53]. The idea is to use the differential of linear approximation scenarios applied to plaintext pairs to reduce the number of plaintexts required to conduct a successful attack. This is done by generating plaintext pairs slightly differently than the method described in standard differential cryptanalysis where a plaintext is first generated then XORed with a fixed value (i.e.: ΔX) to generate the corresponding plaintext pair. In this attack, plaintext pairs will be generated by swapping certain bits of a given plaintext for which the linear approximation scenario has the same effect. This also has the advantage of providing a more efficient ratio of generating plaintext pairs of 3 pairs for 2 plaintexts as opposed to the default 1 pair for 2 plaintexts.

4.8.2 Conditional Characteristics

This attack was first described by I. Ben-Aroya and E. Biham [54] against ciphers using the key itself to modify their cipher characteristics, i.e. how the S-Boxes are defined; specifically the two ciphers Lucifer [55] and Randomized DES (RDES) [56]. With standard differential cryptanalysis such ciphers are very difficult to attack. The idea is to define characteristics which are also key-dependant, called *Conditional Characteristics*, to maximize their probability when using only a limited key space at a time. RDES which is more difficult to attack than DES using differential cryptanalysis becomes overall less complex to attack by using conditional characteristics.

4.8.3 High Order Differentials

In response to some ciphers designed to get more uniformed S-Boxes differences to resist differential cryptanalysis, a new attack called High-Order Differentials was created. It was first described by L. Knudsen [57] and extend the original cryptanalysis by focusing on the study of differences between differences. Differential cryptanalysis only uses first order input/output differentials; for example, ΔX and ΔX' are two first order input differentials. In this new attack, the following input differential is considered: $\Delta X^2 = \Delta X \oplus \Delta X'$ where '2' refers to a second order input differential. It is also possible to have a third order, fourth order and so on; hence for n^{th} order we have: $\Delta X^n = \Delta X^{n-1} \oplus \Delta X'^{n-1}$

4.8.4 Truncated Differentials

This attack was also first described in L. Knudsen's paper referenced above. It attempts to make use of truncated differentials or in other words partial differentials for which only part of the input differences are required to produce part of a specific output differences. The advantage is the possibility of recovering bits of a target

partial subkey even when a complete differential characteristic does not hold true. However this may decrease the accuracy and increase the effort required to conduct a successful attack. This was highlighted by B. Schneier who estimates a successful attack on its Twofish [58] algorithm would require 2^{100} chosen plaintexts in response to the Truncated differential cryptanalysis paper written 5 years earlier by S. Moriai and Y. Yin [59].

4.8.5 Impossible Differentials

This attack was first presented by A. Shamir and a video of his generous seven minutes presentation is available online [60]; instead of focusing on highly probable differences, it focuses on differentials that are impossible, hence with a probability of 0. Finding such impossible differentials was described in a technique called *miss in the middle attack* [61] which consists in generating an input and output differentials and roll back each end of the differential until the two comes to a contradiction, meaning they do not match. When this occurs, the related subkeys value will be discarded as a potential candidate until only one subkey value candidate remains. This type of attack also applies to other form of cryptanalysis such as Linear and Differential cryptanalysis and some of their variants.

4.8.6 Boomerang Attacks

Differential Cryptanalysis exploits differential characteristics of the full cipher; this attack focuses instead on differential characteristics only of part of the cipher. D. Wagner, who first published this technique [62], describes a technique of differential meet-in-the-middle attack where the cipher encryption workflow is broken into two stages. This allows for attacks when no suitable differential characteristics have been found for the full cipher.

For P_0 being a plaintext, C_0 the corresponding ciphertext, $E_i()$ the i^{th} stage of the encryption function and $E_i^{-1}()$ the i^{th} stage of the decryption function, then breaking the cipher into two parts can be represented as:
$C_0 = E(P_0) = E_1(E_0(P_0))$ for the encryption process and
$P_0 = E^{-1}(C_0) = E_0^{-1}(E_1^{-1}(C_0))$ for the decryption process.

The first part of the attack creates differential characteristics for the first stage of the encryption process, E_0, and the second stage of the decryption process, E_1^{-1}. The second part derives a new differential related to where the encryption and decryption stages “meet”. Then, this new differential is combined with a plaintext and corresponding ciphertext to derive four differentials, referred as a *quartet*, and will be used when holding true to attack the key.

This variant has also been itself extended in a number of other attacks such as the *Amplified Boomerang attack* [63] aimed at helping to identify the right quartet and the *Rectangle attack* [64] with the additional aim to improve the probability of the attack.

4.8.7 Related Key Attacks in differential cryptanalysis

The Related Key attack described previously in section 3.8 can be adapted to Differential Cryptanalysis where different values of the key being attacked are used to generate differentials.
Until recently this variant was considered as unpractical and with limited success potential. As of August 2009, it seems this could not have been more wrong. As discussed later in this thesis, this variant combined with the Boomerang attack has become the basis of a new wave of promising attacks against AES.

4.9 Other Cryptanalysis Attacks

Below is a brief summary and references to other relevant cryptanalysis attacks.

4.9.1 Integral Cryptanalysis

This attack was designed by L. Knudsen [65] against the Square cipher [66], it is therefore also known as the *Square Attack*. It is was primarily aimed at Block Ciphers based on Substitution-Permutation-Networks but it has also been successfully adapted to work against Feistel based cipher into what is a called a Saturation Attack [67]. Unlike differential cryptanalysis which focuses on the difference between input/output pairs of values, this attack focuses on the relationship between input/output sums of values. It has the characteristic to apply to some ciphers which are not vulnerable to differential cryptanalysis. This attack can also be combined with the previous Interpolation attack.

4.9.2 Algebraic Cryptanalysis

This class of attack, which is getting an increasing level of attention, takes a targeted block cipher and represent it as a set of polynomial equations. It has the advantage of requiring less plaintext-ciphertext pairs than conventional block cipher cryptanalysis. A more in-depth look at this technique and emerging variants is discussed in C. Cid and R. Weinmann paper [68]. The techniques mentioned below are examples of algebraic cryptanalysis.

4.9.3 Interpolation Attack

First presented in 1997 by T. Jakobsen and L. Knudsen [69], this attack is based on the *Lagrange interpolation formula*[13] and uses simple algebraic functions as S-Boxes to construct polynomials with pairs of plaintexts and ciphertexts. Labelled by some as being very academic in nature this attack tends to be discarded maybe too quickly.

4.9.4 XSL Attack

This controversial Algebraic cryptanalysis attack was first published in 2002 by N. Courtois and J. Pieprzyk [70] claiming to have the potential to break AES. This seems to have been an over statement as explained in 2005 by C. Cid and G. Leurent [71]. However, it did put the spotlight on this type of cryptanalysis and still generates some related research on this technique.
The main concept is to derive, from the cipher being attacked, large systems of quadratic polynomial equations which can be solved using a method called Extended Sparse Linearization (XSL).

[13] The author also suggests using the *Newton Interpolation formula* instead to speed up the attack.

4.9.5 Algebraic IV Differential Attack or the Cube Attack

Even more controversial, this attack on symmetric-key cipher applies to both block and stream ciphers but appears not to be as practical as maybe first claimed.
A. Shamir and I. Dinur published a paper in 2008 describing an attack called the *Cube Attack* [72] and claiming it could break previously unbreakable block and stream ciphers alike even if the inner working of the targeted cipher is not known. The idea is that a cipher is vulnerable if an output bit can be represented by a polynomial of relatively low degree over a finite field[14] of secret and public variables, in other words, key and input bits.
The controversy started when M. Vielhaber complained the *Cube attack* was in fact a plagiarism of his own paper entitled *Algebraic IV Differential Attack (AIDA)* [73] and published earlier. At the Eurocrypt 2009 conference I. Dinur attempted to defend their paper by saying this was a generalisation and improvement over AIDA.
Although M. Vielhaber still disagrees it seems to have divided some members of the cryptanalysis community as some key differences are apparent as well as strong similitudes. Finally, D. Bernstein also claims the cube attack is in fact based on the work of X. Lai [74].

[14] Finite Field: Also referred to as a Galois Field and in the cube attack is noted as GF(2).

5. Applied Cryptanalysis to known algorithms

5.1 Introduction

We will now discuss how this theory can be applied in practise and what are the implications in the real world. This chapter will be focusing on four specific attacks on high profile cipher implementations, it is divided in two parts with a common structure: a background of the cipher implementation, the type of attack used, an overview of this attack and finally the implications of that attack. Each part will focus on a stream and a block cipher.

The first part deals on old implementation and successful attacks of RC4 and DES which have often been very well documented. The second part takes a more in-depth look at current attacks on Bluetooth and AES; although those attacks are not yet practical, they are improving each year (or month as recently seen with AES!) and we extrapolate the possible implication of a successful attack.

5.2 Past Attacks

5.2.1 Stream Cipher - An attack on WEP

Background

Wired Equivalent Privacy (WEP) was designed to protect wireless network communications with a similar level of confidentiality found on wired networks. It is based on an implementation of RC4 which was originally created by Ron Rivest in 1987 but kept secret until 1994. It was subsequently implemented with serious security flaws in WEP by the IEEE standards group in 1997 [75]. As early as 2001 various successful attacks were discovered, details of RC4 and all those related attacks can be found in M. Stamp and R. Low's book on cryptanalysis [76].

Type of Attacks

Some attacks are possible on the WEP integrity checks (CRC) to modify valid encrypted traffic without knowing the key; we will however focus on the higher profile attacks used to recover the keys: *statistical*, *plaintext, related key and correlation attacks*.

Attacks Overview

One of the possible attacks is a *statistical attack*. The WEP Protocol combines an Initialisation Vector (IV) to the RC4 initial key in order to avoid repetition while encrypting a communication. This is an attempt to create something similar to a one time pad but instead provides a way of attacking its security by taking advantage of the 24 bits size of the IV and the fact it is sent in plaintext over the network. Indeed, 24 bits is not long enough to avoid repetition on a busy wireless network and if we intercept enough IVs and their related ciphertexts we can start building statistics when repetitions occur and attack the keystream.

If we are able to force a *known plaintext* to be sent encrypted and intercept the corresponding ciphertext and IV we are then able to decrypt any ciphertexts encrypted using that IV. This can be achieved by initiating a known network

communication with a person on that targeted network for which we know the content, for example, an instant messaging conversation, an email, a web page being accessed, etc.

A more effective attack is a *related key attack* which was first described by A. Shamir et al. in 2001 [77] and is often referred as the FMS attack. They identified a weakness in how the IV was used to initialise the RC4 keystream, regardless of where the IV was appended (before or after the key). The main idea is to use the fact the initial plaintext bytes encrypted in every packets are predictable (protocol headers, etc) and because the IV is sent in plaintext, the first 3 bytes of the keystream are also known; this information can then be used to recover part of the key. A full key recovery with this technique requires between 4 to 6 millions packets to be intercepted for a success probability superior to 50%.

Finally, some *correlation attacks* are possible. A flaw exists in the RC4 Pseudo Random Generator used where the first byte generated is not really random and some correlation exists between its output bytes, in such a way that 2^{26} of those bytes act as a distinguisher from a true random sequence [78]. Knowing this and by assuming the first byte of the IV is the first byte of the key being attacked, the key can be retrieved by monitoring the communication until that assumption becomes true. A more recent and practical correlation attack was described by A. Klein [79] where a combination of correlations between the key and the different internal states used to generate the keystream with statistical analysis techniques are used to reduce the number of ciphertexts/IV to be captured for a successful attack. One of the latest attack to date is based on A. Klein's work and was presented by E. Tews et al. in 2007 [80] by providing a tool called Aircrack-ptw [81] to break WPE keys under a minute.

Implications

Successful attacks on this protocol had very early implications; the first one was for the IEEE standard group to urgently come up with a new standard in 1999 called WPA[15] and another improved version using AES called WPA2, in 2004. This meant for many users a change of hardware or at the very least a firmware upgrade of their existing vulnerable wireless kit.

However, even in 2009, WEP is still commonly used mostly for wrong assumptions of the need for backward compatibility with old hardware. More worryingly, this is the case for many major corporations with sometimes devastating consequences as seen in the recent T. J. Maxx incident [82] where the credit/debit card details of 45 millions of their customers were stolen. Although no official explanation was given, many security experts believe they were the victim of a successful "wardriving[16]" attack on their weak wireless security protocol in use, namely, WEP.

[15] WPA using the Temporal Key Integrity Protocol (TKIP) can now also be attacked.

[16] Wardriving refers to a type of attack where the attacker sits in his car sometimes miles away from the victim and uses a laptop with freely available tools, such as the one referenced in this section, to break WEP and gain access to the network.

5.2.2 Block Cipher - An attack on LM Hashes

Background

LAN Manager Hash was designed by Microsoft as one of the formats they use to store passwords on their Windows Operating System. It works by converting a 14 characters or less user password with only letters and digits to uppercase, if required it can be null-padded to 14 bytes. It is then split into two 56 bits chunks which are used to create two DES keys. Those DES keys will be used to encrypt always the same know plaintext "KGS!@#$%" which is 8 bytes long. The two resulting ciphertexts, 64 bits each, will then be concatenated to create a 16 bytes hash against the user ID. A correct password for that User ID will be able to reproduce the corresponding hash of the known plaintext string.

Type of Attack

This design is vulnerable to *Rainbow tables* and was actually used by P. Oechslin in 2003 as an example on how to use rainbow tables when he presented his concept.

Attack Overview

The following numbers have been taken from the example given in C. Swenson's book [83]

Because the user is forced to only use letters and digits which are then converted into uppercase it reduces the possible initial password characters to 36. Furthermore, encrypting the same 8 byte long plaintext means we have a total of 36^7 ciphertexts that can be generated for that plaintext. If we were to generate all possible ciphertexts to conduct some kind of dictionary attack, apart from the fact it would take a very long time, we would require about 583^2 Gb of storage making this attack difficult to implement. A more practical way is to use a Rainbow Table attack with five tables, each with 35,000,000 lines and 4,666 columns taking a maximum of 1.4 Gb of memory and considerably less time to calculate. With such tables and using a software such as Ophcrack [84], it is possible to crack an LM Hash in a few seconds. The details of the attack are available in P. Oechslin's paper discussed in section 4.4.3

Implications

Passwords stored as LM Hash can be broken by anyone who gets hold of them thus defeating the purpose of encryption. Microsoft introduced replacements for LM Hash in the form of NTLMv1, NTLMv2 and Kerberos. However, LM Hash is still used today for backward compatibility reasons and by many large corporations. This implies an added security risk for such companies until they disable this default windows password storage setting.

5.3 Emerging Attacks

5.3.1 Stream Cipher - An attack on Bluetooth

Background

Bluetooth [85] is a wireless protocol created in 1998 and currently used for short distance communication in laptops, mobile phones, etc. Its security architecture is based on a key agreement protocol where a *pairing* between two devices can occur. A number of keys need to be generated, first an *initialisation key* will be created from a PIN and a random number. It will be used to create a *link key* that acts as a temporary key between the two devices to conduct an authentication using an algorithm called E_1. An *encryption key* will be generated, shared by both devices and used onwards to encrypt communication with the E_0 algorithm.

Type of Attack

Although attacks have been proposed against the pairing phase of the protocol, we will focus on attacks attempted on the stream encryption cipher, E_0. There are two types of attacks in theory possible, *algebraic attack* and *correlation attack*.

Attack Overview

Although E_0 is infamous for a number of vulnerabilities, most of the known attacks on that cipher do not apply to Bluetooth due to how it was implemented and mainly because the keystream of 2745 bits used in this implementation is too small to be attacked efficiently. However, progress is being made and a practical attack may even already be available against that specific implementation of E_0.

A promising type of attack is an *algebraic attack* described by N. Courtois and W. Meier in 2003 [86] and which was summarized by D. Singelée and B. Preneel in 2006 [87] by saying E_0 was "vulnerable to algebraic attacks because of the possibility to recover the initial value by solving a system of non-linear equations of degree 4 over the finite field GF(2). This system can be transformed by linearization into a system of linear independent equations with at most 223 unknowns". This is however not practical because Bluetooth uses small packets for its communication and the attack would require a long key stream.

Another type of attack is a *correlation attack* which was first described in 2002 by J. Golic[17] et al. [88] and claims to be able to recover the 128 bit encryption key with a complexity of about 2^{70} but requires too many packets to intercept to be practical. In 2005 however, a *conditional correlation attack* was proposed as practical [89], this type of attack allows for correlations to be found between input and output bits on a stream cipher using nonlinear functions. With a known plaintext attack it was deemed practical, but since that announcement, very little has been done to prove them right, or wrong for that matter.

[17] Although the paper's title starts with "A Linear Cryptanalysis of Bluetooth" it describes a correlation attack.

Implications

As mobile technology is now almost everywhere, so is Bluetooth. If a practical attack was to be widely available then its implication may be relevant to many people. This is especially true for mobile phones where Bluetooth is used to connect the device with headsets, computers, etc. Breaking the pairing protocol would make it possible for someone to hijack a victim phone and either to steal its content or use its connection. Breaking the encryption protocol would not only allow that but also eavesdropping of communication.

At a time where mobile phones are becoming more and more intelligent and versatile this could become a major issue in the future. Especially when services like wireless payment are being considered by banks such as Barclays [90]. The importance of Bluetooth security is being recognised and NIST recently provided a security guide to that effect [91].

5.3.2 Block Cipher - An Attack on AES

Background

The Advance Encryption Standard (AES) was published by NIST as a replacement for DES in late 2001. It followed a 5 year process where public submissions were considered and a cipher named *Rijndael* was finally selected [92]. The standard lists three implementations of the cipher, AES-128, AES-192 and AES-256. All three use a 128 bits block but with a key size related to their names. It is the first time a public algorithm has been approved for the protection of TOP SECRET classified document by the NSA and has become widely popular.

Type of Attack

No practical attacks against this cipher are known to date, but an increasing number of them are now getting close to become practical. Recent attempts using *related key boomerang attack* techniques have received a lot attention.

Attack Overview

At the very end of May 2009, a paper was published by A. Biryukov and D. Khovratovich [93] describing a potential attack on AES based on a *related key boomerang attack* as mentioned in section 4.8.7. Although not currently practical to break AES it was the first attack to be more efficient than pure brute force by lowering the AES-256 complexity from 2^{256} to 2^{119} and AES-192 complexity from 2^{192} to 2^{176}. Because there is no known attacks on AES-128 to lower its 2^{128} complexity, AES-256 has become theoretically the weaker implementation of AES. They enhanced their attack by using *boomerang switching* techniques and a fairly new concept for block ciphers of finding *local collisions* to find their boomerang differentials, which is a technique derived from the cryptanalysis of hash functions.

Shortly after this paper was published another major breakthrough in the cryptanalysis of AES was made public in August 2009 [94] by an extended team responsible for the first paper; and this time it is almost practical against some variants of AES-256. Respectively using a 9 round and 10 round variants they lowered the complexity to 2^{39} and 2^{45}. As pointed out by B. Schneier, the standard implementation of AES-256 uses 14 variants and this type of attack requires access

to “plaintexts encrypted with multiple keys that are related in a specific way”, hence for this attack to succeed specific conditions have to be met and are not yet cause of concern for most implementation of the cipher. Nonetheless, this is the closest AES has come to be broken yet and some cryptographer and security specialist are now advising for the use of AES-128 instead of AES-256.

Implications

As opposed to *differential cryptanalysis* where we suggested the NSA knew about it long before the public, it has been suggested this *related key boomerang attack* may not have been known to NSA as it is chipping away its recommended block cipher’s security. If all AES implementations were to be practically broken, the implications would be tremendous as it has now been used in a very wide range of protocols and applications, such as WPA, banking related applications, password storage, etc. and it could impact their security integrity or confidentiality. Although we are yet far from the need of a new standard, the previous process that selected AES also provided the world with 2 other finalist ciphers deemed as very secure and, if required, the search for a replacement may not take as long as it did for DES. Finally, It is interesting to notice to speed at which those attacks are being published, it may be an indication of a more public awareness on cryptanalysis activities as well as a larger number of researchers working in that field.

6. BUGS Cipher

6.1 Introduction

This cipher, BUGS, started as my personal project in 1995 and a first version was published in 1996 while studying at a French computer school in Lyon [95]. Pressure from the French Domestic Secret service, the DST, forced me to remove my algorithm from the public domain.

Crossing the channel for my studies allowed me to resume work on it as part of my UK Final Year BSc project at Teesside University [96]. I finished its first major redesign in 1997.

By 1999, the cipher started to attract quite a lot of attention by being featured in some magazines and an IEEE newsletter [97]. One person especially, spent a lot of times trying to break the algorithm. That person, who I will just name "Simon", introduced himself as a bored teenager and over a few months showed a very high level of understanding of the algorithm itself, highlighting a number of weaknesses and translating most of the algorithm in pure assembler for efficiency and test purposes. I was responding to each weakness highlighted by the unusually technically talented and politically opinionated teenager, by spending days and night working for an improved algorithm design. This resulted in 10 intense months of work and the latest version of the algorithm (v4.x); as well as the end of the communication with my "muse" who suddenly had personal problems and could no longer spend time on my algorithm.

As much as I would like this "Simon" to be more than just a talented teenager, this is more than unlikely and just an interesting anecdote.

No known cryptanalysis has been done on this version of the cipher. Its creation was based on logic, not mathematics and by an amateur who had just finished reading Bruce Schneider's Applied Cryptography book [98]. Finally, I have never done any cryptanalysis of any cipher or known about the techniques I have described in the previous chapters before writing this thesis.

6.2 BUGS Cipher's Concept

The BUGS Cipher has two main cryptographic primitives, a *Key Scheduler* and a *symmetric-key encryption function*.

It uses an Initialisation Vector (IV) to dynamically change the behaviour of both primitives. It is a highly configurable cipher where the user can decides different default settings which can also be set to dynamically change. For example, the number of round is set to two by default, but it is possible to specify a different value on execution, i.e. ***R=5***. We can also allow or not the algorithm to dynamically change that base value depending of the IV.

The *key scheduler* is designed to extend or stretch the key. In the symmetric-key encryption function, it is used to generate key buffers in an attempt to diffuse repetition.

The *symmetric-key encryption function* has two sub-functions, only referred from then on as *encryption functions*; they can be used independently or combined one after the other. The first of those encryption functions is similar to a block cipher used in a stream cipher mode while the second is similar to a chain block cipher.

The next two sections will present an overview of those different cryptography primitives. For detailed and implementation diagrams please refer to Appendix C.

6.3 Key Scheduler

The Key scheduler has three Phases: Initialisation, Scrambling and Randomisation.

The *initialisation phase* requires an Initialisation Vector (IV) and the size N_s of the subkey the user wants to generate. The IV can either be an initial key or a random number both of size N_i such as $N_s/2 <= N_i <= N_s$

- It starts by padding the IV if required to the size N_s
- It concatenates the **bytes** together
- It generates a pseudo random number (PRN_1) by combing all the IV **bytes** together with a XOR
- It adds a different derived PRN_1 to each IV **bytes**
- It defines a number of variables:
    - Operation, ***O*** (Swap or Logical Ops) = $byte_1$
    - Direction, ***D*** (Left or Right) = $byte_2$
    - Modulo size, M_0 (Big or Small) = $byte_3$
    - Nb or Round, ***R*** (max x2 value specified by user) = $byte_4$

This process generates the first subkey, SK_1

The main part of the key scheduler is the *scrambling phase*. Using SK_1:

- It generates a session modulo, $M_s = byte_1 \oplus byte_2 \% M_0$
- It generates a Shift Window, $SW = byte_1 \% M_s$
- Depending of the value of ***D***, the following may start from the beginning or the end of SK_1
- Between the Starting bit, ***i***, and ***i + SW*** do an operation on those 2 bits.
- The operation will depend of the value of ***O*** and can be a bit swapping or a choice of 5 logical operations
- Change the value of the modulo M_0 from *small* to *big* and vice versa.
- Change the value of the operation ***O*** from a swap to a logical op and vice versa.
- Change the direction ***D*** from left to right and vice versa
- Repeat for ***R*** number of round

This process generates the second subkey, SK_2

The last part of the key scheduler is the *randomisation phase*. Using SK_2:

- It Generates a Random key, ***RND***, either using the system random generator or the ISAAC algorithm [99].
- Using $byte_1$ to select the index ***i***, then $byte_i = byte_i \oplus RND$
- It uses a Linear Feedback Shift Register on RND to generate a new ***RND*** value
- XOR than new RND value to another **byte**
- Repeat the previous two points until all **bytes** have been “randomized”

The result is the key scheduler’s output, the encryption Key ***Ke***

6.4 Symmetric-Key encryption function

It consists of an *Initialisation* phase and two Encryption functions: *Seeding* and *Shuffling*. It can operate in different modes where the encryption process uses one or both of the encryption functions. If using both, then the seeding phase is done first and the resulting seeded ciphertext is used as an input into the shuffling phase.

The *Initialisation* phase consists of the following:

- It breaks the plaintext into blocks of size defined by the user. The block size can be as large as the plaintext itself of a multiple of the key size used for the encryption.
- An IV is used to define the key buffer size, ***S*** (with a minimum of 16)
- A random number, ***RN***, is generated for the encryption, XORed with a key and inserted in a pseudo random position into the final cipher text.

The *Seeding* phase is similar to a *Block cipher used in a Stream cipher mode* where a keystream is applied to the plaintext blocks in a pseudo random sequence.

- It uses the Key Scheduler to generate ***S*** number of encryption keys, $\boldsymbol{Ke_i}$, and stores the keys into a Key Buffer table, ***KT***
- It pseudo randomly chooses two encryption keys from ***KT*** and by XORing them it generates a new encryption key, $\boldsymbol{Ke_{seed}}$, which is then XORed with a pseudo randomly chosen plaintext block, then labelled as "*seeded*".
- $\boldsymbol{Ke_{seed}}$ is then used as an IV into the key scheduler to generate a new $\boldsymbol{Ke_i}$ and replace one of the two encryption key used above into ***KT***
- The process is repeated until all the plaintext blocks have been encrypted with a XOR, the result is a Seeded Cipher text, ***sC***.

The *Shuffling* phase is similar to a *Chain Block Cipher* as described below.

- It pseudo randomly chooses two plaintext block and by XORing them it generates a new encryption key, $\boldsymbol{Ke_{shuffle}}$, which is then XORed with another pseudo randomly chosen plaintext block. That latest plaintext block is labelled as "*shuffled*".
- That process is then repeated with:
    - One of the plaintext blocks used to create $\boldsymbol{Ke_{shuffle}}$ is selected as the next block to be encrypted
    - Once a plaintext block has been shuffled it cannot be used to generate a new $\boldsymbol{Ke_{shuffle}}$
    - And until all the plaintext blocks have been encrypted but two.
- Two blocks must remain "*unshuffled*" to be able for the recipient to reverse the encryption process. They will therefore simply be XORED with a key of the form $\boldsymbol{Ke_{seed}}$ as defined previously.
- The result is a Cipher shuffled text, ***Cs***.

7. BUGS Cryptanalysis

7.1 Introduction

The first step is to attempt to normalise the representation of the cipher algorithm to help confirm what type of cipher it is, its potential design flaws and identify what cryptanalysis attacks may apply. Because it is a complex cipher which is not using cryptography primitives in a standard way, we will attempt to describe that algorithm using standard cryptography nomenclature; in essence mapping the different part of the cipher to known cryptography primitives or their closest relatives. We will then be able to identify relevant attacks and will start discussing their possible implementation.

Its key size is configurable to be as large as an integer can be. However a too large key size would not be practical neither for the user or the cryptanalyst. We will therefore only consider the default key size of 128 bits, and a pure brute force attack would then have a complexity of 2^{128} on its default mode of operation.
Also, in this attack we will not focus on the padding and will assume the original key and Initialisation Vector used in the key scheduler has the same size as the output key and does not require any padding.

It is important to note that all of the attacks mentioned in this section are only proposed attacks, they are not proven to work and may have errors. However, they illustrate the thought process followed during the cryptanalysis of this cipher.
Also, in order to gather statistical data, a series of scripts were designed with the help of T. Martinez [100] for this thesis and are available in Appendix A.

From the description of the cipher given in the previous sections it appears to be a combination of block cipher, stream cipher and chain block cipher. We will now focus on the normalisation of the two main components of the cipher algorithm, the key scheduler and the encryption function.

7.2 Key Scheduler Normalisation

The Key scheduler normalised below appears to use two Linear Feedback Shift Register functions (LFSR) one after the other and as such could be summarized as a standard LFSR key scheduler albeit a rather complex one.

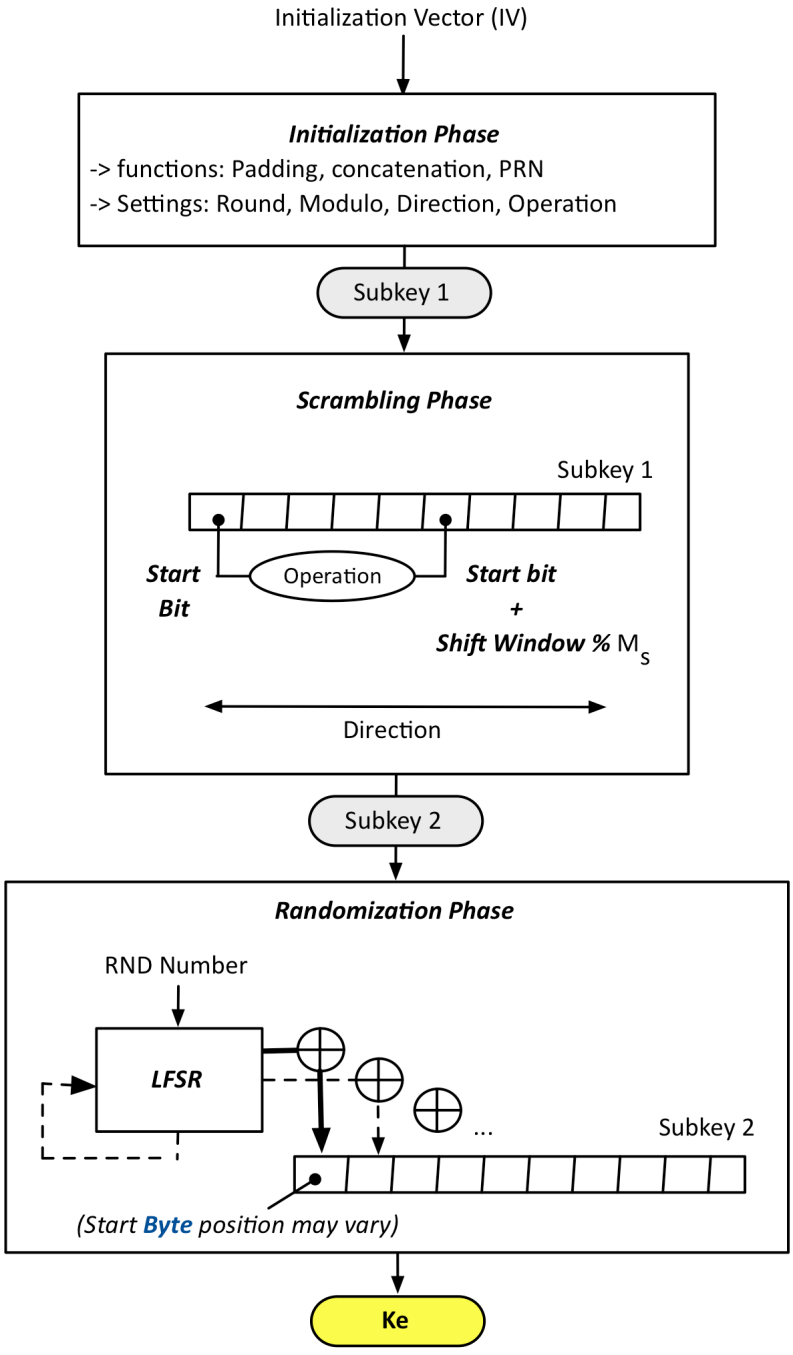

Figure 11 - BUGS Key Scheduler

7.3 Encryption function Normalisation

The key scheduler used can be represented as an SBOX in Linear Feedback mode (LFS-Box) as illustrated in Figure 12 below.

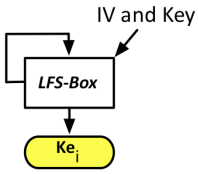

Figure 12. Linear Feedback S-BOX

The seeding function will use the LFS-Box as described in Figure 13. Where a 16 key buffer is generated and one of those key subsequently replaced one at a time.

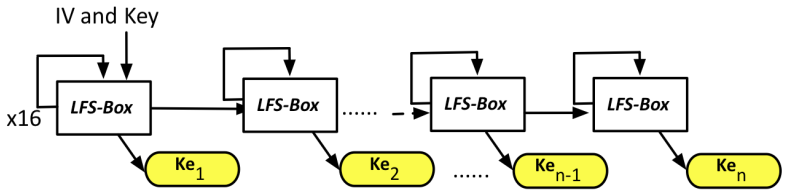

Figure 13. Linear Feedback S-BOX used in the Seeding function

In its default mode, the BUGS cipher uses the *seeding* and *shuffling* functions. An attempt to normalise the full cipher operation is illustrated in Figure 14, where the pseudo random selection of the Plaintext block being XORed during the seeding phase has been represented by a Permutation Network. That Permutation network could only be represented as a P-BOX as it dynamically changes with the IV. The shuffling phase uses a modified version of the key scheduler, LFS-BOX, to generate the final ciphertext represented as the *Seeded Cipher Shuffled Text* with each ciphertext block is labelled ***sCs***

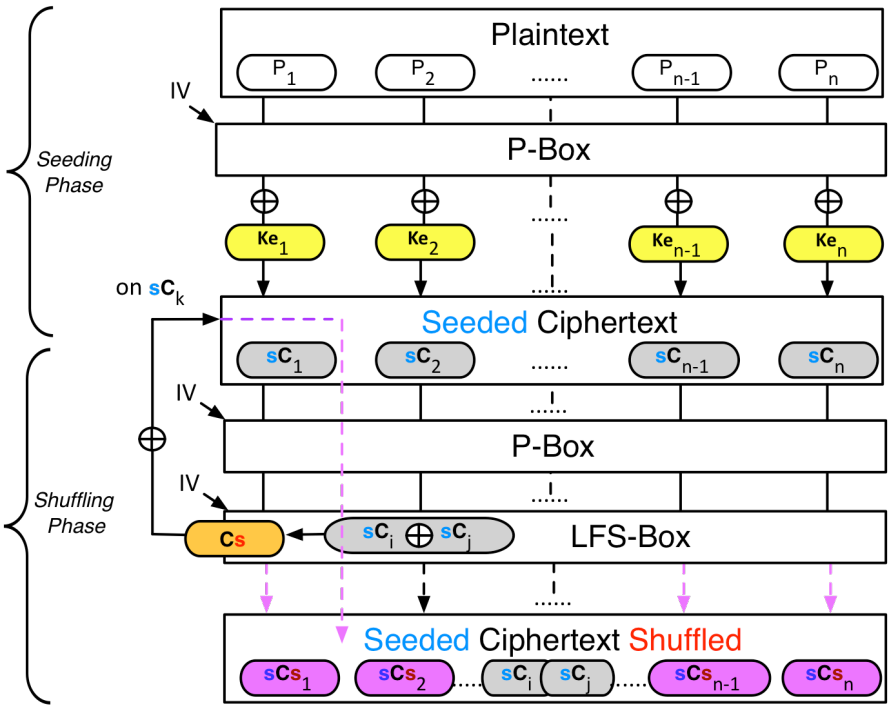

Figure 14. BUGS Encryption Function in default mode

7.4 Attacks Selection

Attempting to normalise the algorithm allows us to view it in a more global and simpler way compare to the first specifications given and the diagrams available in Appendix C. Because the algorithm is complex and combine different type of ciphers' characteristics, some simplifications had to be made; such as the introduction of the P-BOX and the LFS-BOX, thus the details of those operations are not shown in the previous diagram.
However, it makes it easier to map certain type of attacks that may be successful on that algorithm, especially when considering Linear Cryptanalysis.

Two targets can be considered for our attacks, the key scheduler alone[18] or the encryption functions. Each of those targets has two areas of cryptanalysis that apply:

- Algorithm conception cryptanalysis
    - Algorithm review to identify conceptual flaws
    - Diffusion and confusion analysis of the ciphertext to identify obvious potential weaknesses
- Known cryptanalysis attacks
    - Specific type of cryptanalysis attacks adapted for this cipher

For this thesis we will focus on the encryption functions as a target and the following attacks will be discussed:

- A *Statistical analysis* of the cipher to identify the *Diffusion and Confusion* characteristics of the ciphertexts and if there are any inherent weaknesses.
- Because of the XOR properties, it is likely a *known plaintext* attack on the shuffling function would work.
- A *Chosen plaintext* attack on the same function should also work for the same reasons and could be used to speed up the attack.
- How to build a *Linear Cryptanalysis* framework for this cipher.

[18] The software package created with the BUGS algorithm uses the key scheduler alone for its password application.

7.5 Statistical and Probabilistic attack

For this attack, two tests were conducted to analyse the ciphertext bits *confusion* and *diffusion* properties. Both tests used the secret key "angleterre" to encrypt the same plaintext, the BUGS cipher package's History file of 24 Kb, using the default cipher's mode of operation. This meant both encryption functions were used, the seeding and shuffling functions.

7.5.1 Confusion Analysis

A script was designed to take a ciphertext input and generate the following:

- To count the number of times the same hexadecimal value was found.
- To count the average distance between repetitions.

Two tests were conducted on the same ciphertext, the first one to search for hexadecimal values repetitions with an alignment on every byte and the second test with an alignment on every bit. A summary of those results is displayed in the Table 6 where the top 10 most frequent values are highlighted in pink and the lower 10 in blue. The full results are available in Appendix B.

Value	1 byte Alignment	Average Distance		Value	1 bit Alignment	Average Distance
FF	194	976.78		**FF**	1332	141.85
DF	159	1200.36		7F	1157	163.47
EF	156	1207.51		FE	1157	163.47
FE	150	1269.17		DF	1120	168.75
FB	148	1279.27		EF	1113	169.68
BF	146	1293.73		F7	1101	171.71
F7	146	1302.88		FD	1099	171.89
F4	145	1296.55		BF	1091	173.38
CF	143	1336.80		FB	1088	173.78
DE	142	1301.35		F3	973	194.61
65	55	3094.8		2	599	316.87
66	55	3454.4		4	598	316.79
26	54	3533.34		81	597	316.85
64	51	3713.88		40	597	317.06
54	51	3719.64		9	594	318.67
44	50	3817.12		29	593	320.13
68	49	3914		90	590	321.39
35	48	3825.38		20	587	322.46
61	48	3898.91		52	583	325.00
6A	45	4242.27		4A	580	327.64

Table 6. BUGS - Hexadecimal repetitions

The results of the first test show a high repetition for the value ***FF***, which could be an indication of a cipher characteristic, in which case it may be possible by analysing the repetition pattern of large number of plaintexts to derive information from the key

used. The second test results are interesting, although ***FF*** is still the most found value many other values also have a high repetition count. This may indicate that the first results were actually not a sign of bad confusion. The average distance results did not highlight any characteristics for any values.
A further test was conducted this time with the same plaintext encrypted with a different cipher, using the standard mode of operation of GPG [101], and with a byte alignment. The results displayed in Table 7 show what looks like a much better confusion.

Values	1 byte Alignment	Average Distance
44	59	153.37
36	52	180.96
Fe	52	180.80
fc	49	192.45
d0	49	192.41
9e	49	186.22
aa	49	185.04
67	48	184.63
31	47	190.08
64	47	195.30
19	27	323.80
ad	27	317.46
a2	26	366.36
9a	26	342.12
23	25	386.37
a5	23	350.31
fb	23	397.40
c0	22	362.33
d8	21	448.35
d3	19	456.33

Table 7. PGP - Hexadecimal repetitions

Finally, to confirm or not the FF value characteristic on the BUGS cipher, further tests would be required. For example, different keys with specific values could also be used such as:

- A key with all the bits sets to 1
- A key with all the bits sets to 0
- A key with half the bits sets to 1 and the other half to 0

7.5.2 Diffusion Analysis

Another script was designed to display a file as a picture with the following rule: to read the file as a bit stream and each time a 1 was found then a black pixel would be marked on the picture and for a value of 0 a white pixel will be used. With a parameter to add a new line in the picture every 512 bits, the following files were created as a graphical representation of:

- The plaintext as shown in Figure 15. A pattern is apparent, which is to be expected as this is a text file which only uses a certain set of ASCII character and as such has a repeated set of bits.
- The ciphertext as shown in Figure 16. Where no pattern is apparent. For comparison an image was also created from a random stream of bits (/dev/random) and the picture result was similar.
- A difference between the plaintext and ciphertext as shown in Figure 17. The white pixels indicate the different pixels between the two files. No pattern is apparent either. A bad results would have been if parts of the picture were either all the same or all different.
- The GPG ciphertext generated for the previous test was reused in Figure 18. It shows a similar level of diffusion as BUGS. Because it produces a smaller ciphertext the picture appears smaller.

Although no pattern is an indication of a good diffusion for the cipher it does not prove it is secure. It just proves it is not apparently vulnerable to a bad diffusion attack.

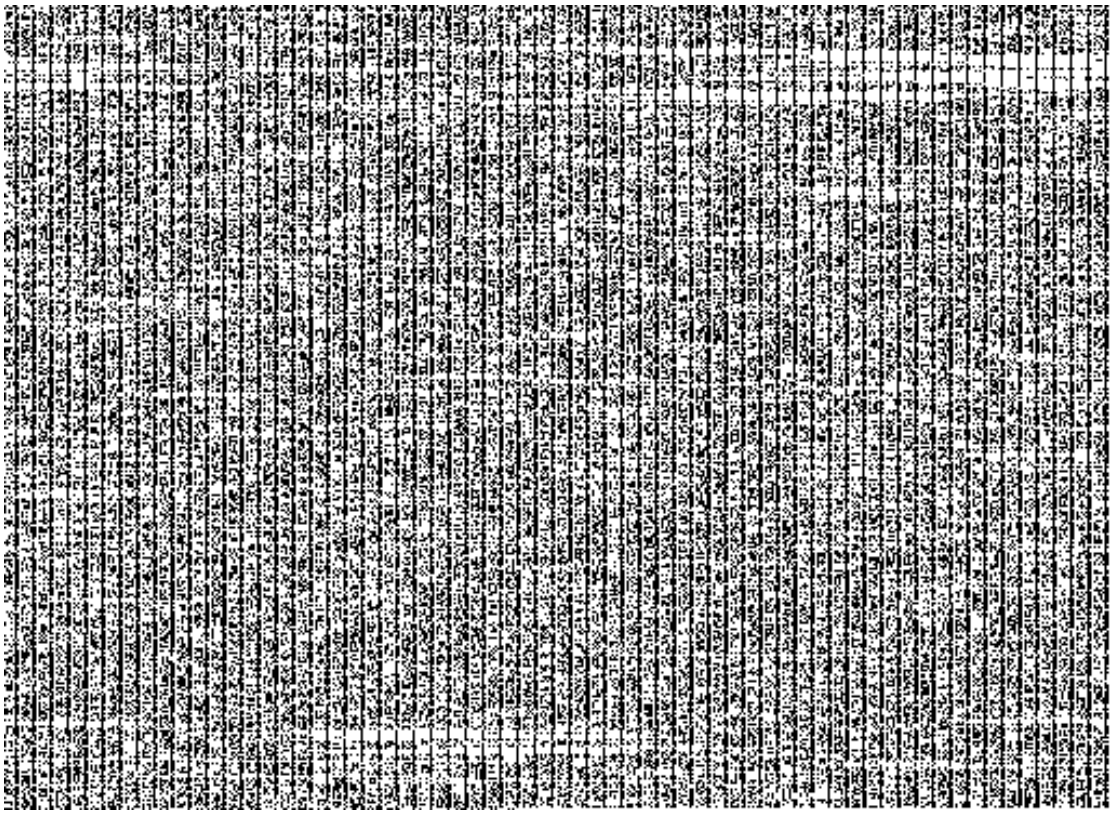

Figure 15. Plaintext Diffusion Representation

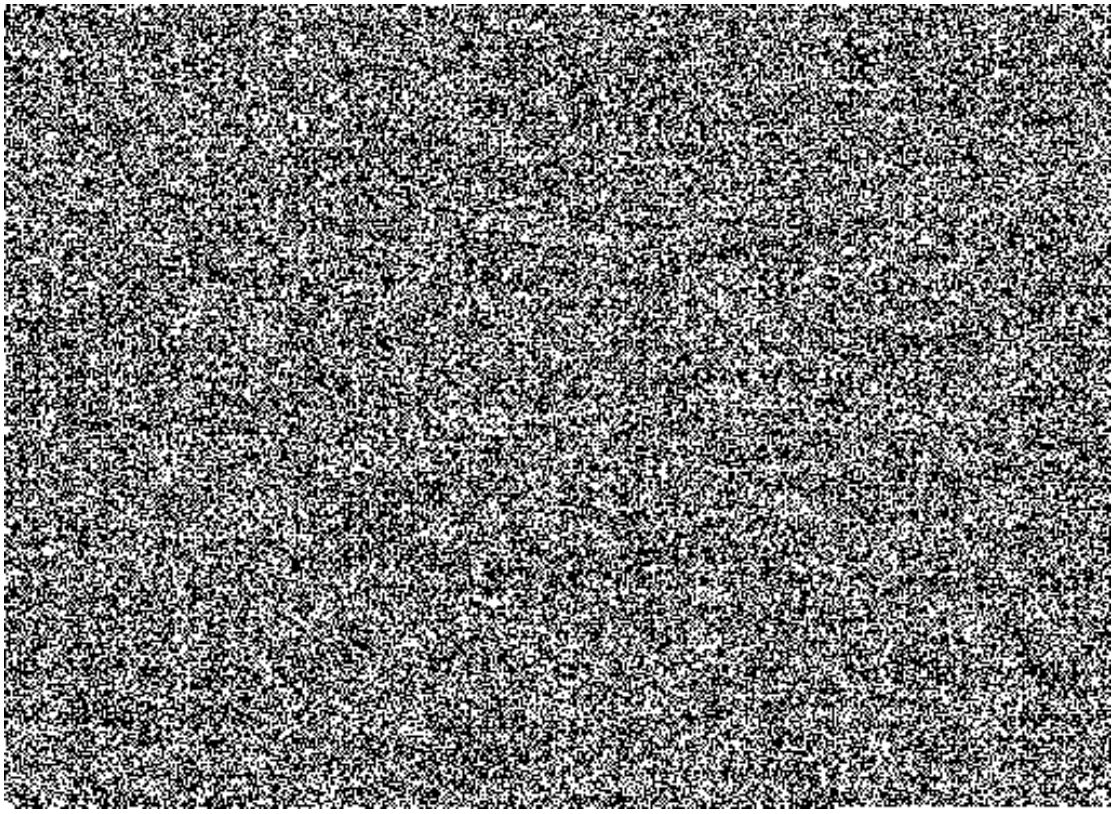

Figure 16. BUGS - Ciphertext Graphical Representation

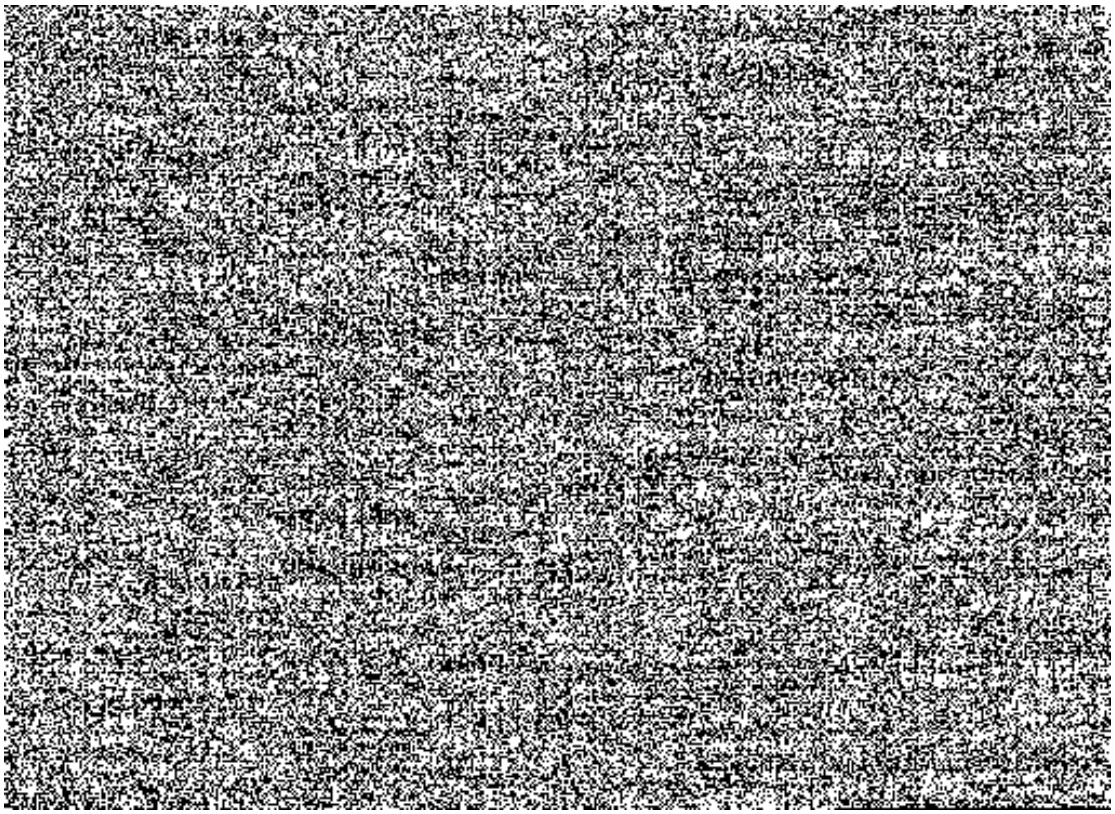

Figure 17. Plaintext/Ciphertext Difference Graphical Representation

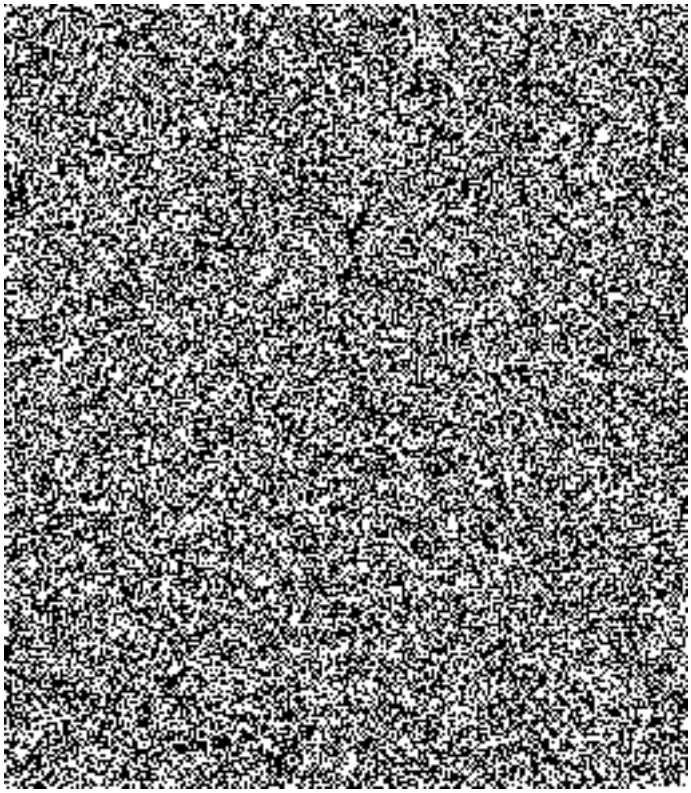

Figure 18. GPG - Ciphertext Graphical Representation

7.6 Known plaintext attack

If we are using the Shuffling function alone, then a known plaintext/ciphertext attack is possible.
This is because the cipher combines two plaintext blocks to encrypt another. By intercepting a ciphertext if we XOR the corresponding plaintext we are left with the combined plaintext blocks or XOR-sums, in other words, all the $\boldsymbol{Ke_{shuffle}}$ used.
We also recover the last two keys ($\mathbf{sC_i}$ and $\mathbf{sC_j}$) used to encrypt the last two blocks of plaintext left unshuffled but we don't know which two blocks represent those two keys. Knowing all the values of the $\boldsymbol{Ke_{shuffle}}$ is useful, because we can now attempt to combine in pairs all the original plaintext blocks until we find a matching $\boldsymbol{Ke_{shuffle}}$.
Once we find a match we would have recovered which blocks were used to create that specific $\boldsymbol{Ke_{shuffle}}$.Doing this for all the keys allows us to reconstruct the sequence used during the encryption process to generate all the $\boldsymbol{Ke_{shuffle}}$, because this sequence is dependant of the IV we would gather information to help us derive the original IV.
However we would need to take false positives into consideration as two different pairs of plaintext block could generate the same value. Also, because two of the "combined blocks" are in fact the keys $\mathbf{sC_i}$ and $\mathbf{sC_j}$, there should be no plaintext block pairs with those values; but it can happens and if it does it would add to the false positives .

7.7 Chosen plaintext attack with restricted input elements

This is an extension of the previous attack by choosing the plaintext to be encrypted. The idea is to find a way to populate the plaintext with specific values for which it would be easy to extract which individual plaintext blocks were used to created all the $\boldsymbol{Ke_{shuffle}}$

A solution would be to have values producing unique paired XOR-sums but this is generally not the case as shown below where two different pairs produce the same XOR-sums:
$10110 \oplus 10000 = 00110$
$00100 \oplus 00010 = 00110$
To solve our problem, we can restrict plaintext values to be power of twos. This implies that only values whose representation in base 2 is of the form shown below are allowed:
S_0={1, 10, 100, 1000, …etc}
If we XOR two of values from our new set S then we have for example:
$1000 \oplus 100 = 1100$
And only 1000 and 100 could have been the values to have a XOR-sum of 1100
We will call this property a *XOR-sum uniqueness* property from now on

This allows us to reconstruct the full XOR sequence used during the encryption process in a very fast time. As before, because this sequence is dependant of the key used to encrypt the plaintext we would gather information that could be used to recover the key.

It is interesting to note that our very simple rule, used to create the S_0 set mentioned above, is also very restrictive in the number of unique values that can populate S_0 for a given bit length. Therefore, the size of the chosen plaintexts will also be governed by the limitation in the number of unique values available for a given plaintext block size. If the attack cipher uses 128 bits plaintext block, then the maximum number of unique values we can generate is 128. This means the maximum size our chosen plaintext could have in this context is: 128 bits * 128 = 2Kb
This has no impact on our ability to reconstruct the full XOR encryption sequence but if more statistics were required to recover the key then it may impact the performance of this attack.

However, while researching this problem it seems there might be different sets which could keep the same properties we are looking for, a XOR-sum uniqueness on two elements of S, but with a larger element population. Indeed, the following elements are of 8 bits size and have that XOR-sum uniqueness property:
S_1={ 1, 10, 100, 1000, 1111, 10000, 100000, 110011, 1000000, 1010101, 1101010, 10000000, 10010110, 10101011, 11011011, 11101101, 11110111, ...}
What is interesting to note is that there are more than 8 elements in this set. These numbers were filtered out by designing a rudimentary computer program designed to brute force all possible set of 8 bits values yielding the XOR-sum uniqueness property. To our knowledge, there is currently no known method to optimally compute those numbers other than using a near optimal “Greedy” algorithm mathematical function [102]. Finding such optimal method to generalize that property is not in scope for this thesis.

Further extension of this technique could yield an even more interesting concept as described in the next attack.

7.8 Unrestricted XOR-Sum Uniqueness Cryptanalysis attack

It is important to note at this point, this is only a theoretical attack which may not be practical.

Building from the previous attack, if the plaintext was not chosen but had been created using elements from S_0={10,100,1000,…} then it would still be possible to attack the corresponding ciphertext without choosing or knowing the plaintext because our XOR-sum uniqueness property still holds true for any number of XOR operations on elements of S_0
For example, if our plaintext is: P={10,100,1000} and the ciphertext is C={1110} we can reconstruct the XOR sequence as follow:
$10 \oplus 100 \oplus 1000 = 1110$
As only 10, 100, 1000 could have produced that XOR. This means we do not have to first XOR the plaintext out of the ciphertext to conduct our attack and in this specific case we do not need to know the plaintext.

This attack is obviously not practical because it is more than unlikely anyone would use this type of elements to produce plaintexts. However, the really interesting aspect of this concept, is to extend it one last step further.

For a practical attack we would need the XOR-sum uniqueness property discussed earlier to also hold true for a set of values created without any restrictions.
However this is not the case, if we XOR two elements from such a set the XOR-sum will not be unique.
Taking into consideration what we have learnt when discussing the different modern cryptanalysis techniques, one attack was created to resolve the following problem:
How to define linear equations from a non linear cipher?
The solution, through Linear Cryptanalysis (LC), was to define linear approximations of the cipher and use probabilities and statistics to attack it.

It may be possible to use the same logic to resolve the following problem:
How to define XOR-sum uniqueness properties on an unrestricted set of elements?
A possible solution would be to define XOR-sum uniqueness "approximations" of the cipher combined with a similar LC technique of using probabilities and statistics to attack it.

By "approximations" we mean the following:
Let E() be an encryption function which takes two plaintext blocks, calculate a XOR-sum and use it to encrypt another plaintext block.
For $P=\{P_0, P_1, P_2\} = \{11110000, 11111111, 10101010\}$
Then $C_0= E(P_0 \oplus (P_1 \oplus P_2)) = 11110000 \oplus 01010101 = 10100101$

The approximation would single out bits on the plaintext and ciphertext so the XOR-sum uniqueness property holds true, in our example a possible approximation would be:
From P= {11**1**10000, 11111**1**11, **1**0101010} and C_0= **1**0**1**00**1**01
Then single out the bits highlighted in Red and consider the following approximation:
P= {00**1**00000, 00000**1**00, **1**0000000} and C_0= **1**0**1**00**1**00
This new set has the XOR-sum uniqueness property we are looking for and is related to the original plaintext and corresponding ciphertext on the bits highlighted in Red.

Therefore, as stated before, using a similar logic as the one used in LC we could calculate the probabilities for this type of approximation holding true, produce some statistics using large number of plaintext/ciphertext pairs encrypted with the same secret key, try all possible subkey values and record when our approximation did hold true. This may indicate probable correct bits values of the secret key.
This technique could be generalized and extended to attack anything being a result of a XOR-sum, such as keys being XORed to plaintext/ciphertext blocks.

It may be possible to refine that technique by studying the *probability distribution* of the possible XOR-sums. It is unfortunately out of scope for this project to investigate that concept further and as stated at the beginning of this section, it may be a completely unpractical attack with no chances of success. On the other hand, if this could work, it would be called an *Unrestricted XOR-sum Uniqueness Cryptanalysis attack*.

7.9 Linear Cryptanalysis attack

A successful conventional Linear Cryptanalysis (LC) attack gives a partial view of the second to last subkey used to generate a ciphertext. From that subkey the original key can be recovered with a combination of LC and brute force attack. This implies the key scheduler is not a one way function.

The BUGS cipher uses a key scheduler which is a one way function. As such even if we were able to recover parts of the last subkey used, or even a complete subkey for that matter, it would still be very difficult to attack the original key.

However, there might be some weaknesses in the Key Scheduler; hence we should still consider a LC attack on this cipher as if we could prove some of the input bits have certain relationships with some output bits, some information could be derived about the original key used.

It is difficult to conduct a conventional linear approximation of this cipher because of the following reasons:

- The P-BOX and LF-SBOX settings change every time a different original key and IV is used
- It combines three type of ciphers (Block, Chain Block and Stream Cipher)
- The permutation table occurs on block of plaintexts and ciphertexts as opposed to bits.

The encryption function normalisation illustrated in Figure 14 was actually designed with Linear Cryptanalysis in mind, as a result it will help us define a Linear Cryptanalysis framework for this cipher. The idea is not to focus on bits but instead on whole block/keys and apply a similar Linear Cryptanalysis technique, as highlighted in the Figure 19, where we draw the following linear approximation of the cipher in blue: $P_1 + P_2 = sCs_1 + sCs_2 + sCs_{n-1}$

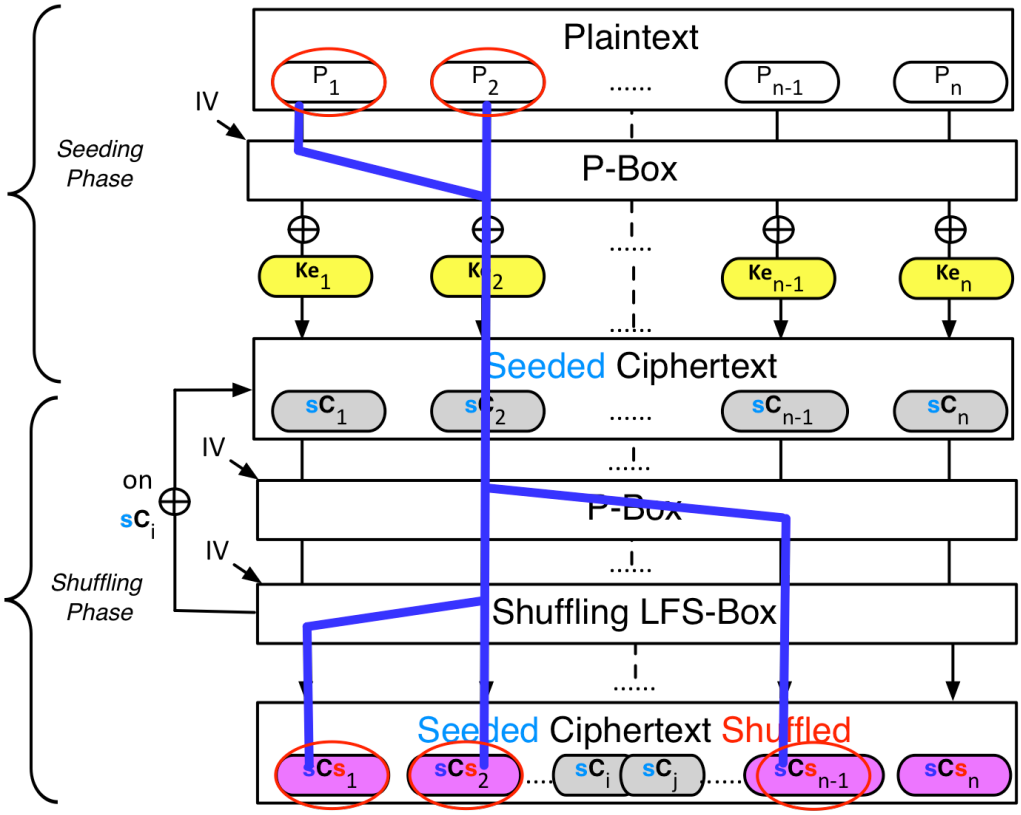

Figure 19. BUGS Cipher High-Order Linear Approximation

This is obviously different from conventional LC where we focus on the second to last round subkey. With the BUGS cipher, the only such round occurs in the seeding phase when generating the keystream. Instead we could try to identify if for the SUM of whole plaintext block values there is a relationship with the SUM of the corresponding cipher blocks. Calling this attack a *High Order Linear Cryptanalysis*[19], its inspiration was taken from the *High Order Differential Cryptanalysis* technique. Even if by itself it does not provide direct information on the key, it may provide some relationships between the input and output bits and further research on how this could be exploited might be interesting.

[19] While doing the research no mention of that type of attack could be found, hence we are assuming this is right name.

7.10 Findings summary

Those attacks should be considered as an introduction to the BUGS cipher cryptanalysis as some interesting results were found which may help gather further information on its characteristics.

The first attack results indicated that the cipher may have a characteristic of bad confusion and further tests would be required to confirm that.

The second and third attack targeted the *shuffling* encryption function. Although it is not used alone in the cipher default mode of operation, it is available as a standalone option mode and appears to be vulnerable to known and chosen plaintext attacks. As such, from this cryptanalysis we can recommend against using this cipher in that optional operation mode.

The fourth attack while targeting the same encryption function attempted to use originality to identify characteristics between a plaintext and a corresponding ciphertext by considering a potentially new form of attack on XOR uniqueness properties.

The last attack, using a high order linear cryptanalysis framework, suggested it may be possible to gather some information on the keys used to generate the ciphertext, but no obvious LC related attacks were found.

Further attacks should also be considered as they may apply to this cipher:

- Stream cipher attacks such as related key and correlation attacks.
- Differential Cryptanalysis may also be adapted for this cipher. However, with conventional Differential Cryptanalysis, no differences in the input will always result in no differences in the output; this will not be the case with this Cipher as it uses a random IV for the encryption.

Finally, it is not because a cipher is complex that it means it is more secure, on the contrary, as it is likely prone to implementation errors. That algorithm was never designed with resistance to known cryptanalysis attacks in mind. It was purely designed on an ad-hoc basis where thoughts of extra layer of security were added one after the other. This shows in the complexity of its design and although it was not fully broken during this thesis, signs of potential weaknesses were highlighted.

8. Conclusion

8.1 What was achieved

In this thesis I introduced the basic concepts underlying cryptanalysis and how they were used in modern attacks. While providing a general understanding of those attacks I focused on Linear and Differential cryptanalysis, showing how the discovery of those techniques has paved the way to modern cryptanalysis with their variants dominating the different research currently in progress. More importantly, I provided a comprehensive explanation of Linear Cryptanalysis based on H. Keys tutorial and attempted to make it as approachable as possible for the reader by explaining the basic mathematical concept of the attack.

I then provided some context on how those attacks have been used in the past and are used nowadays by looking at four high profile cipher implementations, two stream ciphers and two block ciphers. I highlighted their weaknesses and ongoing cryptanalysis research breakthrough while referencing the related techniques introduced in this document. For each of those ciphers I also discussed the real world implications of those past successes and extrapolated the consequences of the emerging ones.

Finally, in the last chapter I applied what I learnt through the thesis by starting to conduct the cryptanalysis of the latest version of my own cipher, BUGS, which to my knowledge has never been subject to one before. I started by giving an overview of the cipher, normalised the cipher's description and suggested which of the attacks I previously discussed were applicable or not, and why. I then discussed five attacks which may help highlight some of the BUGS cipher's characteristics. Out of those attacks, two are practical and two are an attempt to create new attack frameworks. This should have provided an insight on how one can start a cipher's cryptanalysis.

8.2 Future of Cryptanalysis

As cryptography continues to underline the data security of existing and new technologies, its impact when broken and the need for assessing that level of security is increasing. As a result, Cryptanalysis is a science which is receiving a lot of attention from a wide sector range: academics, the military, corporations, etc.
It was interesting to compare the information available almost 15 years ago when I created the first version of my algorithm and now, writing this thesis. All that was then available were specific internet forums and chat rooms with information not so easy to reach and limited academic exchange of information. This contrasts with my thesis research where I found hundreds of publicly available papers referencing even more, many conferences on that topic and academics joining forces to facilitate that exchange of information. While still not becoming a widely popular subject it is getting more and more accessible.

Specifically, two areas of Cryptanalysis are increasingly producing some interesting, if not yet practical, new attacks: Algebraic Cryptanalysis and some variants of Differential Cryptanalysis. There is also an emerging tendency to combine the same

classes of cryptanalysis variants together to improve existing attacks. In a few cases, as seen recently with AES, this approach has been taken one step further by combining cryptanalysis techniques designed against different cryptography areas such as hash algorithms and block ciphers.

It is however important to remember that as new cryptanalysis techniques are discovered so are new ways to protect against them.

8.3 Personal learning

This has been a challenging thesis to write as there was much to learn and to understand on what is a complex topic. While doing my research I also felt humbled by the "genius" of those techniques' authors and their required dedication. It also highlighted the place this science has in our history and future. Another aspect of cryptanalysis which is often forgotten is that it requires data to be intercepted and this sometimes has to be done at great human costs as shown on the regularly updated NSA National Cryptologic Memorial website [103].

Working on this thesis has been enlightening, towards the end of my research it felt as if I was almost watching live cryptanalysis of the AES cipher, with what could be the birth of a new wave of attacks. Each time I was researching more information on a type of attack I could see progress being made, new papers published and the attack being refined. Having just studied the concept of those techniques being used helped give me a better appreciation of what was being discussed and discovered.

Finally, this work has given me both professional and personal benefits. Professional, because working in IT Security I can now make better informed decisions related to symmetric-key cryptography. I have for example, already seen on security forums discussions about the new AES published attack, with the focus being on its predicted premature death and the conventional recommendation of using longer keys or to change algorithm altogether. What I would have failed to appreciate before doing this thesis is that understanding the nature of that attack means what seems to be a less secure choice of using AES-128 is in fact a safe recommendation. It also gave me a better understanding of the inner working of some of the tools I have been using professionally, such as Ophcrack, when I found myself reading a paper about TMTO which was written by the author of that tool.

On a personal level, I gained a better understanding of the different cryptanalysis techniques and especially Linear Cryptanalysis. This has given me the basis to conduct a full cryptanalysis of my algorithm. It was extremely interesting and motivating to attempt to break that cipher, especially when weaknesses were found. This has inspired me to carry on in this field, at a personal or academic level, and look at more unconventional solutions such as the possible combination of stream and block cipher techniques which may not have been either tried or fully exploited before. While researching this thesis I indeed noticed a lot of progress was being made in "silos", such as linear, differential, algebraic cryptanalysis, but less was done to take a step back and see how those techniques may overlap.

9. APPENDIX A – Thesis Scripts

9.1 Frequency Analysis script

9.1.1 Information

This script takes a file as input and depending of its parameters will look for the repetition of N hexadecimal values aligned on A bits.
The output will be a list of lines with three fields sorted on the values with the highest repetition count:
Hexa value, repetition count, average distance between repetitions

9.1.2 Usage

Looking for the repetition of hexadecimal values (8 bits long specified with the –l parameter) aligned on 1 byte (8 bits long specified with the –a parameter):
$ ./counter.pl -l 8 -a 8 pl file_to_analyse > file_frequency.csv

Looking for the repetition of 2 hexa values (16 bits) aligned on 1 byte:
~ $ ./counter.pl -l 16 -a 8 file_to_analyse > file_frequency.csv

Looking for the repetition of 1 hexa value aligned on half a byte (-a 4)
Hence, for example, on a binary file the hexa values of “BA DC FE” will be accessed as little-endian values “AB CD EF” and the following repetitions will be reported: “AB”, “BC”, “CD”, “DE” and “EF”. The results will however be displayed in Big-Endian (i.e.: “BA” instead of “AB”)
~ $ ./counter.pl -l 8 -a 4 file_to_analyse > file_frequency.csv

The more interesting option is to look for the repetitions of hexa values aligned on 1 bit:
~ $ ./counter.pl -l 8 -a 1 file_to_analyse > file_frequency.csv

9.1.3 ‘counter.pl’ Script

```
#!/usr/bin/env perl
use strict;
use warnings;
use IO::File;

# Alignment of blocks in the file (in bits)
my $block_align = 8;

# Length of sequences to count (in bits)
my $seq_length = 8;

# Name of the file to parse
my $file;

# Parsing command-line arguments
for (my $i = 0; $i < @ARGV; ++$i) {
   # '-a' option takes an alignment for blocks as parameter
   if ($ARGV[$i] eq '-a') {
      $block_align = $ARGV[++$i];
   }

   # '-l' option takes a length for sequences as parameter
   elsif ($ARGV[$i] eq '-l') {
      $seq_length = $ARGV[++$i];
   }
```

```
 # '-h' option displays some help
   elsif ($ARGV[$i] eq '-h') {
      print "$0 [-a <length>] [-l <length>] [FILE]\n";
      print "Count occurrences and repetition distance between\n";
      print "<l>-bit length sequences, aligned every <a> bits in\n";
      print "FILE or in standard input.\n";
      print " -a <length>  Fix the value <a> (sequence alignment, in bits)\n";
      print " -l <length>  Fix the value <l> (length of sequences in bits)\n";
      print " -h           Display this help and exit\n\n";
      exit 0
   }
   elsif ($ARGV[$i] =~ /-.+/) {
      die "Unknown option: $ARGV[$i]";
   }

# Arguments which are not options are file names
   else {
      if ($file) {
         die "File name given twice."
      }
      $file = $ARGV[$i];
   }
}

# Bits will be the array of 0 and 1 filled with the contents of the file.
my @bits = ();
my $handler;
if (!$file) {
   $handler = \*STDIN;
}
else {
   $handler = new IO::File $file, O_RDONLY
      or die "Cannot open $file";
}
my $char;
while ($handler->read($char, 1) == 1) {
   my $value = ord($char);
   # Given the 8-bit number $value, write it in @bits
   for (my $i = 0; $i < 8; ++$i) {
      if ($value % 2) {
         push @bits, 1
      }
      else {
         push @bits, 0
      }
      $value /= 2;
   }
}

if ($file) {
   $handler->close;
}

my %table_occurrences = ();
my %table_first_occurrence = ();
my %table_last_occurrence = ();

# For each aligned block
for (my $i = 0; $i < @bits - $seq_length + 1; $i += $block_align) {
   # Compose the $seq sequence of $seq_length bits written in hexa.
   my $v = 0;
   my $coef = 1;
   my $seq = "";

  for (my $j = 0; $j < $seq_length; ++$j) {
      $v += $bits[$i + $j] * $coef;
      if ($coef == 0x80) {
         $seq .= sprintf("%.2X", $v);
         $v = 0;
         $coef = 1;
```

```
        }
        else {
            $coef *= 2;
        }
    }
    if ($coef > 1) {
        $seq .= sprintf("%.2X", $v);
    }
    # Count it
    ++$table_occurrences{$seq};
    if ($table_occurrences{$seq} == 1) {
        $table_first_occurrence{$seq} = $i;
    }
    $table_last_occurrence{$seq} = $i;
}


# Print the table sorted decreasingly on values.
foreach my $key
    (sort {$table_occurrences{$b} <=> $table_occurrences{$a}}
     keys %table_occurrences) {
    my $repetitions;
    if ($table_occurrences{$key} >= 2) {
        $repetitions =
            ($table_last_occurrence{$key}
             - $table_first_occurrence{$key} - $table_occurrences{$key} + 1) /
                ($table_occurrences{$key} - 1);
    }
    else {
        $repetitions = "0";
    }
    print "$key, $table_occurrences{$key}, $repetitions\n";
}
```

9.2 Graphical bits representation script

9.2.1 Information

This script takes a file as input and converts its bitstream into a XPM file format which it sends through the standard output stream. This format allows the definition of an image into a character table which can then be used in tools such as Imagemagick [104] to generate a picture from that table.

9.2.2 Usage

For a file in ASCII format:
~$ ./image.pl file_to_convert > file_converted.xpm

For a file in binary format, we use a '-r' parameter:
~$ ./image.pl –r file_to_convert_binary > file_converted.xpm

By default the script will generate a representation for a 256 pixels wide image, it is possible to specify a different pixel wide size, i.e.: 512, by using the '-w' parameter:
~$ ./image.pl –w 512 file_to_convert > file_converted.xpm

Once the .xpm file has been generated we can use Imagemagick to generate the picture:
~$ convert file_converted.xpm file_picture.png

Another interesting command is to use Imagemagick to generate a differencial picture between two pictures:
~$ convert plaintext.xpm ciphertext.xpm –compose difference –composite difference.png

9.2.3 'Image.pl' Script

```
#!/usr/bin/perl
use strict;
use warnings;
use IO::File;
use integer;

my $column_count = 256;
my $file;
my $raw = 0;
# Parsing command-line arguments
for (my $i = 0; $i < @ARGV; ++$i) {
   # '-w' option takes a column count as parameter
   if ($ARGV[$i] eq '-w') {
      $column_count = $ARGV[++$i];
   }
   # '-r' option for raw files
   elsif ($ARGV[$i] eq '-r') {
      $raw = 1;
   }
   elsif ($ARGV[$i] =~ /-.+/) {
      die "Unknown option: $ARGV[$i]";
   }
   # Arguments which are not options are file names
   else {
      if ($file) {
         die "Twice file names"
      }
      $file = $ARGV[$i];
   }
}
```

```
if (!$file) {
   die "Need a file name as argument.";
}
# Bits will be an array of 0 and 1
my @bits = ();
sub extract($) {
   my ($value) = @_;
   # Given the 8-bit number $value, write it in @bits
   for (my $j = 0; $j < 8; ++$j) {
      if ($value % 2) {
         push @bits, 1
      }
      else {
         push @bits, 0
      }
      $value /= 2;
   }
}

my $handler = new IO::File $file, O_RDONLY
   or die "Cannot open $file";
if ($raw) {
   # Raw files are read char by char
   my $char;
   while ($handler->read($char, 1) == 1) {
      extract(ord($char))
   }
}
else {
   # Raw files are read line by line
   while (my $line = <$handler>) {
      chomp $line;
      $line =~ s/\[BUGS_ASCII_MODE_v04_START\][[:digit:]]*,//g;
      $line =~ s/\[BUGS_ASCII_MODE_v04_END\]//g;
      for (my $i = 0; $i < length $line; $i += 2) {
         # Each character is written by two hexagits
         extract(hex(substr($line, $i, 2)));
      }
   }
}

$handler->close;
# Compute line_count in function of column_count and the number of
# read bits.  It is ceil(bit count / column count), that is to say
# floor((bit count + column count - 1) / column count).

my $line_count = (@bits + $column_count - 1) / $column_count;
# XPM header
print "static char *file_xpm[] = {\n";
print "\"$column_count $line_count 2 1\",\n";
print "\"       c #FFFFFF\",\n";
print "\".      c #000000\"";
my $index = 0;
for (my $i = 0; $i < $line_count; ++$i) {
   print ",\n\"";
   for (my $j = 0; $j < $column_count; ++$j) {
      if ($bits[$index++]) {
         print ".";
      }
      else {
         print " ";
      }
   }
   print "\"";
}
print "};\n"
```

10. APPENDIX B – BUGS Cryptanalysis Results

Below are the results of the ciphertext Frequency analysis, some data highlighted in grey were taken out for this table to only fit on two pages. The values highlighted in pink and red are of interest.

Value	1 bit alignment	Distance		Value	1 byte alignment	Distance
FF	1332	141.85		FF	194	976.78
7F	1157	163.47		DF	159	1200.37
FE	1157	163.47		EF	156	1207.52
DF	1120	168.75		FE	150	1269.17
EF	1113	169.68		FB	148	1279.27
F7	1101	171.71		BF	146	1293.73
FD	1099	171.89		F7	146	1302.89
BF	1091	173.38		F4	145	1296.56
FB	1088	173.78		CF	143	1336.80
F3	973	194.61		DE	142	1301.35
E7	972	194.64		ED	142	1342.15
DD	969	194.73		BA	141	1335.91
EB	964	196.27		8D	137	1396.18
3F	962	196.80		EB	134	1391.42
B7	959	197.10		AE	133	1419.55
FC	954	198.44		98	132	1429.53
F9	948	199.74		B0	132	1438.94
CF	944	200.48		F8	132	1426.36
DB	943	200.25		93	131	1436.42
9F	943	200.70		BB	130	1454.07
6F	942	200.68		9E	130	1465.42
DE	935	202.20		F2	129	1473.69
FA	932	203.08		EC	129	1454.81
BD	932	202.47		BE	129	1455.63
F5	921	205.36		9F	128	1467.28
7D	921	205.50		9A	128	1407.57
BE	920	205.97		EE	127	1493.54
BB	918	205.62		9B	127	1486.75
ED	912	207.10		B7	125	1519.71
7E	896	211.06		B2	125	1514.81
5F	891	212.75		E6	124	1526.28
77	886	213.09		E8	123	1539.92
7B	883	213.95		9C	123	1539.79
EE	874	216.69		E7	123	1549.62
AF	874	216.84		BC	122	1568.52
D7	872	217.10		AC	122	1567.00
F6	870	217.59		B6	121	1556.93
1F	861	220.05		F5	121	1566.00
BA	856	221.16		E3	121	1582.33
75	848	223.12		DA	121	1538.20

F8	847	223.58		AD	120	1561.96
C7	844	224.54		FD	119	1573.58
3E	842	225.20		F3	119	1601.78
7C	841	225.47		8F	119	1592.97
8F	841	225.31		FC	118	1574.04
6E	840	225.05		9D	118	1611.03
CE	834	226.51		DB	118	1554.01
F4	829	227.83		DD	117	1602.93
37	823	230.16		AB	117	1627.97
D5	819	230.69		E5	117	1545.76
AB	818	231.43		8B	117	1600.24
7A	818	230.82		F0	117	1609.07
E6	818	231.72		CE	117	1574.93
C0	646	293.71		4E	68	2795.30
1	646	293.71		41	68	2715.18
A9	646	293.39		58	67	2649.42
44	636	297.88		24	63	2933.58
82	636	295.00		59	63	2969.84
11	634	298.04		55	62	3081.89
14	634	299.33		6C	62	3057.49
42	633	299.11		4A	62	2969.36
58	632	300.41		3C	62	3048.84
0D	632	299.09		60	62	3011.85
21	630	300.66		48	62	2985.36
1A	629	301.18		15	61	2964.87
22	628	302.19		43	60	3170.66
10	627	299.90		51	60	3114.93
2B	627	302.76		72	60	3141.37
…	…	…	…	…	…	…
…	…	…	…	…	…	…
25	615	308.76		7E	56	3361.76
12	612	306.07		73	56	3395.65
6	608	311.30		46	56	3393.76
32	605	313.92		49	55	3373.67
94	602	314.73		25	55	3351.59
2	599	316.87		65	55	3094.85
4	598	316.79		66	55	3454.41
81	597	316.85		26	54	3533.34
40	597	317.06		64	51	3713.88
9	594	318.67		54	51	3719.64
29	593	320.13		44	50	3817.12
90	590	321.39		68	49	3914.00
20	587	322.46		35	48	3825.38
52	583	325.00		61	48	3898.91
4A	580	327.64		6A	45	4242.27

11. APPENDIX C – BUGS Cipher Detailed Diagrams

The diagrams below are provided in order of the cipher's workflow where anything in red is IV/Key dependant.

11.1 Key Scheduling

11.1.1 Initialisation

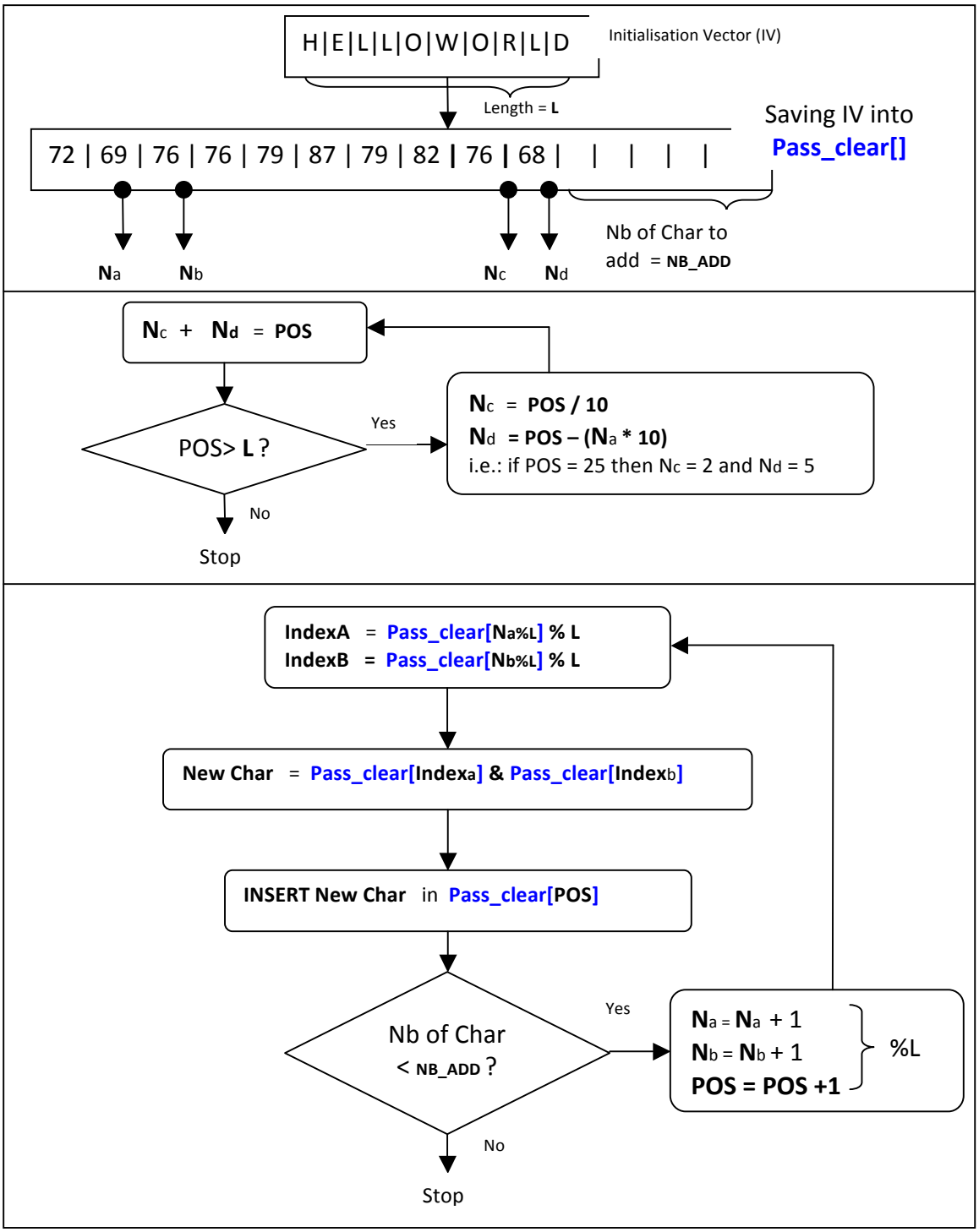

Figure 20. Key Padding function – test_length()

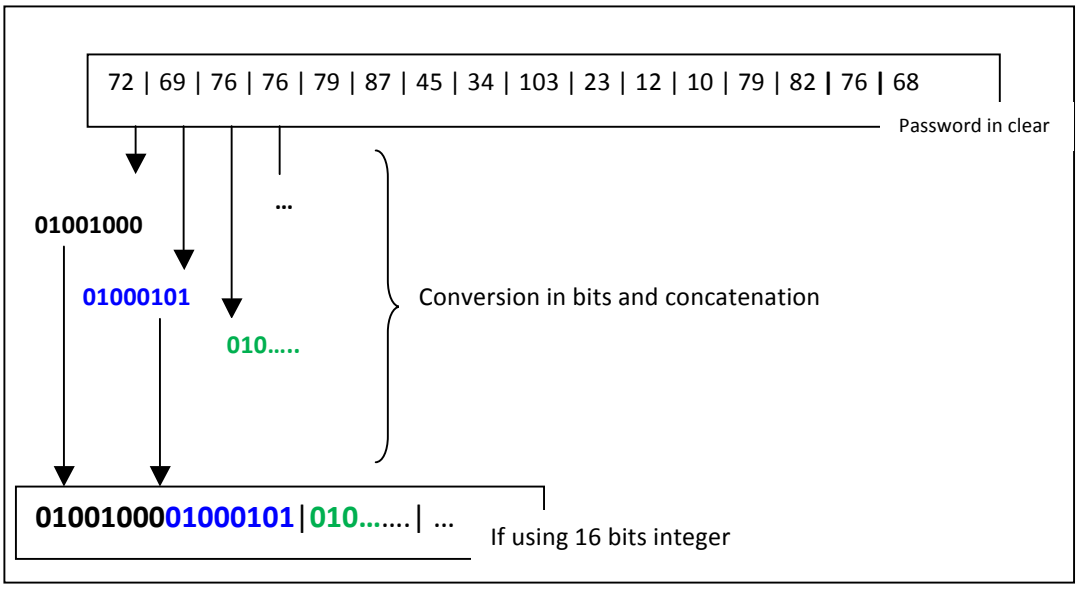

Figure 21. Bits concatenation function – Transcription()

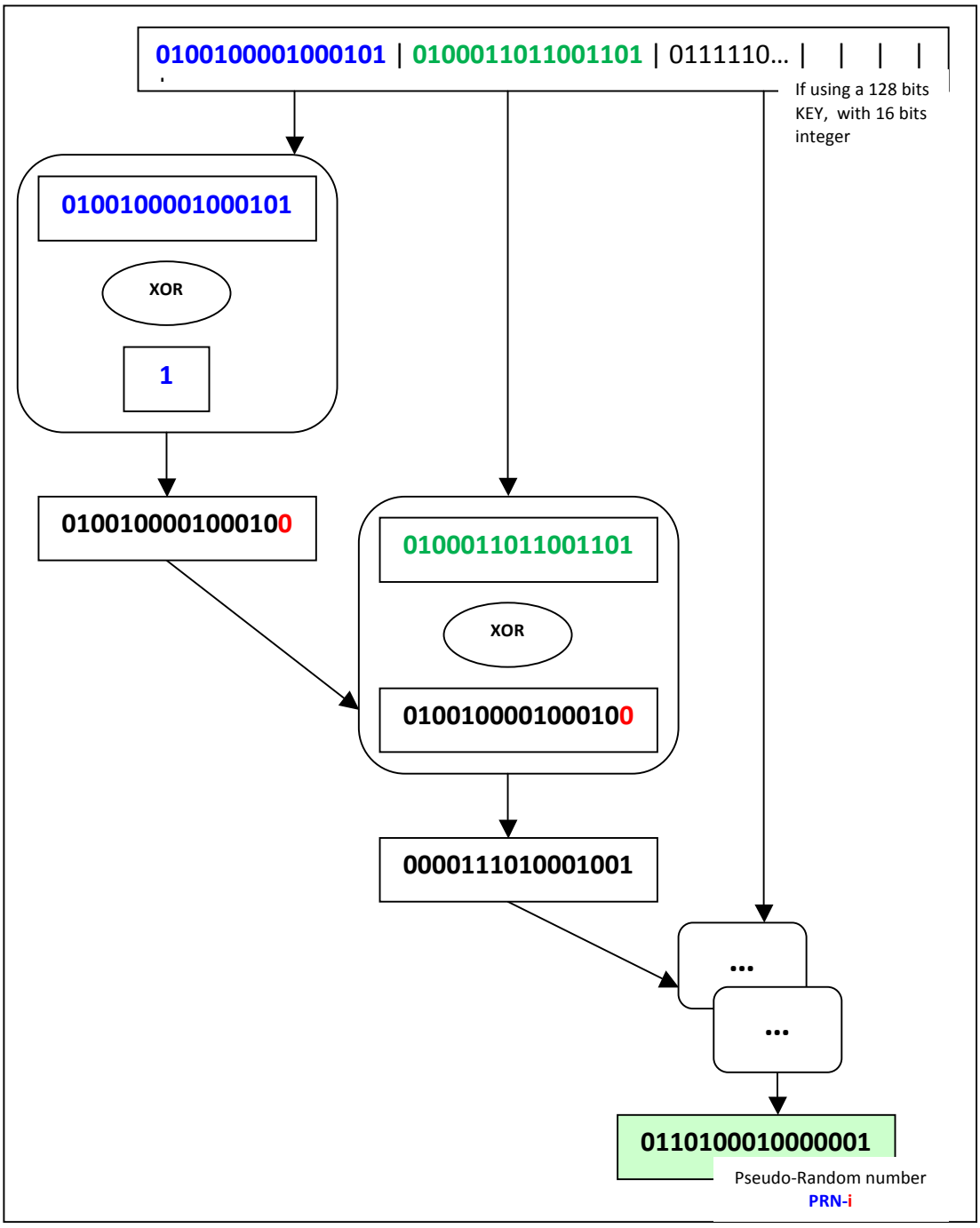

Figure 22. Initial Scrambling – Add() Random Generation

11.1.2 Initial Scrambling

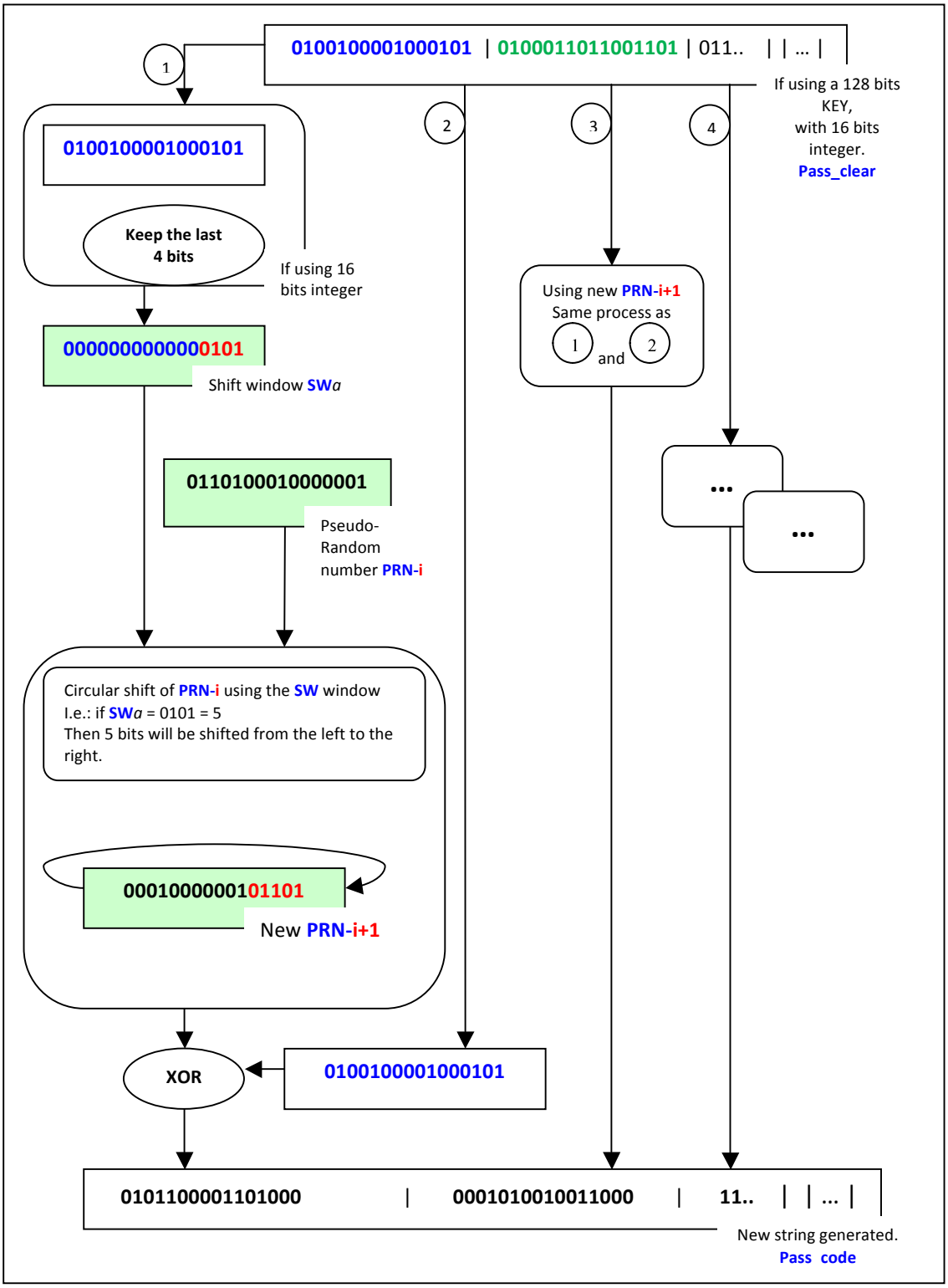

Figure 23. Initial Scrambling - Add() Pass_code generation

11.1.3 Key Encryption – Swapping Part

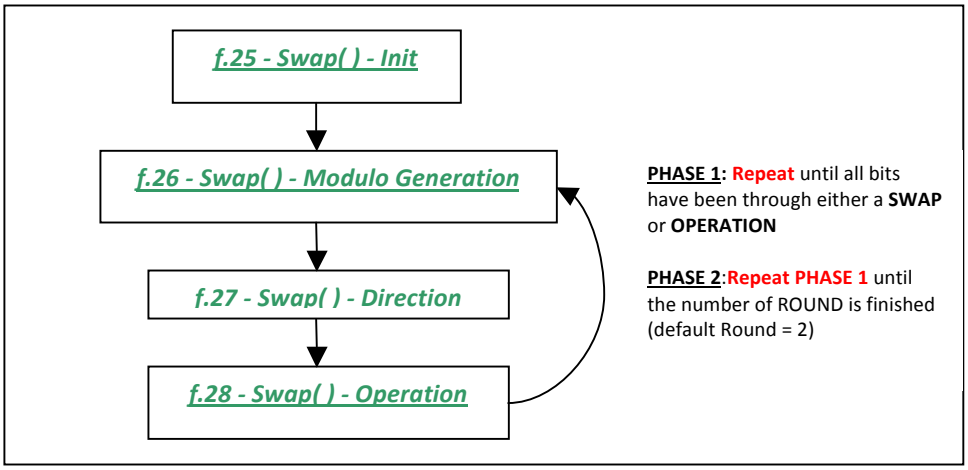

Figure 24. Key Encryption function – Swap() Overview

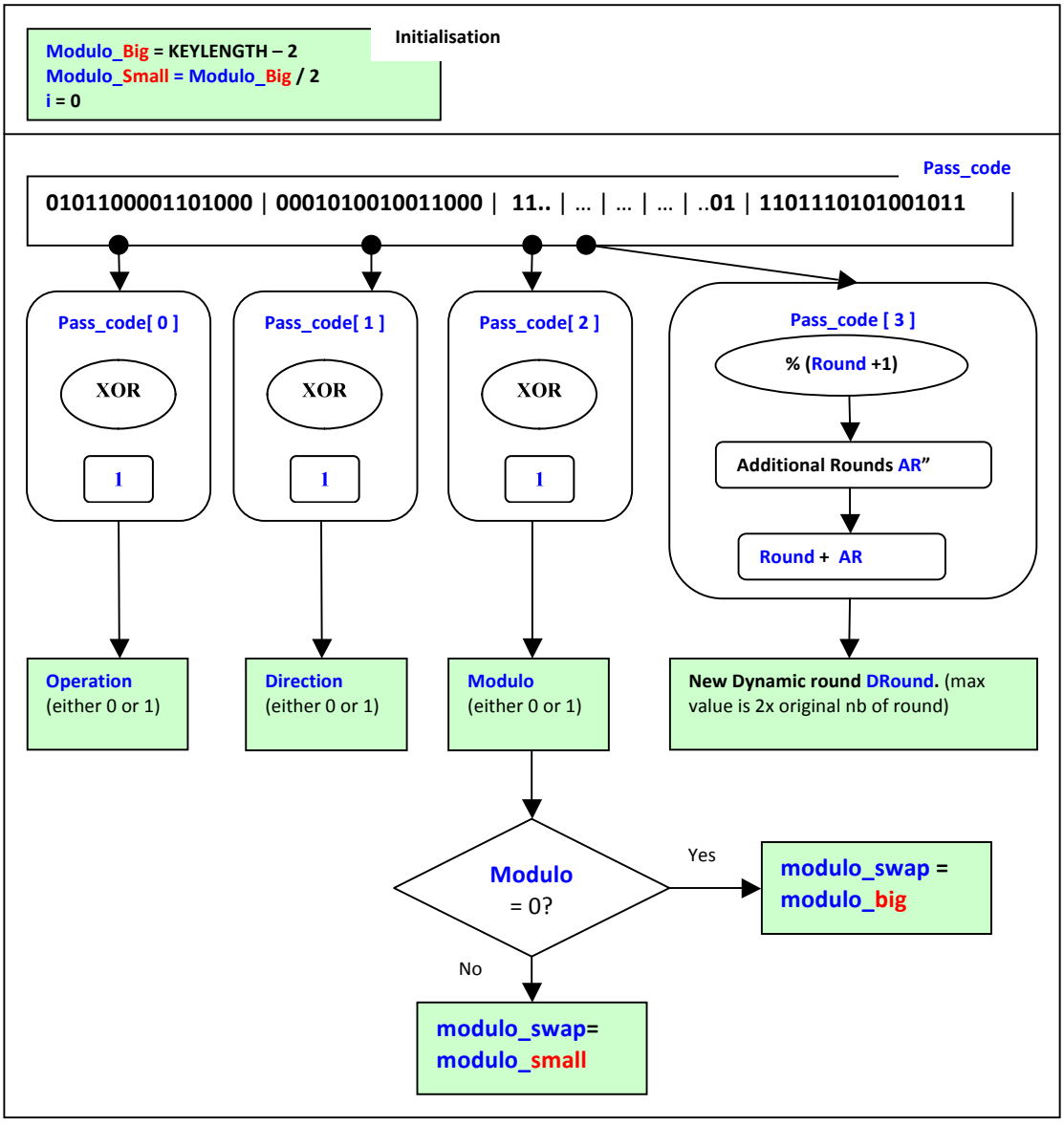

Figure 25. Key Encryption function – Swap() Initialisation

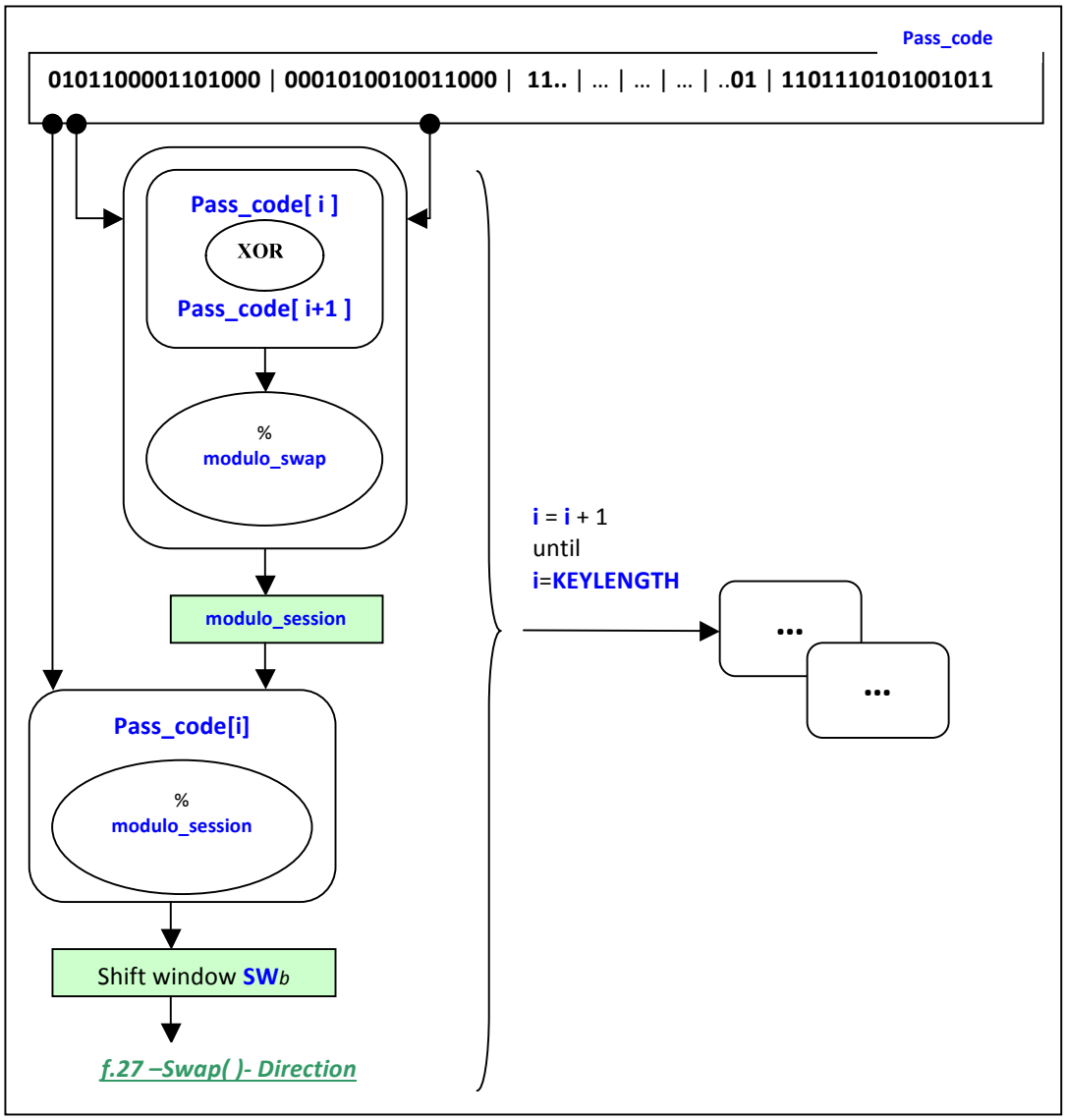

Figure 26. Key Encryption function – Swap() Modulo Generation

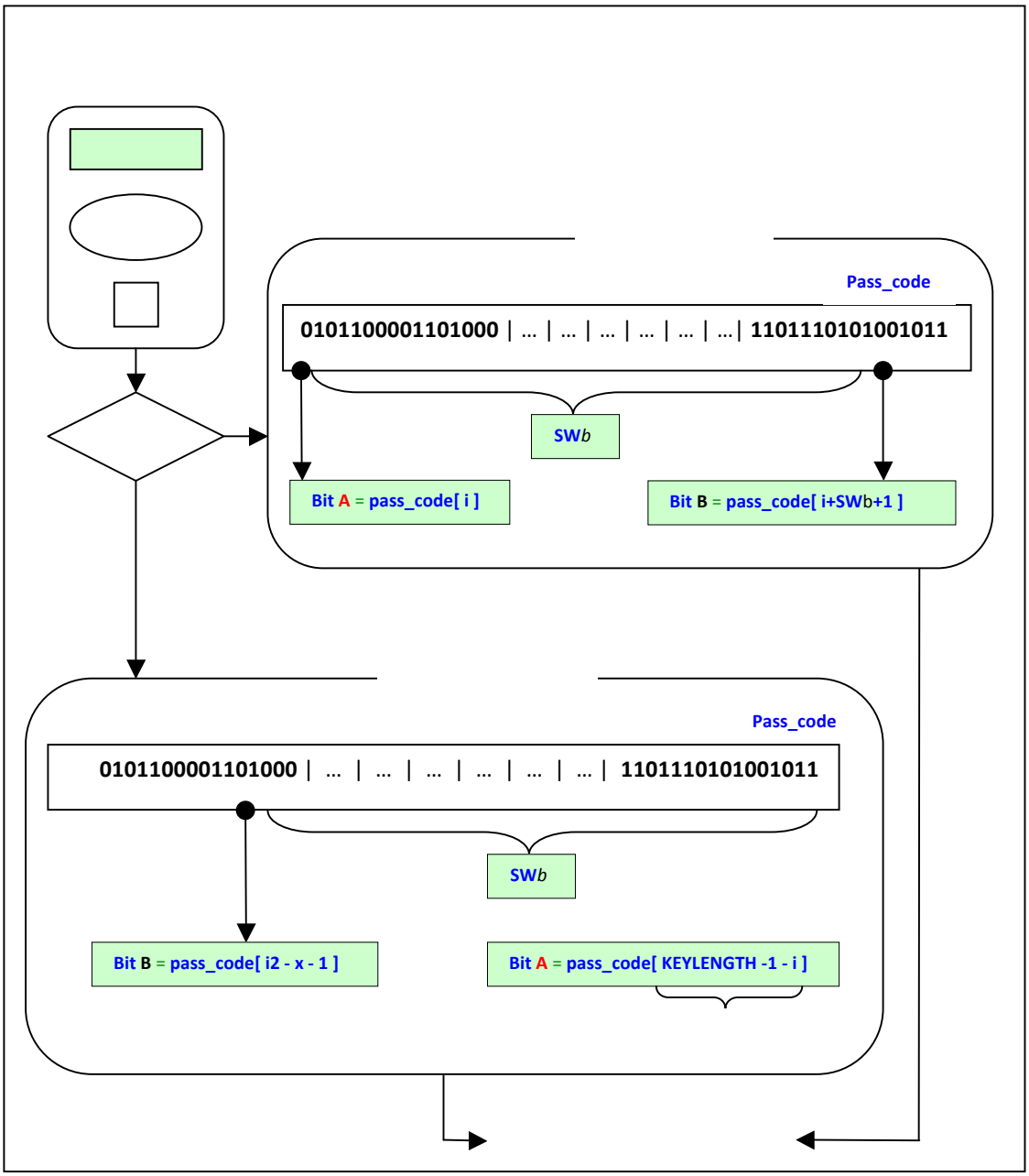

Figure 27. Key Encryption function – Swap() Direction

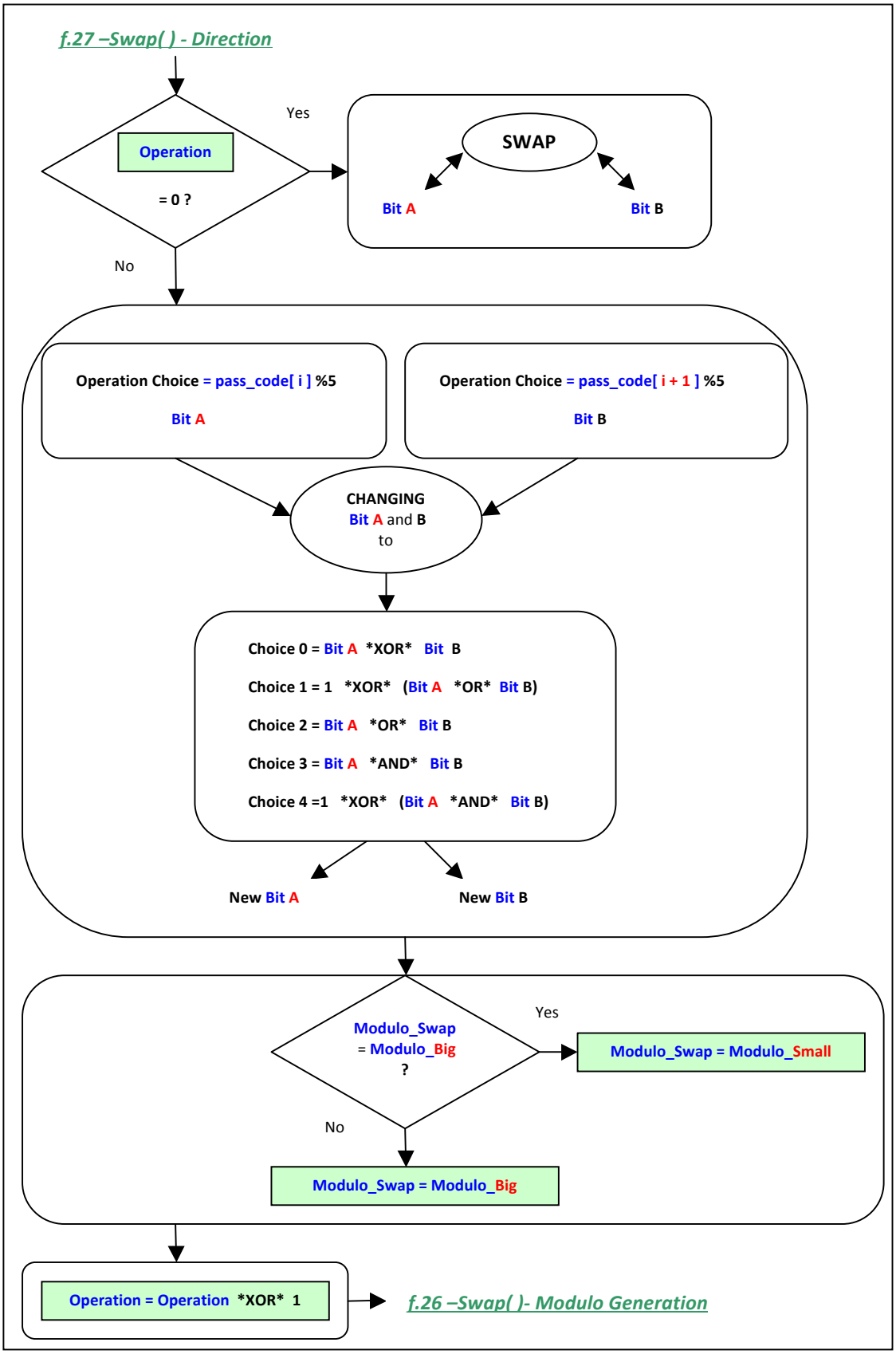

Figure 28. Key Encryption function – Swap() Operation

11.1.4 Key Encryption – Coding part

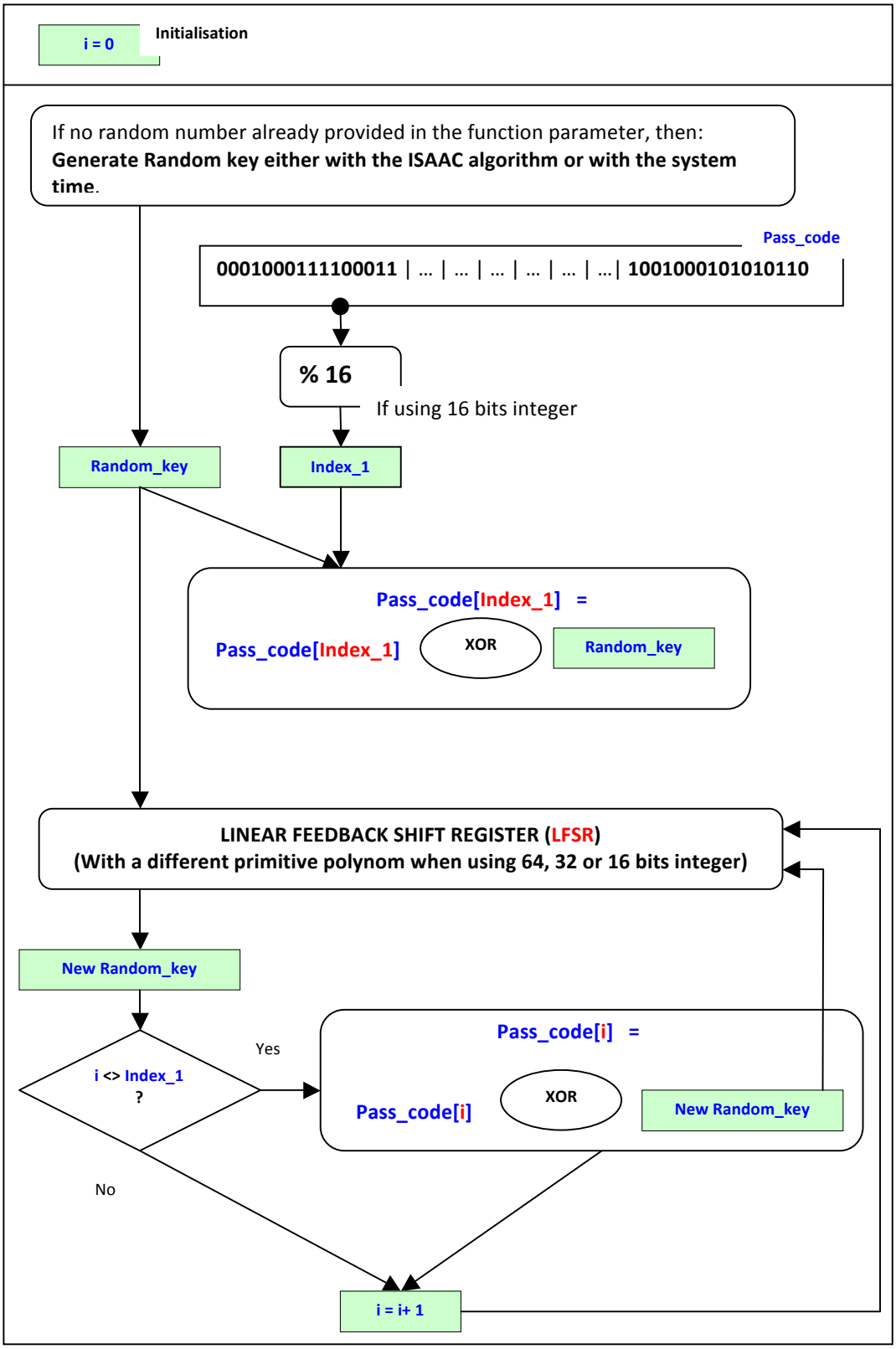

Figure 29. Key Encryption function – Code()

11.2 File/Plaintext Encryption Function

11.2.1 Initialisation

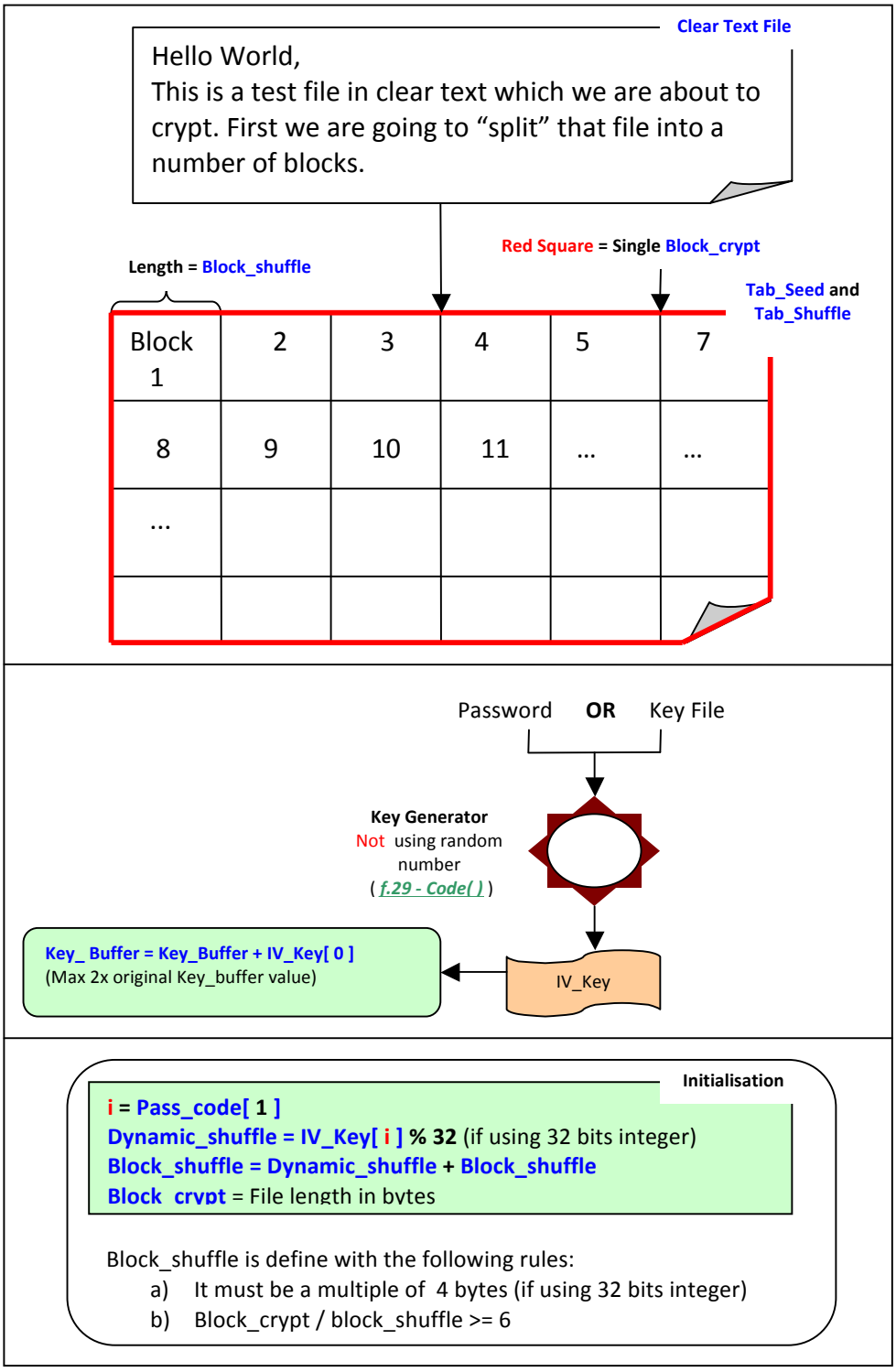

Figure 30. File encryption function – File_crypt() Initialisation

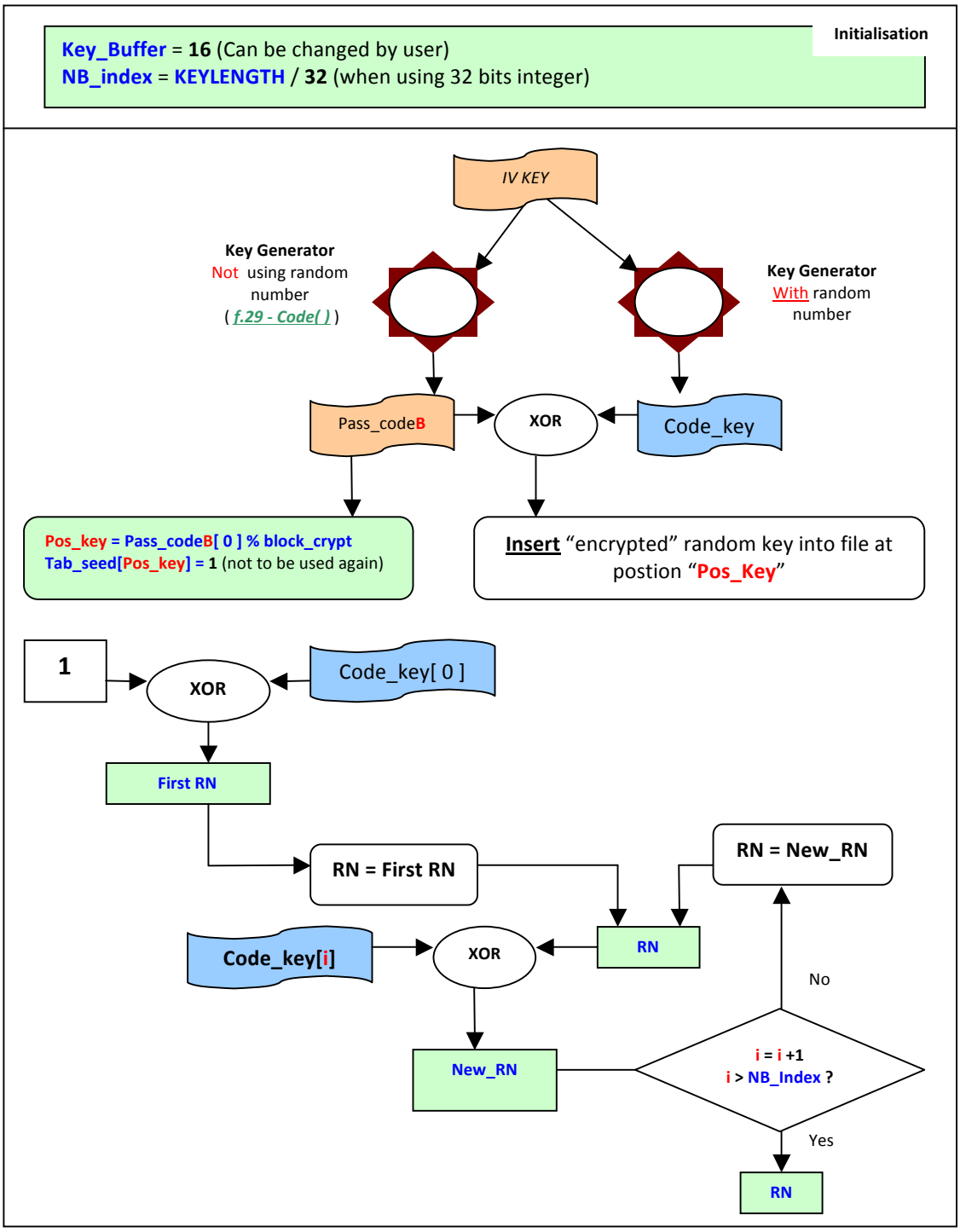

Figure 31. File encryption function – Seed() Random Generation

11.2.2 Seeding

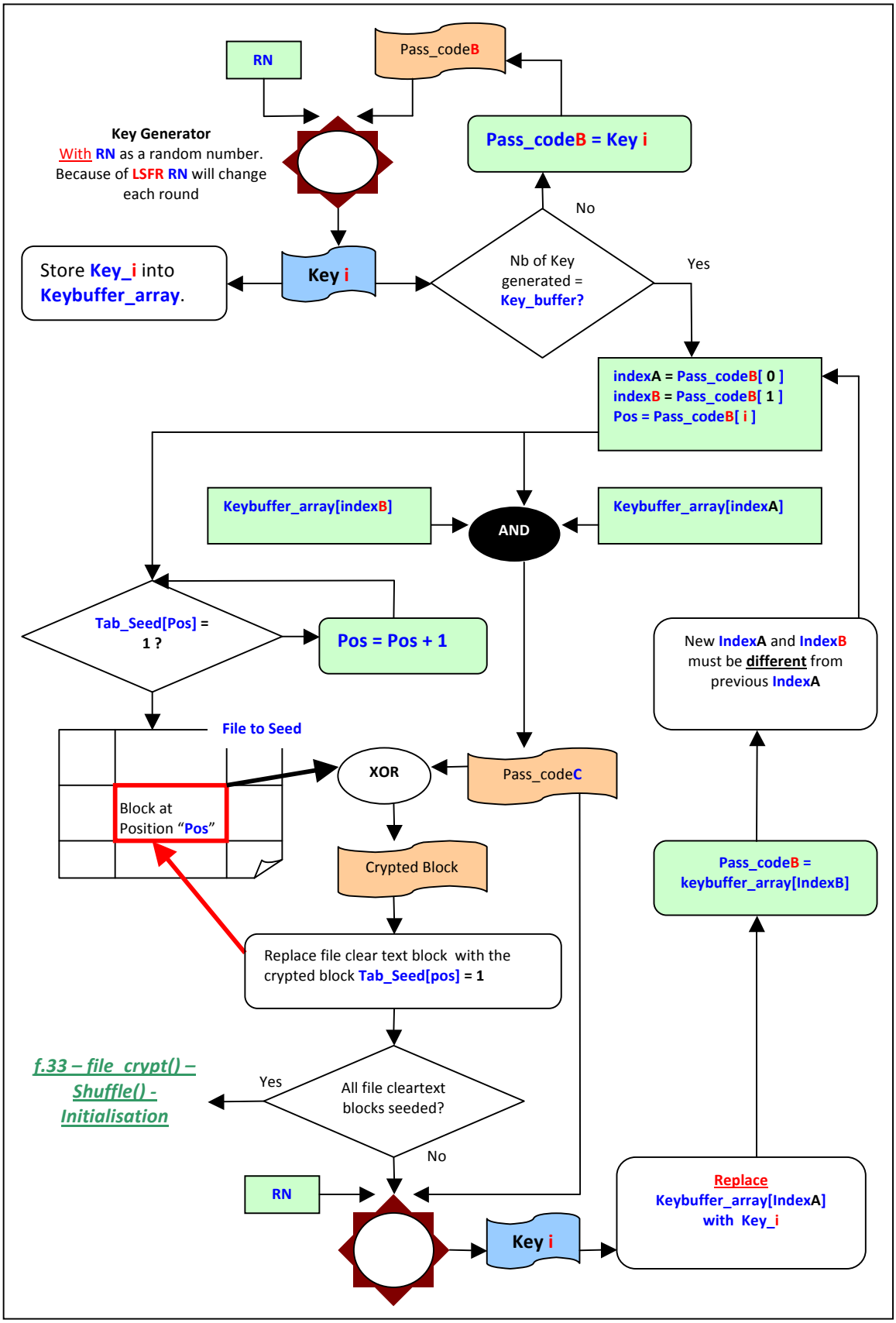

Figure 32. File encryption function – Seed() Probability Seed

11.2.3 Shuffling

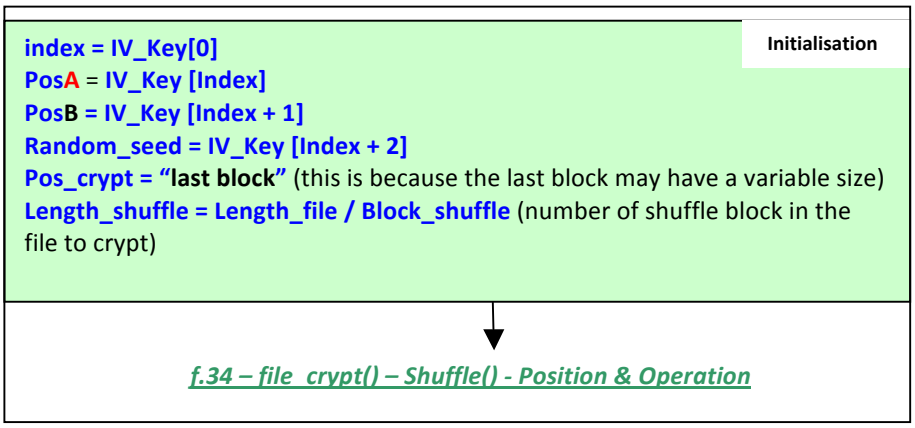

Figure 33. File encryption function – file_crypt() and Shuffle() Initialisation

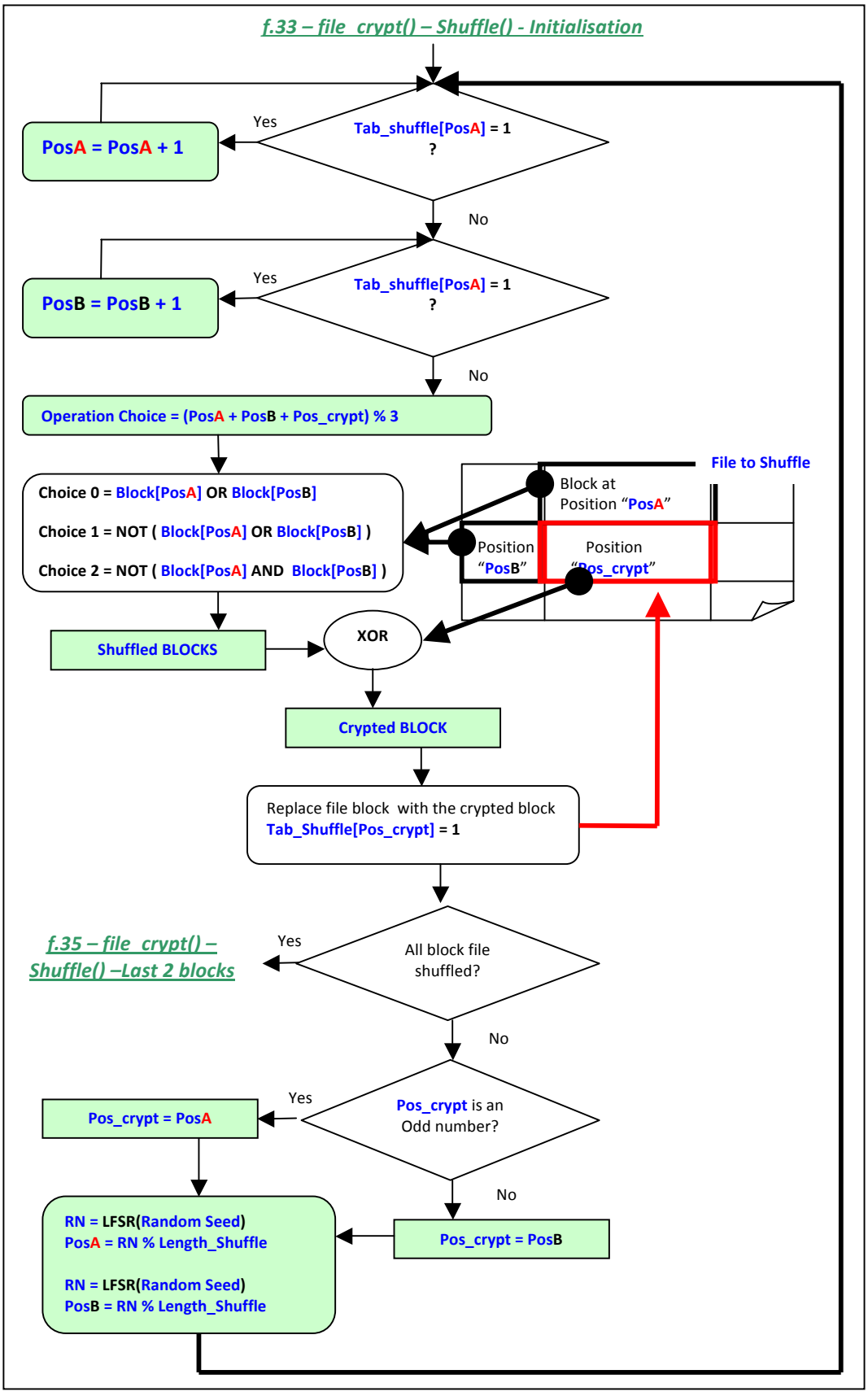

Figure 34. File encryption function –file_crypt() and Shuffle() Position & Operation

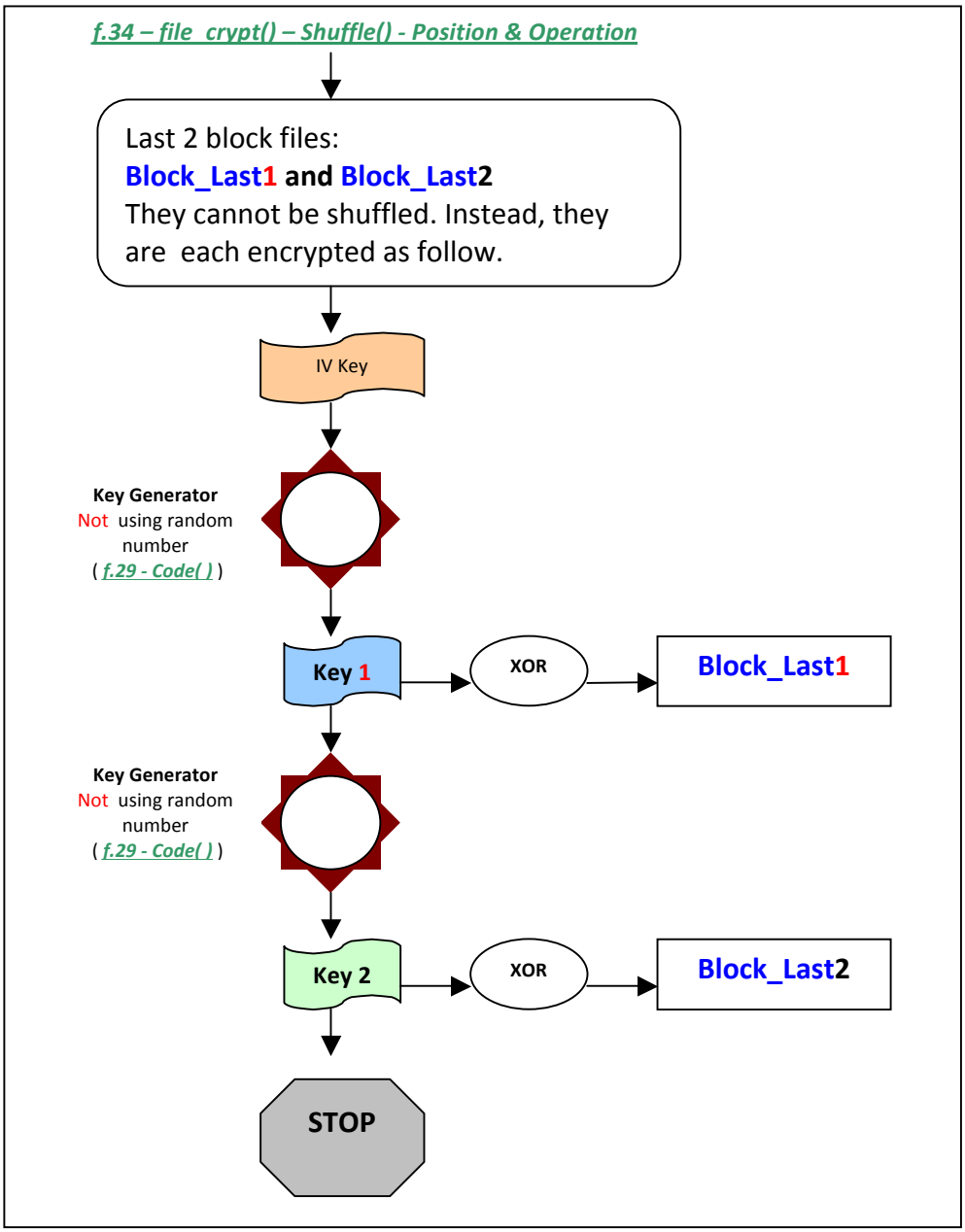

Figure 35. File encryption function – file_crypt() and Shuffle() Last 2 Blocks

11.2.4 Alternatives

As stated at the beginning, all the variables highlighted in red in the previous steps are IV/Key Dependant, they can also be changed to a different static values as an option. The same is true for the size of the "block_crypt" as shown in Figure 36 below. This means that all the above steps can either be conducted across the entire file or within smaller "working blocks".

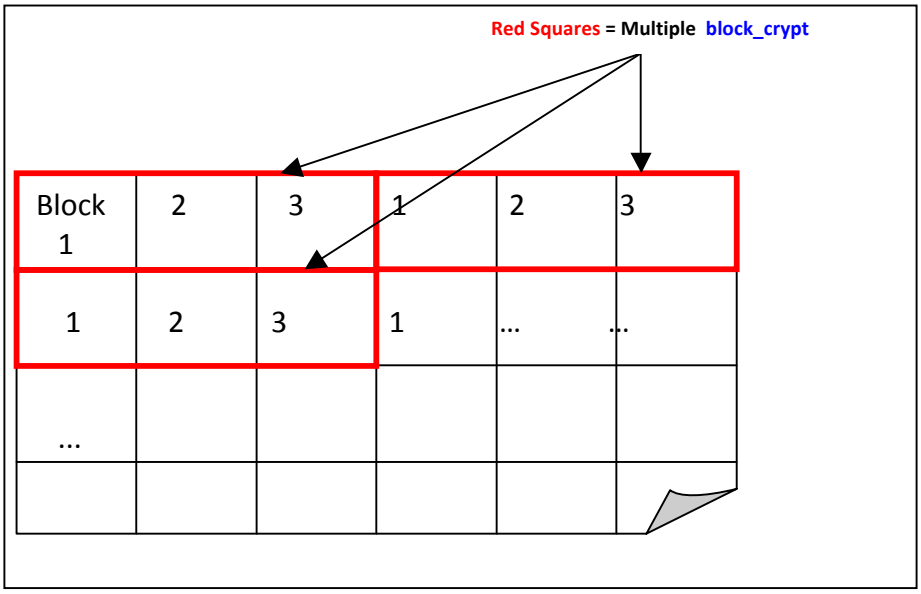

Figure 36. File encryption function – Alternative Block Crypt Size

12. Bibliography

[1] *Cryptography History* - S. Singh; The Code Book; Fourth Estate; 1999

[2] *Cipher definition* - Linear Cryptanalysis of Substitution-Permutation Networks; L. Keliher; Queen's University, Kingston, Ontario, Canada; October 2003; Page 9

[3] *Types of Cryptography* – A. Menezes, P. van Oorschot, S. Vanstone; Handbook of Applied Cryptography; CRC Press; 1997; Page 15-32.

[4] *Block Cipher modes of operation* – A. Menezes, P. van Oorschot, S. Vanstone; Handbook of Applied Cryptography; CRC Press; 1997; Page 228-233.

[5] *Horst Feistel* - http://en.wikipedia.org/wiki/Horst_Feistel

[6] *Feistel Description* - M. Stamp and R. Low; Applied Cryptanalysis: Breaking Ciphers in the Real World; Wiley; 2007; Pages 131-132

[7] *DES* - D. Stinson; Cryptography: Theory and Practice; Third Edition; Chapman & Hall; 2006; Pages 95- 102.
Also described in FIPS 46-3

[8] *Confusion/Diffusion* – C. Shannon; Communication theory of secrecy systems, Bell System Technical Journal Vol 28; 1949; Page 656.

[9] *Quantum Cryptography flawed sense of security* – Bruce Schneier; Crypto-gram newsletter; Quantum Cryptography; 15 November 2008
http://www.schneier.com/crypto-gram-0811.html#4

[10] *The Great Cipher* - S. Singh; The Code Book; Fourth Estate; 1999; Pages 52-58

[11] *The Playfair Cipher* - S. Singh; The Code Book; Fourth Estate; 1999; Appendix E.
A good explanation is also currently available on Wikipedia, http://tinyurl.com/388wt8

[12] *The Vigenère Cipher* - An interactive explanation of the cipher is available on Simon Singh's website, http://tinyurl.com/q5opzs
An interactive explanation on how to break that cipher is available on the website of the Information Security Laboratory organization at the Oregon State University: http://tinyurl.com/qcomse

[13] *Index of Coincidence* – C. Swenson; Modern Cryptanalysis: Techniques For Advanced Code Breaking; Willey Publishing; 2008; Pages 12-15.

[14] *Feistel Cipher* – A. Menezes, P. van Oorschot, S. Vanstone; Handbook of Applied Cryptography; CRC Press; 1997; Page 251.

[15] Cryptanalysis of Enigma – Wikipedia: http://en.wikipedia.org/wiki/Cryptanalysis_of_the_Enigma

[16] *Forced Plain Text in WWII* - S. Singh; The Code Book; Fourth Estate; 1999; Page 183

[17] *AES* - NIST; FIPS 197; 2001
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[18] *Related Keys* - E. Biham; New type of cryptanalytic attacks using related keys; Eurocrypt '93; Pages 398-409

[19] *Slide Attack History* - E. Grossman and B. Tuckerman; Analysis of a Feistel-like cipher weakened by having no rotating key; IBM; Thomas J. Watson Research; Technical Report RC 6375.
[20] *Slide Attack Details* – A. Biryukov and D. Wagner; Slide Attacks; FSE'99; Pages 245–259.
[21] *Birthday Paradox* - http://en.wikipedia.org/wiki/Birthday_paradox
[22] *Slide Attacks Extensions* – A. Biryukov and D. Wagner – Advanced Slide Attacks; EUROCRYPT 2000; Pages 589-606
[23] *Meet in the Middle Attack* - W. Diffie and M. E. Hellman, Exhaustive Cryptanalysis of the NBS Data Encryption Standard, 1977
[24] *Hellman Time-Space Tradeoff Attack* - M. Hellman; A Cryptanalytic Time Memory Trade-off; IEEE no 26(4); 1980
[25] *Hellman TMTO explanation* – M. Stamp and R. Low; Applied Cryptanalysis: Breaking Ciphers in the Real World; Wiley; 2007; Pages 133-142
[26] *Distinguished Points* – D. Denning; Cryptography and Data Security; Addison-Wesley; 1982
[27] *Rainbow Tables* - P. Oechslin ; Making a Faster Cryptanalytic Time-Memory Trade-Off; CRYPTO'03
[28] *Distinguished Rainbow Points* - Tim Guneysu, Andy Rupp and Stefan Spitz; Cryptanalytic Time-Memory Tradeoffs on COPACOBANA; 2007
[29] *COBACOBANA* - http://www.copacobana.org/
[30] *Start of Linear Cryptanalysis* – M. Matsui and A. Yamagishi; A new method for known plaintext attack of FEAL Cipher; Eurocrypt '93; Pages 81-91
[31] *FEAL Cipher* – A. Shimizu and S. Miyaguchi; Fast Data Encipherment Algorithm FEAL; Eurocrypt '87; Pages 267-278
[32] *Linear Cryptanalysis Tutorial* – H. Heys; A Tutorial on Linear and Differential Cryptanalysis; Memorial University of Newfoundland
[33] *Matsui Linear search algorithm* - M. Matsui; The first experimental crypt analysis of the Data Encryption Standard; Crypto'94, Pages 1-11.
[34] *Another Linear search algorithm* – K. Ohta, S. Moriai, K. Aoki; Improving the search algorithm for the best linear expression; Crypto'95; Pages 157-170.
[35] *Stream Ciphers Linear Cryptanalysis* – J. Golic: Linear Cryptanalysis of Stream Ciphers; FSE 1994; Pages 154-169
[36] *Bluetooth Stream Cipher* - J. Golic, V. Bagini, G. Morgari; Linear Cryptanalysis of Bluetooth Stream Cipher; EUROCRYPT 2002; Pages 238-255
[37] *Complexity attack ratio for DES* – M. Matsui; Linear Cryptanalysis method for DES Cipher; Eurocrypt'93; Page 393, Formula no. 15.
[38] *Complexity attack ratio for FEAL-8* – E. Biham; On Matsui's Linear Cryptanalysis; 1994; Page 353
[39] *Linear Hull Definition* - K. Nyberg; Linear approximation of block ciphers; EUROCRYPT'94; Pages 439-444
[40] *Linear Hull modern application* - Linear Cryptanalysis of Substitution-Permutation Networks; L. Keliher; Queen's University, Kingston, Ontario, Canada; October 2003.

[41] *Multiple Linear Approximations* – B. Kaliski Jr, M. Robshaw; Linear Cryptanalysis using multiple approximations; Crypto ’94; Pages 26-39.
[42] *Multiple Linear Approximations Framework* - A. Biryukov, C. De Canniere, M. Quisquater; On Multiple Linear Approximations; Lecture Notes in Computer Science 3152, proceedings of CRYPTO'2004, Pages 1-22.
[43] *Linear Cryptanalysis Variant* - By H. Tilborg ; Encyclopedia of Cryptography and Security; Springer; 2005.
[44] *Key-Ranking* - P. Junod, S. Vaudenay; Optimal key ranking procedures in a statistical cryptanalysis; Swiss Federal Institute of Technology; 2003.
[45] *Partitioning Cryptanalysis* – C. Harpes, G. Kramer, J. Massey; A Generalization of Linear Cryptanalysis and the Applicability of Matsui's Piling-up Lemma; Eurocrypt’95; May 1995; Pages 24-38.
[46] *Chi-Square Attack* – J. Kelsey , B. Schneier , D. Wagner, C. Hall; Side Channel Cryptanalysis of Product Ciphers; Counterpane Internet Security
[47] *Chi-Square Introduction*- http://tinyurl.com/kk7yhc
[48] *Non-Linear Cryptanalysis* - L. Knudsen and M. Robshaw, Non-linear approximations in linear cryptanalysis; Eurocrypt’96; Pages 224–236.
[49] *Differential Cryptanalysis* – E. Biham and A. Shamir; Differential Cryptanalysis of DES like cryptosystems; Crypto’90; Pages 2-21.
[50] *IBM and NSA on differential Cryptanalysis* – D. Coppersmith; The Data Encryption Standard (DES) and its strength against attacks; IBM Journal of Research and Development 38
[51] *American Cryptology during the cold* war - Thomas R. Johnson, http://tinyurl.com/55926o
[52] *Modern Cryptanalysis* – C. Swenson; Modern Cryptanalysis, Techniques for Advanced Code Breaking; Wiley; 2008.
[53] *Differential-Linear combined attack*– M. Hellman and S. Langford; Differential-linear Cryptanalysis, Crypto’94; Pages 26-39
[54] *Conditional Characteristics* – I. Ben-Aroya and E. Biham; Differential Cryptanalysis of Lucifer; Crypto’93; Pages 187-199.
[55] *Luficer Algorithm* – A. Sorkin; Lucifer, a Cryptographic Algorithm; Cryptologia volume 8; 1984; Pages 22-41
[56] *RDES* – K. Koyama and R. Terada; How to Strenghtn DES-like Cryptosystems against Differential Cryptanalysis; IEICE Volume E76-A; 1993; Pages 63-69
[57] *Differential variants*- L. Knudsen; Truncated and Higher Order Differentials; Fast Software Encryption: Second International Workshop; 1994; Pages 196-211.
[58] *Twofish* – B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall and N. Ferguson; The Twofish Encryption Algorithm: A 128-bit Block Cipher; John Wiley & Sons; 1999
[59] *Twofish Cryptanalysis* – S. Moriai and Y. Yin; Cryptanalysis of Twofish (II); 2000
[60] *Impossible Differential Introduction* – E. Biham, A. Biryukov and A. Shamir; Rump session presentation at Crypto '98; http://tinyurl.com/kpp4sn

[61] *Impossible Differential Technique* – E. Biham, A. Biryukov and A. Shamir; Miss in the Middle Attacks on IDEA, Khufu and Khafre; FSE; 1999*;* Pages 124–138.
[62] *Boomerang Attack* – D. Wagner; The Boomerang Attack; FSE'99; Pages 156-170.
[63] *Amplified Boomerang Attack* – J. Kelsey, T. Kohno and B. Schneier; Amplified Boomerang attacks against reduced-round MARD and Serpent; 2000.
[64] *Rectangle Attack* – E. Biham, O. Dunkelman and N. Keller; The Rectangle Attack – Rectangling the Serpent; 2001
[65] *Integral Cryptanalysis* - L Knudsen and D. Wagner; *Integral Cryptanalysis; FSE;* Lecture Notes in Computer Science Vol 2365; 2002; Pages 112-127
[66] *Square* – J. Daemen, L. Knudsen, V. Rijmen; The Block Cipher Square; FSE; Lecture Notes in Computer Science Volume 1267; 1997; Pages 149–165
[67] *Saturation Attack* – S. Lucks; The Saturation Attack - a Bait for Twofish; FSE; 2001
[68] *Algebraic Cryptanalysis* – C. Cid and R; Block Ciphers: Algebraic Cryptanalysis and Grobner Bases; Gröbner Bases, Coding, and Cryptography ; 2009; Springer; Pages 307-327
[69] *Interpolation Attack* - T. Jakobsen and L. Knudsen; The Interpolation Attack on Block Ciphers; FSE; 1997
[70] *XSL Attacks* - N. Courtois and J. Pieprzyk; Cryptanalysis of Block Ciphers with Overdefined Systems of Equations; 2002; http://eprint.iacr.org/2002/044
[71] *XLS Attack Analysis* - C. Cid and G. Leurent; An Analysis of the XSL algorithm; ASIACRYPT'05; LNCS Volume 3788; Pages 333-352
[72] *Cube Attack* - A. Shamir and I. Dinur; Cube Attacks on Tweakable Black Box Polynomials; 2008; http://eprint.iacr.org/2008/385.pdf
[73] *Cube Attack Controversy 1* - M. Vielhaber; Shamir's "cube attack": A Remake of AIDA;
http://tinyurl.com/lg33rt
[74] *Cube Attack Controversy 2* - D. Bernstein; Why haven't cube attacks broken anything;
http://cr.yp.to/cubeattacks.html
[75] *WEP* - IEEE Standard 802.11-1997; http://tinyurl.com/qapb3o
[76] *RC4 algorithm and attacks* – M. Stamp and R. Low; Applied Cryptanalysis: Breaking Ciphers in the Real World; Wiley; 2007; Pages 103-110
[77] *WEP Early Attack* – S. Fluhrer, I. Mantin, and A. Shamir; Weaknesses in the Key Scheduling Algorithm of RC4; 2001
[78] *RC4 Random Generator* - I. Mantin; Predicting and Distinguishing Attacks on RC4 Keystream Generator; EUROCRYPT05; volume 3494 of LNCS; pages 491–506
[79] *RC4 Recent Attack* - A. Klein; Attacks on the RC4 stream cipher; 2006
[80] *WEP Latest Attack* – E. Tews, A. Pychkine, and R.P. Weinmann; Breaking 104 bit WEP in less than 60 seconds; 2007
[81] *Aircrack-ptw* - http://www.cdc.informatik.tu-darmstadt.de/aircrack-ptw/
[82] *WEP T.J. Max Incident* – EETimes - http://tinyurl.com/rc69ve

[83] *LM Hash Attack* - C. Swenson; Modern Cryptanalysis: Techniques For Advanced Code Breaking; Willey Publishing; 2008; Page 158.
[84] *Ophcrack* - http://ophcrack.sourceforge.net/
[85] *Bluetooth Specification* - http://www.bluetooth.com/Bluetooth/Technology/
[86] *Bluetooth Related Algebraic Attack* – N. Courtois and W. Meier; Fast Algebraic Attacks on Stream Ciphers with Linear Feedback; 2003
[87] *Bluetooth Attack Summary* - D. Singelée and B. Preneel; STILL ONLY A DRAFT; Review of the Bluetooth Security Architecture; 2006
[88] *Bluetooth Correlation Attack* – J. Golic, V. Bagini and G. Morgari; Linear Cryptanalysis of Bluetooth Stream Cipher; EUROCRYPT'02
[89] *Bluetooth Conditional Correlation Attack* - Y. Lu, W. Meier, S. Vaudenay; The Conditional Correlation Attack:A Practical Attack on Bluetooth Encryption;
[90] *Barclays and wireless phone* payment – The Daily Mail Newspaper; http://tinyurl.com/5mbu5o
[91] *Bluetooth Security Guide* – K. Scarfone and J. Padgette; NIST guide to Bluetooth security; Special Publication 800-121; 2008
[92] *AES Candicate*– J. Daemen and V. Rijmen; AES Proposal: Rijndael; 1999 Also described in FIPS 197
[93] *AES Attack* – A. Biryukov and D. Khovratovich; Related-key Cryptanalysis of the Full AES-192 and AES-256; University of Luxembourg; 2009
[94] *AES Practical Cryptanalysis* – A. Biryukov, O. Dunkelman, N. Keller, D. Khovratovich, and A. Shamir; Key Recovery Attacks of Practical Complexity on AES Variants With Up To 10 Rounds; 2009; http://eprint.iacr.org/2009/374
[95] *French University Lyon* – L'Institut Universitaire de Technologie A - Lyon1 http://iut-a.univ-lyon1.fr/
[96] *UK University and BSc project* – Teesside University, School of Computing; http://www-scm.tees.ac.uk/
BUGS Project Website, http://www.encryptolutions.com
[97] *IEEE Newsletter* – Newsletter of the IEEE Computer Society's TC on Security and Privacy; Electronic Issue 28; 13 July 1998. http://tinyurl.com/r83xhc
[98] *Applied Cryptography book* – B Schneier; Applied Cryptography; Second Edition; John Wiley & Sons; 1996
[99] *ISAAC Random Generator* – B. Jenkins; http://burtleburtle.net/bob/rand/isaacafa.html
[100] *Cryptanalysis Scripts Contributor* – T. Martinez; INRIA France; http://contraintes.inria.fr/~tmartine
[101] *GPG* - http://www.gnupg.org/
[102] *Greedy Algorithm* - http://en.wikipedia.org/wiki/Greedy_algorithm#References
[103] *NSA Cryptologic Memorial* - http://tinyurl.com/l2j3eo
[104] *Imagemagick* - http://www.imagemagick.org/script/index.php