The National Centre for the Distinguished

A Practical Implementation of Linear Cryptanalysis

Jaber Adeeb - Khaled Ismaeel - Mahmoud Kara Fallah

5/5/2016

Abstract

In this project, we present a detailed mathematical study of linear cryptanalysis on substitution-permutation networks. Plus, we implement it on a specific SPN structure.

A Practical Implementation of Linear Cryptanalysis on Substitution-Permutation Networks

Khaled Ismaeel - Jaber Adeeb - Mahmoud Kara Fallah

A project submitted in mathematics

Supervised by Mr. Habeeb Issa

May 5, 2016

Contents

1 Introduction

1.1 Historical Overview

In the few past decades, mankind have witnessed a revolution in communication technology. Ranging from the increasing number of mobile phones to the ever growing Internet and many other aspects of our daily lives. This huge advancement came as a two-edged sword, because it opened up an entire new field of war, the so-called *cyber war*. The main target in this war is the information being sent across these communication networks. This led to the increasing demand for secrecy and information security. And so, we have been in a battle between the field of cryptography, which responsible for secret communication, and the field of cryptanalysis, the one who extracts the information from the network.

In these days, cryptography has outran cryptanalysis. We have various information security technologies that remain till this day unbroken. This is in fact thanks to cryptanalysts. They are the people responsible for exploiting the weaknesses in the security technologies, and help designing schemes that they are not capable of breaking. Ed Giorgio, the chief code maker and code breaker in the NSA states that, in the NSA they were a team of 17 cryptographers, and 1700 cryptanalysists. This resembles the level of information security level that we have reached.

In the mid 1970s, the data encryption standard, which is the first commercial-grade algorithm with fully open implementation detail was presented. This standard algorithm was a Feistel network with a 64-bit block size and a 56-bit key (in addition to 8 parity check bits). The DES remained unbroken for almost 20 years. Until the mid 1990s, when differential and linear cryptanalysis were used to break the cipher.

After the DES retirement, in 1997, there have been a public quest for the new standard algorithm, which will be called the *advanced encryption standard*. The new algorithm was expected to be at least as efficient as the DES, but immune to the attacks that broke the DES. Further, it must support a 128-bit block size and key lengths of 128-bit, 192-bit and 256-bit. And so, many algorithms have been proposed for becoming the AES. Their evaluation consisted of a series of pubic conferences and meetings. After almost 4 years of filtration, the algorithm Rijndael was chosen as the AES. Rijndael did not exhibit a Feistel network structure, but rather a substitution-permutation network, with a 128-bit block size and supports of the specified key lengths. Which was the main reason for the numerous studies for substitution-permutation ciphers.

1.2 This Project

This project is about linear cryptanalysis, which is a method used by Matsui [6] to break the Data Encryption Standard. Till this day, it remains the most powerful attack on the Data Encryption Standard. It has ever since attracted more and more attention due to its unique power against block ciphers. And here, we will apply it to a specific block cipher architecture, the substitution-permutation network.

For the study of linear cryptanalysis, we must first go through some concepts in block ciphers in general, and substitution-permutation networks in particular and their properties. In addition to a mathematical background in probability theory and boolean mappings. This will be the main concern of chapter 2.

Then, we go through the theoretical description of linear cryptanalysis and its application to substitution-permutation networks. This will be the topic of chapter 3.

Finally, in chapter 4, we will present the practical implementation of linear cryptanalysis over substitution-permutation networks with our results. The practical implementation includes a JAVA simulation of the SPN encryption and decryption. Beside a JAVA server application which will act as the insecure channel. Finally, an algorithm written also in JAVA that performs the attack on the cipher.

2 Background

In this chapter, we shall review general concepts about cryptology in general, and symmetric encryption schemes in particular, whose breaking is what this project is about.

Plus, we build a strong mathematical foundation, which is needed for the study of linear cryptanalysis.

2.1 Cryptography and Cryptanalysis

2.1.1 Motivating Problem

Suppose there are two parties, A and B. And these two parties wish to communicate across a channel. But the problem is that this channel is unsecure, such that every message that crosses this channel can be viewed by parties other than A and B. This scenario occurs very often in modern communication systems. For example, the mobile phone sends its information using radio waves, and these radio waves are sent in all directions, which means that everyone can get hold of the information sent by the mobile phone across this network.

The problem is that A and B may be exchanging sensitive information, such as a bank account password. Having other parties eavesdropping on the network causes a serious security issue.

To get around this problem, A uses an *encryption algorithm*. The encryption algorithm allows A to transform the message (called the *plaintext*) to another message that is not readable (called the *ciphertext*). This message is then sent via the network, where there is no danger, because the attacker cannot get any meaningful information out of it. On the other side, B uses an inverse algorithm

called the *decryption algorithm*, which allows B to reverse the encryption A performed on the message, yielding the original message.

This problem and its solution gave birth to cryptography. This science is nothing new, it was used by ancient Egyptians and Roman generals.

2.1.2 Cryptography

Cryptography can be defined as *The study of mathematical techniques and algorithms to provide information security services.* This field has evolved a lot since the ancient Romans, which gave us the many divisions that form the entire field of cryptography. The techniques and concepts used in cryptography are called *cryptographic primitives.* They are outlined in figure 1 [1].

With the combination of the cryptographic primitives, cryptography delivers *information security services.* There are four main services, which are the following [1]:

1. *Secrecy* (also called *Privacy* or *Confidentiality*): Protecting the information from attackers, just like the scenario in the motivating problem above. Note that it doesn't mean that the attacker cannot eavesdrop on the network, it rather means that he cannot extract the information from the message being sent.

2. *Integrity*: The protection of the information from unauthorized manipulation. This manipulation can be by either insertion, deletion or substitution. This service is crucial in many situations, like money transfer.

3. *Authentication*: The assurance that every party on the network is who/what they claim they are. And that every message sent through the network belongs to the claimed sender. Note that authentication implies integrity, since that data manipulation means that the message will no longer belong to its original sender.

4. *Non-repudiation*: The inability for a party to deny its previous actions and commitments.

The algorithms used to provide the cryptographic primitives can get very mathematically complex. Menezes, VanOorschot and Vanstone [1] give a detailed

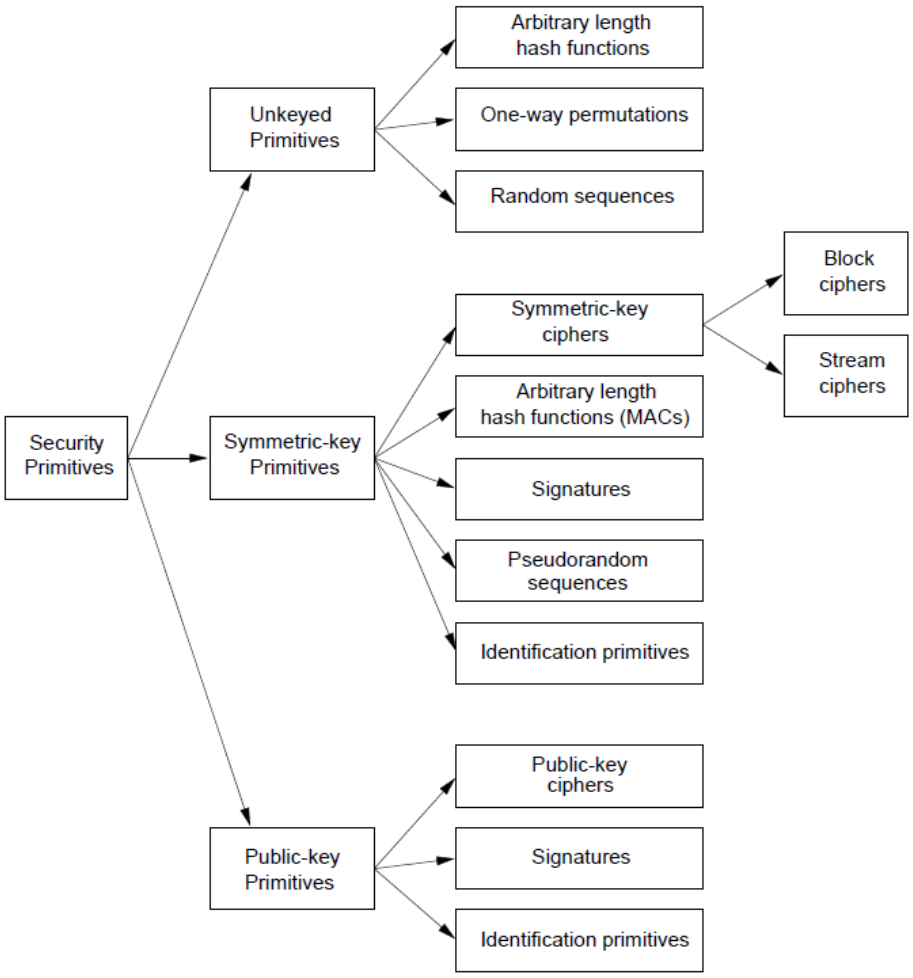

Figure 2.1: A taxonomy of cryptographic primitives

description about the algorithms used in the cryptographic primitives.

2.1.3 Cryptanalysis

Cryptanalysis can be defined as *the process of breaking the secure between parties across a channel*. Throughout history, cryptanalysis has succeeded several times, like the famous Enigma cipher and the English genius Alan Turing. Cryptanalysis is a very diverse field in mathematics and computer science, Paar [2] divided the field of cryptanalysis as follows:

1. *Classical Cryptanalysis*: Which is the process of recovering the plaintext given the ciphertext. And we distinguish between *analytical attacks*, which tend to exploit the internal weaknesses of the encryption method being used, and *brute force attacks*, which perform an exhaustive search on all the possible keys.

2. *Implementation Attacks* (Also called *Side Channel Attacks*): Where the attacker tries to extract the key or the plaintext by eavesdropping on side channels other than the one where the message is being sent, which could give the attacker valuable information about the encryption process. For example, recording the power consumption or electromagnetic emission of the processor while performing encryption.

3. *Social Engineering Attacks*: Bribe, blackmail or threat where human beings are involved in order to get hold of the key.

In this project we are interested in analytical attacks, where we will exploit a pattern in the SPN encryption, and use it to extract information about the key being used.

2.2 Ciphers

Let's consider the motivating problem in section 2.1.1. the sender has a large file to transmit to the receiver. The sender breaks this file into small pieces called plaintexts (usually of fixed length). Each one of these plaintexts p_i is input into an encryption algorithm, which takes an encryption key $\mathbf{k}_e$, as a

parameter to give a ciphertext c_i as an output. The ciphertext is sent over the aforementioned insecure channel, and the receiver recovers the plaintext using a decryption algorithm, which takes a decryption key $\mathbf{k}_d$ as a parameter. Then the plaintexts are reassembled into the original file. The term cipher or encryption scheme refers to the parameterized encryption/decryption algorithms.

This section is derived from [1], [2] and [4].

2.2.1 Symmetric and Asymmetric Ciphers

In a symmetric-key cipher, both the encryption and decryption keys are equal or are easily derived from each other. We will assume that $\mathbf{k}_e = \mathbf{k}_d$. Clearly this scheme requires the establishment of a shared key between the sender and the receiver, which raises the key-distribution problem.

Public-key ciphers on the other hand provide a safer and more secure method to encrypt data. This method requires a key pair, the encryption key (the public key) which can be widely distributed, and the decryption key which is generally known only to the receiver. So anybody can use the encryption key to encrypt information and send it to the receiver, which is the only one capable of using the decryption key to decrypt the information. We conclude that public-key ciphers provide an elegant solution to the key distribution problem. However, they are much slower than symmetric-key ciphers (approximately 1/1000 the speed) and requires much longer keys to achieve the same level of security. As a result, hybrid techniques incorporating symmetric-key and public-key ciphers were created. For example, one party can randomly generate a key to be used in a symmetric-key cipher. This key is then encrypted with the second party's public key, and sent over the channel. The second party then decrypts this key with the corresponding private key, and then both parties switch to an agreed-upon symmetric-key cipher using this key.

2.2.2 Block and Stream Ciphers

Symmetric-key ciphers could be further categorized into block ciphers and stream ciphers. A block cipher is a bijective mapping from $\{0,1\}^N$ to $\{0,1\}^N$, parameterized by a key $\mathbf{k} \in \{0,1\}^K$ (N is called the block size). Block sizes typically are

64, 128, with typical key lengths 128, 192, 256 (key lengths of 56 and 64 bits are common in older block ciphers). A block cipher has the obvious feature that, for a fixed key, a given plaintext will always map to the same ciphertext (as in the straightforward application of a block cipher, called electronic codebook (ECB) mode). There are other block cipher modes for which this no longer applies.

A stream cipher breaks a message to be encrypted into much smaller plaintexts, typically individual bits (sometimes bytes), $x_1, x_2, x_3 \ldots$ the key, k, is expanded into a keystream, $z_1, z_2, z_3 \ldots$ the i^th ciphertext is obtained by combining x_i and z_i according to some rule (often the XOR operation, which requires that x_i and z_i have the same number of bits). It follows that identical plaintexts do not always encrypt to identical ciphertexts. A stream cipher derives its name from the fact that it can be viewed as processing its input as a continuous stream.

2.3 Mathematical Background

As we shall see later, linear cryptanalysis is a probabilistic attack. That is, it doesn't discover the key, it rather discovers the key with the largest probability of being the correct key. That's why we need to introduce some concepts in probability theory that are needed for implementing the attack.

Plus, modern block ciphers rely heavily on the concept of the boolean mappings, that's why there have been heavy research on the properties of boolean mappings and their interrelations. Here, we shall discuss only a limited number of concepts, the ones that are important for the SPN structure and its linear cryptanalysis.

In this section, we build the mathematical foundation needed for successfully building a SPN, and for implementing linear cryptanalysis on it.

2.3.1 Probability Theory

Definition 2.3.1. *Let Ω be a set, and let $\mathcal{A} \subseteq \mathcal{P}(\Omega)$. We call $\mathcal{A}$ a* complete boolean algebra *or* complete algebra *over Ω if and only if all of the following is true:*

1. $\Phi \in \mathcal{A}$

2. $\forall A \in \mathcal{A}, \bar{A} \in \mathcal{A}$

3. *For any sequence* $(A_n)_{n\in\mathbb{N}}$ *such that* $\forall n \in \mathbb{N}, A_n \in \mathcal{A}$ *we have:*

$$\bigcup_{n\in\mathbb{N}} A_n \in \mathcal{A}$$

Lemma 2.3.2. *If* A, B *are two complete algebras, then* $A \cap B$ *is a complete algebra.*

Definition 2.3.3. *Let* Ω *be a set and let* $\mathcal{C} \subseteq \mathcal{P}(\Omega)$*. We define the* sigma algebra of $\mathcal{C}$ over Ω*, denoted* $\sigma(\Omega)$*, as the intersection of all complete algebras containing* $\mathcal{C}$ *over* Ω*.*

Definition 2.3.4. *We define the* sigma algebra of Borel sets*, denoted* $\mathcal{B}_{\mathbb{R}}$*, as the algebra* $\sigma(\mathcal{I})$ *over* $\mathbb{R}$*, where* $\mathcal{I}$ *is the set of all domains of real numbers.*

Definition 2.3.5. *Let* Ω *be a set, and let* $\mathcal{A}$ *be a complete algebra of parts of* Ω*. We define the* probability $\mathbb{P}$ *over* $(\Omega, \mathcal{A})$ *as every function that satisfies the two following conditions:*

1. $\mathbb{P}(\Omega) = 1$

2. *For any sequence* $(A_n)_{n\in\mathbb{N}}$ *such that* $\forall n \in \mathbb{N}$ *,*$A_n \in \mathcal{A}$*, and* $\forall (i,j) \in \mathbb{N}^2$*,* $A_i \cap A_j = \Phi$ *we have:*

$$\mathbb{P}(\bigcup_{n\in\mathbb{N}} A_n) = \sum_{n=0}^{\infty} \mathbb{P}(A_n)$$

Definition 2.3.6. *We call the triplet* $(\Omega, \mathcal{A}, \mathbb{P})$ *a* probability space*. Let* $A, B \in \mathcal{A}$*, the following is true:*

1. $\mathbb{P}(\bar{A}) = 1 - \mathbb{P}(A)$

2. $A \cap B = \Phi \Rightarrow \mathbb{P}(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B)$

3. $A \subset B \Rightarrow \mathbb{P}(A) \leq \mathbb{P}(B)$

4. $\mathbb{P}(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A \cap B)$

Definition 2.3.7. *Let* $(\Omega, \mathcal{A}, \mathbb{P})$ *be a probability space, and let* $A, B \in \mathcal{A}$ *where* $\mathbb{P}(B) \neq 0$*. We define the* conditional probability of A given B *as follows:*

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}$$

Definition 2.3.8. *Let $(\Omega, \mathcal{A}, \mathbb{P})$ be a probability space, we call a* random variable *in $(\Omega, \mathcal{A}, \mathbb{P})$ any function $X : \Omega \to \mathbb{R}$ such that:*

$$\forall B \in \mathcal{B}_{\mathbb{R}}, X^{-1}(B) \in \mathcal{A}$$

We call X a discrete random variable *if and only if Ω was finite or countable.*

We use the notation $\{X \in B\}$ instead of $X^{-1}(B)$. Beside, if the argument for a probability is a mathematical proposition, we usually use square brackets for the probability.

Definition 2.3.9. *Let $(\Omega, \mathcal{A}, \mathbb{P})$ be a probability space, and let X be a random variable in $(\Omega, \mathcal{A}, \mathbb{P})$. We define the* cumulative distribution function *of X as the function F_X which satisfies:*

$$F_X : \mathbb{R} \to [0, 1] : x \mapsto \mathbb{P}\left[\omega \in \Omega : X(\omega) \leq x\right]$$

Lemma 2.3.10. *The following is true:*

1. *F is non-decreasing.*
2. *F is right-continuous.*
3. $\lim_{x \to -\infty} F(x) = 0$, $\lim_{x \to \infty} F(x) = 1$.

Definition 2.3.11. *Let $(\Omega, \mathcal{A}, \mathbb{P})$ be a probability space, and let X be a random variable in $(\Omega, \mathcal{A}, \mathbb{P})$. We define the* probability law *of X as the probability $\mathbb{P}_X$ over $(\mathbb{R}, \mathcal{B}_{\mathbb{R}})$ which satisfies:*

$$\forall B \in \mathcal{B}_{\mathbb{R}}, \mathbb{P}_X(B) = \mathbb{P}\left[X \in B\right]$$

Definition 2.3.12. *Let $(\Omega, \mathcal{A}, \mathbb{P})$ be a probability space, and let X be a random variable in $(\Omega, \mathcal{A}, \mathbb{P})$. We say that X has a* probability density *if and only if there exits an integrable function $f_X : \mathbb{R} \to \mathbb{R}_+$ which satisfies:*

$$\forall x \in \mathbb{R}, F_X(x) = \int_{-\infty}^{x} f_X(t)dt$$

We call f_X the probability density function *of X.*

Definition 2.3.13. *Let $(\Omega, \mathcal{A}, \mathbb{P})$ be a probability space, and let X be a discrete random variable in $(\Omega, \mathcal{A}, \mathbb{P})$. By definition of a discrete random variable there*

exist a bijection $\Phi : \mathcal{N} \to \Omega : k \mapsto x_k$*,where* $\mathcal{N} \subset \mathbb{N}$*. We define the* probability mass function of X *as the function* F_X *which satisfies:*

$$F_X : X(\Omega) \to [0,1] : x \mapsto \mathbb{P}(\omega \in \Omega : X(w) = x)$$

Definition 2.3.14. *Let* $(\Omega, \mathcal{A}, \mathbb{P})$ *be a probability space, and let* X *be a discrete random variable in* $(\Omega, \mathcal{A}, \mathbb{P})$*. We define the* expected value of X *as:*

$$\mathbb{E}[X] = \sum_{x \in X(\Omega)} x \cdot \mathbb{P}_X(x)$$

And we define the variance of X *as:*

$$var[X] = \mathbb{E}\left[[X - \mathbb{E}[X]]^2\right]$$

2.3.2 Boolean Mappings

Boolean mappings are a special family of functions of the form $f : \{0,1\}^N \to \{0,1\}^N; N \in \mathbb{N}$. The importance of this concept is that a block ciphers components, such as the rounds and permutations, are usually represented as a boolean mapping. Note that a boolean mapping $B : \{0,1\}^N \to \{0,1\}^N; N \in \mathbb{N}$ can be represented as a combination of N functions $f_i : \{0,1\}^N \to \{0,1\}; 1 \leq i \leq N$

Linear Probability

Definition 2.3.15. *Let* $B : \{0,1\}^N \to \{0,1\}^N; N \in \mathbb{N}$ *be a bijective boolean mapping and* $\mathbf{a}, \mathbf{b} \in \{0,1\}^N$ *be fixed. The* linear probability of masks $\mathbf{a}, \mathbf{b}$*, denoted by* $LP(\mathbf{a}, \mathbf{b})$ *is defined as:*

$$LP(\mathbf{a}, \mathbf{b}) \stackrel{def}{=} \mathbb{P}_{\mathbf{X}}[\mathbf{a} \cdot \mathbf{X} = \mathbf{b} \cdot B(\mathbf{X})]$$

where $\mathbf{X}$ *is a random variable uniformly distributed over* $\{0,1\}^N$*. If* B *if parameterized by a key* $\mathbf{k}$*, we define it as:*

$$LP(\mathbf{a}, \mathbf{b}; \mathbf{k}) \stackrel{def}{=} \mathbb{P}_{\mathbf{X}}[\mathbf{a} \cdot \mathbf{X} = \mathbf{b} \cdot B(\mathbf{X}; \mathbf{k})])$$

and the expected linear probability of masks $\mathbf{a}, \mathbf{b}$ *is defined as:*

$$ELP(\mathbf{a}, \mathbf{b}) \stackrel{def}{=} \mathbb{E}_{\mathbf{K}}[LP(\mathbf{a}, \mathbf{b}; \mathbf{k})]$$

where $\mathbf{K}$ *is a random variable uniformly distributed over the space of keys.*

The values LP, ELP can be viewed as entries in a $2^N \times 2^N$ table, where the element in row $\mathbf{a}$ and column $\mathbf{b}$ [1] is $LP(\mathbf{a}, \mathbf{b}), ELP(\mathbf{a}, \mathbf{b})$ with $\mathbf{a}, \mathbf{b} \in \{0,1\}^N$. Rijmen and Daemen [3] called $\mathbf{a}, \mathbf{b}$ *selection patterns*. Here, we use Keliher's [4] terminology and use the term *masks*.

Even though LP values are not actual probabilities, we use the term *probability* because they are bounded in the interval $[0, 1]$. The reason behind this definition is that cryptologists are interested in masks $\mathbf{a}, \mathbf{b}$ that either maximize or minimize $P_{\mathbf{X}}[\mathbf{a} \cdot \mathbf{X} = \mathbf{b} \cdot B(\mathbf{X})]$, since both of these situations mean that a pattern has been found to correlate the input and output of this mapping. In both these cases, the corresponding LP value will be close to 1. However, if the masks $\mathbf{a}, \mathbf{b}$ are chosen in a way that doesn't represent the correlation between the input and output, then the value $P_{\mathbf{X}}[\mathbf{a} \cdot \mathbf{X} = \mathbf{b} \cdot B(\mathbf{X})]$ will be close to $1/2$, and the corresponding LP value will be close to 0.

Keliher [4] derived the following lemma from Parseval's theorem [5].

Lemma 2.3.16. *Let* B *be a bijective boolean mapping parameterized by a key* $\mathbf{k}$*, and* $\mathbf{a}, \mathbf{b} \in \{0,1\}^N$ *are fixed. Then:*

$$\sum_{\mathbf{x} \in \{0,1\}^N} LP(\mathbf{a}, \mathbf{x}; \mathbf{k}) = \sum_{\mathbf{x} \in \{0,1\}^N} LP(\mathbf{x}, \mathbf{b}; \mathbf{k}) = 1$$

$$\sum_{\mathbf{x} \in \{0,1\}^N} ELP(\mathbf{a}, \mathbf{x}) = \sum_{\mathbf{x} \in \{0,1\}^N} ELP(\mathbf{x}, \mathbf{b}) = 1$$

Linear probabilities are the main tool that Matsui [6] used to perform linear cryptanalysis. Now we come the concept of differential probability, which is the tool used by Shamir and Biham [7] to perform *differential cryptanalysis*.

Differential Probability

Definition 2.3.17. *Let* $B : \{0,1\}^N \to \{0,1\}^N$ *be a bijective boolean mapping, and* $\Delta\mathbf{x}, \Delta\mathbf{y} \in \{0,1\}^N$ *are fixed. We define the* differential probability of $\Delta\mathbf{x}, \Delta\mathbf{y}$*, denoted by* $DP(\Delta\mathbf{x}, \Delta\mathbf{y})$ *as:*

$$DP(\Delta\mathbf{x}, \Delta\mathbf{y}) \stackrel{def}{=} \mathbb{P}_{\mathbf{X}}[B(\mathbf{X}) \oplus B(\mathbf{X} \oplus \Delta\mathbf{x}) = \Delta\mathbf{y}]$$

[1] We can index the table with vectors if we view the binary vectors $\mathbf{a}, \mathbf{b}$ as integers written in binary.

where $\mathbf{X}$ *is a random variable uniformly distributed over* $\{0,1\}^N$. *If* B *is parametrized by a key* $\mathbf{k}$, *we define it as:*

$$DP(\Delta\mathbf{x}, \Delta\mathbf{y}) \stackrel{def}{=} \mathbb{P}_{\mathbf{X}}\left[B(\mathbf{X};\mathbf{k}) \oplus B(\mathbf{X} \oplus \Delta\mathbf{x};\mathbf{k}) = \Delta\mathbf{y}\right]$$

and the expected differential probability, *denoted* $EDP(\Delta\mathbf{x}, \Delta\mathbf{y})$, *is defined as:*

$$EDP(\Delta\mathbf{x}, \Delta\mathbf{y}) \stackrel{def}{=} \mathbb{P}_{\mathbf{K}}\left[DP(\Delta\mathbf{x}, \Delta\mathbf{y};\mathbf{K})\right]$$

where $\mathbf{K}$ *is a random variable uniformly distributed over the space of keys.*

In the same way as in linear probability, the DP, EDP values can be viewed in a $2^N \times 2^N$ table. Many researches refer to the *XOR Table* instead, denoted by $XOR(,)$. In the XOR table, entry $\boldsymbol{\Delta}\mathbf{x}, \boldsymbol{\Delta}\mathbf{y}$ is given by:

$$XOR(\Delta\mathbf{x}, \Delta\mathbf{y}) = \#\left\{\mathbf{X} \in \{0,1\}^N | B(\mathbf{X}) \oplus B(\mathbf{X} \oplus \Delta\mathbf{x}) = \Delta\mathbf{y}\right\}$$

Both tables are equally valuable, since:

$$DP(\Delta\mathbf{x}, \Delta\mathbf{y}) = \frac{1}{2^N} XOR(\Delta\mathbf{x}, \Delta\mathbf{y})$$

Vaudenay [8] set the following theorem, which links the linear probability with the differential probability.

Theorem 2.3.18. *Let* $B : \{0,1\}^N \to \{0,1\}^N$ *be a bijective boolean mapping, and* $\mathbf{a}, \mathbf{b}, \Delta\mathbf{x}, \Delta\mathbf{y} \in \{0,1\}^N$ *are fixed. Then:*

$$LP(\mathbf{a}, \mathbf{b}) = \frac{1}{2^N} \sum_{\mathbf{u},\mathbf{v} \in \{0,1\}^N} (-1)^{(\mathbf{a}\cdot\mathbf{u})+(\mathbf{b}\cdot\mathbf{v})} DP(\mathbf{u}, \mathbf{v})$$

$$DP(\Delta\mathbf{x}, \Delta\mathbf{y}) = \frac{1}{2^N} \sum_{\mathbf{u},\mathbf{v} \in \{0,1\}^N} (-1)^{(\Delta\mathbf{x}\cdot\mathbf{u})+(\Delta\mathbf{y}\cdot\mathbf{v})} LP(\mathbf{u}, \mathbf{v})$$

Algebraic Degree

Any boolean function $f : \{0,1\}^N \to \{0,1\}$ can be written as a polynomial of the inputs components:

$$f(x_1, x_2, \ldots, x_N) = a + \sum_{1 \le i \le N} a_i x_i + \sum_{1 \le i \le j \le N} a_{i,j} x_i x_j + \cdots + a_{1,2,\ldots N} x_1 x_2 \cdots x_N$$

where multiplication and addition refer to boolean AND and XOR operations, respectively. This form of the boolean function f is called the *algebraic normal form of* f. It can be obtained by a simple operation called the *algebraic normal transform*. The *degree* of a term in the algebraic normal form of a boolean function is the number of distinct x_is in it. The degree of a boolean function f is the highest degree of its terms in the algebraic normal form.

Let $B : \{0,1\}^N \to \{0,1\}^N$ be a boolean mapping. Note that B can be viewed as a combination of N boolean functions $f_i : \{0,1\}^N \to \{0,1\}; 1 \leq i \leq N$, where:

$$B(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_N(\mathbf{x})); \mathbf{x} \in \{0,1\}^N$$

We write that as $B = (f_1, f_2, \ldots f_N)$. With that said, we define the degree of a boolean mapping B as the highest degree of its constituent boolean functions.

Block cipher designers are interested in the degree of a boolean mapping and try to make it as high as possible, because block ciphers whose components have a low algebraic degree may be vulnerable to *higher order differential cryptanalysis*.

Completeness

Kam and Davida [9] defined the property of completeness.

Definition 2.3.19. *Let* $B : \{0,1\}^N \to \{0,1\}^N$. *Then* B *is called* complete *if every output bit depends on every input bit. Formally,* B *is complete if and only if, for every* $1 \leq i, j \leq N$ *there exists* $\mathbf{x}, \mathbf{y} \in \{0,1\}^N$ *such that* $\mathbf{x}$ *and* $\mathbf{y}$ *differ in exactly bit* i, *and* $B(\mathbf{x})$ *and* $B(\mathbf{y})$ *differ in at least bit* j.

Clearly, if $B : \{0,1\}^N \to \{0,1\}^N$ is complete, then, in the algebraic normal form of every $f_i; 1 \leq i \leq N$, each $x_i; 1 \leq i \leq N$ occurs at lest once in a non-zero coefficient term.

The following lemma is due to Keiher [4].

Lemma 2.3.20. *Let* $B : \{0,1\}^N \to \{0,1\}^N$ *be linear and invertible, then* B *cannot be complete.*

Proof Suppose, by the way of contradiction, that B is complete. Let's put $B = (f_1, f_2, \ldots f_N)$. Since B is linear, then in each constituent function $f_i; 1 \leq i \leq N$, each term with non-zero coefficient has degree 1. And since B is complete, then in each constituent function $f_i; 1 \leq i \leq N$, each $x_i; 1 \leq i \leq N$ must appear at

least once. Then:

$$f_j(x_1, x_2, \ldots, x_N) = \sum_{1 \leq i \leq N} x_i$$

That is, all f_i are identical, contradicting the invertibility of B. Therefore B cannot be complete.

2.4 Block Cipher Architectures

Modern block ciphers could be traced back to a landmark paper by Claude Shannon in which the principles of confusion and diffusion were outlined. Confusion is the obscuring of relationship between the plaintext and the ciphertext. Diffusion involves "spreading out" patterns in the plaintext so that they are no longer detectable in the ciphertext. Shannon suggested that confusion and diffusion could be achieved through the use of substitution and linear transformation, respectively. The two main block cipher architectures, substitution-permutation networks and Feistel networks, both use substitution and linear transformation to implement Shannon's principles. They are also good examples of product ciphers, which are constructed by composing two or more encryption operations. An iterated cipher is a product cipher that consists of repeated application of the same encryption step, called a round. A round may itself consist of multiple encryption steps; in general, different keying material is used in each round.

2.4.1 Key Schedule Algorithms

Keying material, in most block ciphers, is mixed with the intermediate block in each round. A separate key-scheduling algorithm is used to generate a series of subkeys (or round keys) from the original key, $\mathbf{k}$ (sometimes called the master key). A poorly designed key-scheduling algorithm may introduce significant weaknesses into a cipher opening the door for certain attacks (related-key attacks). Many cryptographically strong key-scheduling algorithms are designed by incorporating features of the cipher itself (this approach is used by the AES, Camellia, Twofish and Serpent, among others).

It is often prudent for a cryptanalyst to assume the use of an independent key, since an attack is successful in this model may have a higher success rate when there are correlations among the subkeys. As well as the assumption of

an independent key is frequently made by a cipher designer when evaluating resistance to various attacks, but the features of the key-scheduling algorithm should also be given careful consideration.

2.4.2 Feistel Ciphers

A Feistel network is a block cipher that modifies half a block in each round, which requires an even block size. Unlike an SPN a Feistel network round uses R subkeys. Let the left and right halves of the N-bit input to round r be denoted x_L^r and x_R^r, respectively. The right half, x_R^r, becomes the input to a round function $f_r : \{0,1\}^{N/2} \rightarrow \{0,1\}^{N/2}$, which also takes $\mathbf{k}_r$ as a parameter. The output from f_r is XOR'd with x_L^r to form x_R^{r+1} (the right half of the input to the next round), while x_R^r is preserved unchanged as x_L^{r+1}. This swapping of half blocks occurs in every round except the last.

Figure 2.2, taken from [4], explains the basic structure of a Feistel network:

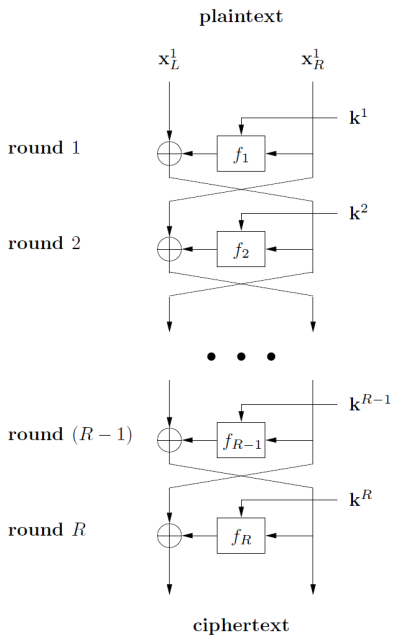

Figure 2.2: General structure of Feistel ciphers

An advantage of the Feistel network structure is that, when implementing it,

encryption and decryption are essentially identical operations—a ciphertext is decrypted by processing it through the encryption algorithm, but reversing the order of the round functions (and corresponding subkeys). If the same round function is used in each round, which is a common approach, only the order of the subkeys needs to be reversed. This eliminates the need to generate/store inverse components.

2.4.3 Substitution-Permutation Networks

An R-round substitution-permutation network (SPN) requires (R+1) N-bit subkeys, $k^1, k^2, \ldots, k^R, k^{R+1}$. Each round consists of three stages, or layers. In the key mixing stage, the N-bit round input is bitwise XOR'd with the subkey for that round. In the substitution stage, the resulting block is partitioned into M subblocks of size $n(N = Mn)$, and each subblock becomes the input to a bijective $n \times n$ substitution box (s-box)—a bijective mapping from $\{0,1\}^n$ to $\{0,1\}^n$. In the linear transformation stage, the output from the substitution stage is processed through an invertible N-bit linear transformation. (Classically, the linear transformation was a bitwise permutation, hence the origin of the name substitution-permutation network). The linear transformation is usually omitted from the last round, since it is easily shown that its inclusion adds no cryptographic strength to the SPN. A final subkey, k^{R+1}, is XOR'd with the output of round R to form the ciphertext.

Decryption is accomplished by running the SPN "backwards." Subkey K^{R+1} is first XOR'd with the ciphertext, and then in each round r (from R down to 1), the inverse linear transformation is applied, followed by the inverse s-boxes, and the resulting block is XOR'd with $\mathbf{k}_r$.

2.4.4 Other Block Cipher Architectures

Besides SPNs and Feistel networks, there is a number of ciphers that do not adhere to either structure. However, most of these ciphers are constructed from repeated rounds. For example, In the block cipher IDEA, which has a 64-bit block size and consists of 8 rounds, the round input is split into four 16-bit words, and these are combined with each other and with six 16-bit subkeys using a combination of binary operations on $0, 1^16$ from different algebraic groups.

IDEA has been extensively analyzed and widely implemented; it appears that this mixing of algebraic groups is a good source of security. The block cipher RC6 has a 128-bit block size and consists of 20 rounds. RC6 can essentially be viewed as two 64-bit Feistel networks operating in parallel, with interactions occurring in each round. RC6 makes extensive use of data-dependent rotations—bitwise rotations of data words in which the amount of rotation depends on other intermediate values.

2.4.5 The True Random Cipher

For a given block size N, the true random cipher (or ideal cipher) is the key-parameterized family of all bijective mappings from $\{0,1\}^N$ to $\{0,1\}^N$ such that each mapping is realized by exactly one key. For common block sizes, the true random cipher cannot be practically implemented since it would require a key of astronomical length (approximately $N \times 2^N$ bits). However, the true random cipher is important theoretically, and is generally considered to be the ideal block cipher model.

3 Linear Cryptanalysis

Linear cryptanalysis was invented by Matsui [6]. Originally, it was intended for the DES cipher, and it remains the most powerful attack on the DES to date. Even thougn DES is a Feistel cipher, linear cryptanalysis was later generalized over a variety of block cipher architectures, including SPNs. Heys [10] gave a simplified approach for linear cryptanalysis of SPNs. Keliher [4] (Which is the main reference for this project) studied linear cryptanalysis over a general form of block ciphers, one that included both Feistel ciphers and SPNs.

In this chapter, we shall introduce the concept of linear cryptanalysis of *Markov ciphers*, which are a general form of substitution-permutation networks. Later, we specify this attack on a specific SPN structure.

3.1 Markov Ciphers

Definition 3.1.1. *Let $\{X_i\}_{i\geqslant 0}$ be a sequence of discrete random variables. Then $\{X_i\}_{i\geqslant 0}$ is called a* Markov chain *if and only if, for every $i \geqslant 1$ and every fixed $\alpha_j; j \geqslant 0$:*

$$\mathbb{P}\left[X_i = \alpha_i \middle| X_{i-1} = \alpha_{i-1}, \ldots, X_0 = \alpha_0\right] = \mathbb{P}\left[X_i = \alpha \middle| X_{i-1} = \alpha_{i-1}\right]$$

That is, the X_i depends on only X_{i-1}, not on any other random variable in the sequence.

Definition 3.1.2. *A Markov chain is called* homogeneous *if and only if, for every $i, j \geqslant 1$ and every fixed α, β:*

$$P(X_i = \alpha | X_{i-1} = \beta) = P(X_j = \alpha | X_{j-1} = \beta)$$

Definition 3.1.3. *Let $\Psi : \{0,1\}^N \to \{0,1\}^N$ be an R round block cipher, where the the round r function is given by $\mathbf{y}_r = \psi_r(\mathbf{x}_r; \mathbf{k}_r)$, where $\mathbf{x}_r, \mathbf{y}_r, \mathbf{k}_r$ are the*

input, output, and key for round r, respectively. Then Ψ is a Markov cipher *if and only if, for $1 \leq r \leq R$, and every $\mathbf{x}, \Delta\mathbf{x}, \Delta\mathbf{y} \in \{0,1\}^N$:*

$$P_{\mathbf{K}}\left[\psi(\mathbf{x};\mathbf{K}) \oplus \psi(\mathbf{x} \oplus \Delta\mathbf{x};\mathbf{K}) = \Delta\mathbf{y}\right] = P_{\mathbf{X},\mathbf{K}}\left[\psi(\mathbf{X};\mathbf{K}) \oplus \psi(\mathbf{X} \oplus \Delta\mathbf{x};\mathbf{K}) = \Delta\mathbf{y}\right]$$

The connection between Markov chains and Markov ciphers is in the next theorem, which is due to Keliher [4].

Theorem 3.1.4. *Given an R round block Markov cipher, consider the encryption of pairs of plaintexts. Let $\Delta\mathbf{Y}_r \in \{0,1\}^N; 0 \leq r \leq R$ be random variables, where $\Delta\mathbf{Y}_0$ represents the XOR of the pair of plaintexts, and $\Delta\mathbf{Y}_r; 1 \leq r \leq R$ represents the XOR of the outputs of round r for the pair of plaintexts. If the plaintexts are chosen independently, and the subkeys are uniformly distributed over the space of subkeys, then $\Delta\mathbf{Y}_r; 0 \leq r \leq R$* is a homogeneous Markov chain.

The concept of Markov ciphers provides a general context in which linear cryptanalysis has been developed in. After the study of linear cryptanalysis of Markov ciphers, it is not difficult to specialize the operation for SPNs.

In fact, Markov ciphers were originally used for the study of differential cryptanalysis, which is based on the idea of XOR differences. But this is relevant for linear cryptanalysis too, since there is a close a connection between the two attacks in theorem 2.3.18.

3.2 Linear Cryptanalysis of Markov Ciphers

Linear cryptanalysis is a *known plaintext attack*. That is, it assumes that the attacker has at disposal a certain number of plaintexts and their corresponding ciphertext when encrypted using the target cipher. There are several methods by which these plaintext-ciphertext pairs have been obtained. For example, many files sent over a network contain standard header information, such as the receiver identity. Or, some parties over a network might send identical data for several clients, such as emails. Eavesdropping on these channels may yield known plaintext-ciphertext pairs.

3.2.1 A Straightforward Approach

Matsui [6] presented two algorithms that can be used to apply cryptanalysis. the first one, called *Algorithm 1*, can be used to extract one bit of information about the key being used. The other, called *Algorithm 2*, can be used to extract either one or both of $\mathbf{k}_1, \mathbf{k}_R$. Here, we focus on algorithm 2, and use it to extract $\mathbf{k}_1$. Once $\mathbf{k}_1$ is known, round 1 can be stripped off, and we are left with an $R-1$ round cipher to break. Then we reapply algorithm 2 to the remaining rounds, until the entire set of subkeys known.

Let's view the round 2 through R as a single boolean mapping $\Psi_2^R : \{0,1\}^N \rightarrow \{0,1\}^N$ parameterzied by key $\mathbf{k}_2^R = \langle \mathbf{k}_2, \mathbf{k}_3, \ldots, \mathbf{k}_R \rangle$. We want to find masks $\mathbf{a}, \mathbf{b} \in \{0,1\}^N$ that maximize $LP(\mathbf{a}, \mathbf{b}; \mathbf{k}_2^R)$.

Let's denote to the number of plaintext-ciphertext pairs at disposal P, and call the pairs $\langle p_i, c_i \rangle ; 1 \leq i \leq \mathrm{P}$. Matsui [6] makes the assumption that P is inversely proportional to $LP(\mathbf{a}, \mathbf{b}, \mathbf{k}_2^R)$. i.e.

$$\mathrm{P} = \frac{c}{LP(\mathbf{a}, \mathbf{b}, \mathbf{k}_2^R)}$$

for some constant c. Keliher [4] gave the success rate of algorithm 2 for various values of the constant c, as shown in the table below:

c	8	16	32	64
Success rate	48.6%	78.5%	96.7%	99.9%

Table 3.1: Success rate for Algorithm 2

Which are the same as the values given by Matsui [6] multiplied by 4, since Matsui used *bias* values, not LP.

Once masks $\mathbf{a}, \mathbf{b}$ have been found, the attacker tries all possible values for the round 1 subkey, each values called a *guess*. And for each guess $\mathbf{k}$, round 1 is completely known, so the attacker encrypts every known plaintext p_i to get a corresponding round 1 ciphertext $\mathbf{x}_{1|\mathbf{k}}$. Every time $\mathbf{a} \cdot \mathbf{x}_{1|\mathbf{k}} = \mathbf{b} \cdot \psi_2^R(\mathbf{x}_{1|\mathbf{k}}; \mathbf{k}_2^R)$, we increment a counter specified for every guess, denoted $C_{\mathbf{k}}$ (which was initialized to 0).

Pascal Junod [11] defined the *wrong key randomization hypothesis*. Which states

that, if $\mathbf{k}_T, \mathbf{k}_F$ were right and wrong guesses, respectively, then:

$$\frac{|C_{\mathbf{k}_T}/\mathrm{P} - 1/2|}{|C_{\mathbf{k}_F}/\mathrm{P} - 1/2|} \gg 1$$

Which can also be expressed in the form:

$$\frac{(2C_{\mathbf{k}_T} - \mathrm{P})^2}{(2C_{\mathbf{k}_F} - \mathrm{P})^2} \gg 1$$

Based on the above hypothesis, we take the guess that maximizes $(2C_{\mathbf{k}_T} - \mathrm{P})^2$ to be the correct subkey $\mathbf{k}_1$.

Since cryptanalysis in a probabilistic attack, there may be rare cases where the key randomization hypothesis doesn't hold true. This problem can be solved easily by using the *key ranking technique* [12]. Here, we sort the guesses $\mathbf{k}_r$ in non-increasing order $\mathbf{k}_1^g, \mathbf{k}_2^g, \ldots$. Then try the guesses starting from $\mathbf{k}_1^g$.

3.2.2 A Modified Approach

There is a problem in the straightforward approach, The number of guesses is the same as the number of round 1 subkeys. The problem is that we have to maintain a counter for every guess, which is computationally infeasible for typical subkey sizes. To get around this problem, Matsui [6] presented a modified approach.

The computation of $\mathbf{a} \cdot \psi(p_i; \mathbf{k}_r)$ only requires knowledge of the bits in $\psi(p_i; \mathbf{k}_r)$ for which the corresponding bits in $\mathbf{a}$ are 1. Therefore, it suffices to keep a counter for a subset of round 1 subkeys, those that effect the bits in $\psi(p_i; \mathbf{k}_r)$ that correspond to 1s in $\mathbf{a}$, which Mastui called *effective key bits*.

Once a correct guess for the subkey bits has been found, there are several techniques for extracting the full subkey. These techniques vary depending of the structure of round 1. Matsui successfully extracted 26 effective key bits. Due to the simplicity of of the DES key schedule, 26 bits of the full key were discovered. At this point, an exhaustive search on the remaining 30 key bits was feasible, which was done at the time with the help of 12 computers.

3.2.3 Finding the Appropriate Masks

Now we show how to compute the masks $\mathbf{a}, \mathbf{b}$. Our target is:

$$\mathbf{a}, \mathbf{b} \in \{0,1\}^N : \max_{\mathbf{u},\mathbf{v} \in \{0,1\}^N} \left\{ LP(\mathbf{u}, \mathbf{v}; \mathbf{k}_2^R) \right\} = LP(\mathbf{a}, \mathbf{b}; \mathbf{k}_2^R)$$

Direct computation of $\mathbf{a}, \mathbf{b}$ is not an option, for two reasons. First, it depends on the unknown key $\mathbf{k}_2^R$. Furthermore, it requires encrypting all N-bit binary vectors through rounds $[2, R]$ for every value of $\mathbf{a}, \mathbf{b}$, which is computationally unfeasible for typical block sizes (64-bits or 128-bits).

Researchers have dealt with the first problem by working with ELP values, which are key-independent, and make the assumption:

$$\forall \mathbf{a}, \mathbf{b} \in \{0,1\}^N, \mathbf{k} \in \{0,1\}^K : LP(\mathbf{a}, \mathbf{b}; \mathbf{k}) \approx ELP(\mathbf{a}, \mathbf{b})$$

By following this approach, our target becomes:

$$\mathbf{a}, \mathbf{b} \in \{0,1\}^N : \max_{\mathbf{u},\mathbf{v} \in \{0,1\}^N} \{ ELP(\mathbf{u}, \mathbf{v}) \} = ELP(\mathbf{a}, \mathbf{b})$$

Which solved the key-dependence problem. However, direct computation of $\max_{\mathbf{u},\mathbf{v} \in \{0,1\}^N} \{ELP(\mathbf{u}, \mathbf{v})\}$ is even more computationally expensive, since it requires encrypting every N-bit vector through rounds $2, \ldots, R$ using all possible values for $\mathbf{k}_2^R$. This problem was solved by using *linear characteristics* and *linear hulls*.

3.2.4 Linear Characteristics and Feasible Computation of ELP

Definition 3.2.1. *Let $1 \leq B \leq E \leq R$. A B through E round characteristic, denoted $\Omega[B, E]$ is an $E - B + 1$-tuple of masks:*

$$\Omega_B^E = \langle \mathbf{a}_B, \mathbf{a}_{B+1}, \ldots, \mathbf{a}_E, \mathbf{a}_{E+1} \rangle ; \mathbf{a}_i \in \{0,1\}^N ; B \leq i \leq E$$

Where we view $\mathbf{a}_i, \mathbf{a}_{i+1}$ as input and output masks for round i, respectively.

Definition 3.2.2. *Let $\Omega_B^E = \langle \mathbf{a}_B, \mathbf{a}_{B+1}, \ldots, \mathbf{a}_E, \mathbf{a}_{E+1} \rangle$ be a B through E round characteristic and $\mathbf{k}_2^R = \langle \mathbf{k}_B, \mathbf{k}_{B+1}, \ldots, \mathbf{k}_E \rangle$ is the key being used. We define*

the linear characteristic probability*, denoted* LCP *and the* expected linear characteristic probability*, denoted* $ELCP$ *as:*

$$LCP(\Omega_B^E; \mathbf{k}_B^E) \stackrel{def}{=} \prod_{i=B}^{E} LP(a_i, a_{i+1}; \mathbf{k}_i)$$

$$ELCP(\Omega_B^E) \stackrel{def}{=} \prod_{i=B}^{E} ELP(a_i, a_{i+1})$$

The reason behind the computational feasiblity of LCP and $ELCP$ is that usually block ciphers exhibit a simple round structure, which also turns out to be the case for SPNs. This simplicity allows for feasible computation of $ELP(a_i, a_{i+1})$. Once $ELP(a_i, a_{i+1})$ has been obtained for $B \leq i \leq E$, $ELCP(\Omega_B^E)$ can be obtained easily from the definition.

Now we explain the use of linear characteristics for solving the computational infeasiblity of $\max_{\mathbf{u},\mathbf{v}\in\{0,1\}^N} \{ELP(\mathbf{u}, \mathbf{v})\}$. First, we run a straightforward algorithm for finding the linear characteristic $M\Omega_2^R = \langle \mathbf{a}_2, \mathbf{a}_3, \ldots, \mathbf{a}_R, \mathbf{a}_{R+1} \rangle$ for which $ELCP(M\Omega_2^R)$ is maximal. Once this characteristic has been found, we take the masks for rounds $[2, R]$ in the algorithm above to be $\mathbf{a}_2, \mathbf{a}_{R+1}$. Then we find $\max_{\mathbf{u},\mathbf{v}\in\{0,1\}^N} \{ELP(\mathbf{u}, \mathbf{v})\}$ by approximating:

$$ELP(\mathbf{a}, \mathbf{b}) \approx ELCP(M\Omega)$$

This approximation has been widely used for the study of block ciphers. Later, Nyberg [13] showed that this approximation can cause an over estimation of the data complexity of the attack, by introducing the concept of linear hulls.

3.2.5 Linear Hulls

The following definition and theorem were set by Nyberg [13].

Definition 3.2.3. *Let* $\mathbf{a}, \mathbf{b} \in \{0,1\}^N$*. The* linear hull of masks $\mathbf{a}, \mathbf{b}$*, denoted* $LH(\mathbf{a}, \mathbf{b})$*, is the set of all* B *through* E *linear characteristics (where* B *through* E *are the rounds under consideration) having* $\mathbf{a}$ *as its first round input mask, and* $\mathbf{b}$ *as its last round output mask. That is, it is the set:*

$$LH(\mathbf{a}, \mathbf{b}) = \left\{ \Omega_B^E \,|\, \Omega_B^E = \langle \mathbf{a}, \mathbf{a}_{B+1}, \ldots, \mathbf{a}_E, \mathbf{b} \rangle \right\}$$

Theorem 3.2.4. *Let* $\mathbf{a}, \mathbf{b} \in \{0,1\}^N$. *Then:*

$$ELP(\mathbf{a}, \mathbf{b}) = \sum_{\Omega \in LH(\mathbf{a},\mathbf{b})} ELCP(\Omega)$$

Since the sum in the above theorem is taken above a relatively large set of linear characteristics, the expected linear probability is strictly larger than the best characteristics expected linear probability. Therefore, from the relation:

$$\mathrm{P} = \frac{c}{LP(\mathbf{a}, \mathbf{b}, \mathbf{k}_2^R)}$$

this results an over estimation of the data complexity required for a successful attack, the so-called *linear hull effect*.

Even though linear characteristics have the advantage of computational feasiblity, due to the linear hull effect, we should view the data complexity of the attack based on the value $ELCP$ of the best linear characteristic only as a first approximation of the actual data complexity. If it was infeasible to implement, one shouldn't stop here, but rather make use of the linear hull effect, which may yield feasible data complexities.

3.2.6 Provable Security

The primary source for computing the data complexity required for a successful linear crypanaylsis attack is the value:

$$\mathbf{a}, \mathbf{b} \in \{0,1\}^N : \max_{\mathbf{u},\mathbf{v} \in \{0,1\}^N} \{ELP(\mathbf{u}, \mathbf{v})\} = ELP(\mathbf{a}, \mathbf{b})$$

If this value was proven to be sufficiently small, *provable security* can be claimed. Many researches aim to upper-bound the above value (since direct computation is not an option), and many have succeeded.

Note that there is a difference between this use of the term *provable security* and that used in the context of assymetric cryptography. In the latter, a cipher is called *provably secure* if breaking it is at least as difficult as some other famous problem for which there is no efficient solution, such as the discrete logarithm and integer factorization. Formally, if there is a reduction from the problem of recovering the key given the ciphertext to some other NP-hard problem. Assymetric encryption has been a field of work for the authors.

3.3 Specialization Over Substitution-Permutation Networks

In this section, we shall take the concepts from the previous section and apply them to SPNs, which are a special form of Markov ciphers. This specialization is due to Keliher [4].

First, the proof that SPNs are a special case of Markov ciphers.

Theorem 3.3.1. *The SPN block cipher architecture described in chapter 1 is a Markov cipher.*

Proof Let $1 \leq r \leq R$, and let $\mathbf{x}_r \in \{0,1\}^N$ be round r input and $\Delta\mathbf{x}, \Delta\mathbf{y} \in \{0,1\}^N$ be fixed. We define the round r function as $\psi_r(\mathbf{x}_r; \mathbf{k}_r) = \zeta_r(\mathbf{x}_r \oplus \mathbf{k}_r)$. Now consider the relation from definition 3.1.3.:

$$P_{\mathbf{K}}\left[\psi(\mathbf{x};\mathbf{K}) \oplus \psi(\mathbf{x} \oplus \Delta\mathbf{x};\mathbf{K}) = \Delta\mathbf{y}\right] = P_{\mathbf{X},\mathbf{K}}\left[\psi(\mathbf{X};\mathbf{K}) \oplus \psi(\mathbf{X} \oplus \Delta\mathbf{x};\mathbf{K}) = \Delta\mathbf{y}\right]$$

Substituting ψ with ζ, we get the term:

$$P_K\left[\zeta_r(\mathbf{x} \oplus \mathbf{K}) \oplus \zeta(\mathbf{x} \oplus \Delta\mathbf{x} \oplus \mathbf{K}) = \Delta\mathbf{y}\right]$$

If we define the random variable $\mathbf{X} = \mathbf{x} \oplus \mathbf{K}$ and substitute into the above term, we get:

$$P_{\mathbf{X}}\left[\zeta_r(\mathbf{X}) \oplus \zeta_r(\mathbf{X} \oplus \mathbf{x}) = \Delta\mathbf{y}\right]$$

Which yields the same value as the previous term. And the result follows.

Theorem 3.3.2. *Let* $\mathbf{a}, \mathbf{b} \in \{0,1\}^N$. *Then the value* $LP(\mathbf{a}, \mathbf{b}; \mathbf{k})$ *is independent of* $\mathbf{k}$ *And therefore, for every* $\mathbf{k} \in \{0,1\}^N$:

$$LP(\mathbf{a}, \mathbf{b}; \mathbf{k}) = ELP(\mathbf{a}, \mathbf{b})$$

Proof Let $\mathbf{X} \in \{0,1\}^N$ and $\mathbf{X}^* = \mathbf{X} \oplus \mathbf{k}$ be random variables. It suffices to prove that $LP(\mathbf{a}, \mathbf{b}; \mathbf{k}) = LP(\mathbf{a}, \mathbf{b}; \mathbf{0})$ for all $\mathbf{k}$. Note that:

$$\begin{aligned} LP(\mathbf{a}, \mathbf{b}; \mathbf{k}) &= \left(2P_{\mathbf{X}}\left[\mathbf{a} \cdot \mathbf{X} = \mathbf{b} \cdot \zeta(\mathbf{X} \oplus \mathbf{k})\right] - 1\right)^2 \\ &= \left(2P_{X^*}\left[\mathbf{a} \cdot (\mathbf{X}^* \oplus \mathbf{k}) = \mathbf{b} \cdot \zeta(\mathbf{X}^*)\right] - 1\right)^2 \\ &= \left(2P_{X^*}\left[(\mathbf{a} \cdot \mathbf{X}^*) \oplus (\mathbf{a} \cdot \mathbf{k}) = \mathbf{b} \cdot \zeta(\mathbf{X}^*)\right] - 1\right)^2 \\ &= \left(2P_{X^*}\left[(\mathbf{a} \cdot \mathbf{X}^*) \oplus (\mathbf{a} \cdot \mathbf{k}) = \mathbf{b} \cdot \zeta(\mathbf{X}^*)\right] - 1\right)^2 \end{aligned}$$

If $\mathbf{a}\cdot\mathbf{k}=0$ then the term in the final relation is equal to $LP(\mathbf{a},\mathbf{b};\mathbf{0})$. Otherwise, if $\mathbf{a}\cdot\mathbf{k}=1$, then:

$$P_{X^*}\left[(\mathbf{a}\cdot\mathbf{X}^*)\oplus(\mathbf{a}\cdot\mathbf{k})=\mathbf{b}\cdot\zeta(\mathbf{X}^*)\right]=1-P_{\mathbf{X}^*}\left[\mathbf{a}\cdot\mathbf{X}^*=\mathbf{b}\cdot\zeta(\mathbf{X}^*)\right]$$

And after substituting, we again return to $LP(\mathbf{a},\mathbf{b};\mathbf{k})$

Theorem 3.3.3. *Let Ω_B^E be a linear characteristic and $\mathbf{k}_B^E$ be the vector of subkeys used for the rounds under consideration. Then:*

$$LCP(\Omega_B^E;\mathbf{k}_B^E)=ELCP(\Omega_B^E)$$

Proof This follows easily from the definition of $ELCP$ and theorem 3.3.2.

Lemma 3.3.4. **[3]** *If $\mathbf{a}\in\{0,1\}^N$ is a mask applied to the input of a linear transformation* T*, Then there is a unique corresponding output mask $\mathbf{b}$ such that for every $\mathbf{x}\in\{0,1\}^N$:*

$$\mathbf{a}\cdot\mathbf{x}=\mathbf{b}\cdot\mathrm{T}(\mathbf{x})$$

And the relation between the input and output masks is given by $\mathbf{a}=\mathrm{T}'\mathbf{b}$.

From the above lemma, if $\mathbf{a}_r,\mathbf{a}_{r+1}\in\{0,1\}^N$ are input and output masks respectively for round r, then the input and output for round r excluding the key mixing phase and the linear transformation phase are $\mathbf{a}_r$ and $\mathbf{b}_r=\mathrm{T}'\mathbf{a}_{r+1}$. Using $\mathbf{a}_r$ and $\mathbf{b}_r$, we can get a pair of input-output masks for every S-box in round r.

Let's denote the S-boxes in round r $S_r^1,S_r^2,\ldots$ and denote the input and output masks for S-box i in round r with $\mathbf{a}_r^i,\mathbf{b}_r^i$, respectively. The following relation can be derived directly from Matsui's piling up lemma[6] and lemma 3.3.2.:

$$ELP_r(\mathbf{a}_r,\mathbf{a}_{r+1})=\prod_{i=1}^{M}LP_{S_i^r}(\mathbf{a}_i,\mathbf{b}_i)$$

In the product above there are no subkey parameters, since the substitution phase doesn't involve the subkey mixing or any other operation concerning the subkeys.

4 Practical Implementation

In this chapter, we shall introduce the general concepts on which we based our practical implementation of the SPN software and the server application. Finally, we discuss the practical implementation of linear cryptanalysis and our results.

4.1 SPN Software Implementation

4.1.1 Our SPN Specifications

The SPN we are using in this project has a 16-bit block size, with four encryption rounds. In order to simplify our software code and computations, we did not emit the substitution and permutation phases in the last round, even though they add no additional security.

The key-mixing phase in each round constitute of XORing the current plaintext with a 16-bit subkey. After the key-mixing phase, come four 4×4 s-boxes, which were generated randomly using Wolfram Mathematica 10.2 software.

We used the linear transformation proposed by Kam and Davida [9]. In this transformation, the input bit j for s-box i in round r comes from the output bit i from s-box j in round $r - 1$. A nice feature of this transformation is that its inverse is the same transformation, which allows for simpler code.

These SPN specifications are not efficient, for several reasons. It has a small block size, no key schedule algorithm and the key is large relative to the block size. We only use it to illustrate how linear cryptanalysis can be applied to SPNs.

4.1.2 Software

The SPN software was written in JAVA, using NetBeans IDE 8.0.1 software. First, we created a user-defined class which represents the data block, in which the 16-bit data blocks are stored in a 32-bit integer, for easier bit manipulation. Plus, a variety of methods that allow a flexible bit handling.

The source code for the data block class is available at:

`https://github.com/Jay-611/Linear-Cryptanalysis-of-SPNs/blob/master/SPNImplementation/DataBlock16Bit.java`

.

After that, we created an SPN implementer class, in which the encryption and decryption methods were defined.

The source code for the SPN implementer class is available at:

`https://github.com/Jay-611/Linear-Cryptanalysis-of-SPNs/blob/master/SPNImplementation/SPNImplementer16Bit.java`

4.2 Network Application

In order to illustrate how a linear cryptanalysis implementation works, we created a JAVA project using NetBeans IDE 8.0.1 software, which includes a multi-threaded client/server model, an SPN implementation and a cryptanalyst implementation. We will describe each of these parts in detail.

4.2.1 The Multi-Threaded Client/Server Model

We had created this model using two classes from the JAVA.net package, which are the Socket and the ServerSocket classes.

Sockets

A socket is the one end-point of a two-way communication link between two programs running over the network. The server implements a dedicated logic,

called a service. Clients connect to the server to get served. For example, to obtain some data or to ask for the computation of some data. Different client/server applications implement different kinds of services.

Ports

To distinguish different services, a numbering convention was proposed. This convention uses integer numbers, called port numbers, to denote services. A server implementing a service assigns a specific port number to the entry point of the service. There are no specific physical entry points for services in a computer. The port numbers for services are stored in configuration files and are used by the computer software to create network connections.

Socket Mechanism

A socket is a complex data structure that contains an internet address and a port number. A socket, however, is referenced by its descriptor, just like a file which is referenced by a file descriptor. That is why, sockets are accessed via an application programming interface (API) similar to the file input/output API. This makes programming network applications very simple. The two-way communication link between the programs running on different computers is done by reading from and writing to the sockets created on these computers. The data read from a socket is the data wrote into the other socket of the link. And reciprocally, the data wrote into a socket is the data read from the other socket of the link. These two sockets are created and linked during the connection creation phase. The link between two sockets is like a pipe that is implemented using a stack of protocols. This linking of the sockets involves that a socket internally has a much more complex data structure, or more precisely, a collaboration of data structures.

In order to run our model, we first need to create a network connection.

Creating a Network Connection

A network connection is initiated by a client program when it creates a socket to communicate with the server. To create the socket in JAVA, the client calls

the Socket constructor and passes the server address and the specific server port number to it. At this stage the server must be started on the machine having the specified address and listening for connections on its specific port number.

The server uses a specific port dedicated only to listening for connection requests from clients. It cannot use this specific port for data communication with clients because the server must be able to accept client connections at any instant. So, its specific port is dedicated to only listen for new connection requests. The server side socket associated with a specific port is called a server socket. When a connection request arrives to this socket from the client side, this client and the server establish a connection. This connection is established as follows:

1. When the server receives a connection request on its specific server port, it creates a new socket for it and binds a port number to it.
2. It sends the new port number to the client to inform it that the connection is established.
3. The server goes on now by listening on two ports:
    - it waits for new incoming connection requests on its specific port
    - it reads and writes messages on the established connection (with the new port assigned to it) with the accepted client.

The server communicates with the client by reading from and writing to the new port. If other connection requests arrive, the server accepts them in a similar way creating a new port for each new connection. Thus, at any instant, the server must be able to communicate simultaneously with many clients and to wait on the same time for incoming requests on its specific server port. The communication with each client is done via the sockets created for each communication.

The JAVA.net Package

The JAVA.net package in the JAVA development environment provides the Socket class, which implements the client side, and the ServerSocket class, which implements the server side. Both the client and the server must agree on the lan-

guage of the information transferred back and forth through the socket. Meaning that they must agree on a protocol. There are two communication protocols:

- Stream communication protocol
- Datagram communication protocol

The TCP Protocol

The stream communication protocol is known as TCP (Transfer Control Protocol). TCP is a connection-oriented protocol. In order to communicate over the TCP protocol, a connection must first be established between two sockets. While one of the sockets (the server socket) listens for a connection request, the other (the client socket) asks for a connection. Once the two sockets are connected, they can be used to transmit and/or to receive data. When we say "two sockets are connected" we mean the fact that the server accepted a connection. As it was explained above the server creates a new local socket for the new connection. The process of the new local socket creation, however, is transparent for the client.

The UDP Protocol

The datagram communication protocol, known as UDP (user datagram protocol), is a connectionless protocol. No connection is established before sending the data. The data are sent in a packet called datagram. The datagram is sent like a request for establishing a connection. However, the datagram contains not only the addresses, it contains the user data also. Once it arrives to the destination the user data are read by the remote application and no connection is established. This protocol requires that each time a datagram is sent, the local socket and the remote socket addresses must also be sent in the datagram. These addresses are sent in each datagram.

The JAVA.net package provides the DatagramSocket class for programming datagram communications. UDP is an unreliable protocol. There is no guarantee that the datagrams will be delivered in a good order to the destination socket. For example, a long text, split in several pages and sent one page per datagram, can be received in a different page order. On the other side TCP is

a reliable protocol. TCP guarantees that the pages will be received in the order in which they are sent.

When programming TCP and UDP based applications in JAVA, different types of sockets are used. These sockets are implemented in different classes. The ServerSocket and Socket classes implement TCP based sockets and the DatagramSocket class implements UDP based sockets as follows:

- Stream socket to listen for client requests (TCP): The ServerSocket class.
- Stream Socket (TCP): The Socket class.
- Datagram socket (UDP): The DatagramSocket class.

In our project we created a TCP network without any UDP based sockets. We also used the DataInputStream class and the PrintStream class for the I/O operations.

We implemented the client using two threads – the first one is used to interact with the server and the other is used to interact with the standard input. Two threads are needed because the client must communicate with the server and, simultaneously, it must be ready to read messages from the standard input to be sent to the server. We implemented the server using threads also. It uses a separate thread for each connection. It spawns a new client thread every time a new connection from a client is accepted. Using threads could generate some synchronization issues that might crash the application, in order to avoid these issues, the threads must be synchronized. We did that using synchronized(this) statements.

The source codes for the client/server model are available at:

`https://github.com/Jay-611/Linear-Cryptanalysis-of-SPNs/tree/master/Client-ServerModel`

4.3 The Attack

For attacking our SPN implementation, we will apply a linear cryptanalysis attack on the first round. This attack will allow us to recover a subset of the bits of the full key bits.

In order to attack the first round subkey, we must find a round 2 input mask and a round 4 output mask. To get hold of these masks, we found a 2 through 4 round characteristic for which the $ELCP$ value is sufficiently large. Here's how we found it, in a bottom-up fashion.

First, for each round 2 S-Box, we found using exhaustive search input and output masks that hold with the best probability, which is feasible since it contains only $2^4 \times 2^4 = 2^8 = 256$ different possibilities. We have excluded the case were the input and output masks are both 0. The masks are the following:

$$\mathbf{a}_2^1 = \langle 1,1,1,0\rangle, \mathbf{a}_2^2 = \langle 1,0,0,0\rangle, \mathbf{a}_2^3 = \langle 0,0,0,1\rangle, \mathbf{a}_2^4 = \langle 0,1,0,1\rangle$$

$$\mathbf{b}_2^1 = \langle 0,1,1,0\rangle, \mathbf{b}_2^2 = \langle 1,0,1,1\rangle, \mathbf{b}_2^3 = \langle 1,0,1,1\rangle, \mathbf{b}_2^4 = \langle 1,0,1,1\rangle$$

And their corresponding LP values are:

$$LP\left(\mathbf{a}_2^1, \mathbf{b}_2^1\right) = 1.0000, LP\left(\mathbf{a}_2^2, \mathbf{b}_2^2\right) = 0.5625$$

$$LP\left(\mathbf{a}_2^3, \mathbf{b}_2^3\right) = 0.5625, LP\left(\mathbf{a}_2^4, \mathbf{b}_2^4\right) = 0.5625$$

Combining the input masks for the S-Boxes we get the input mask for round 2, which is:

$$\mathbf{a}_2 = \langle 1,1,1,0,1,0,0,0,0,0,0,1,0,1,0,1\rangle$$

And combining the output masks for the S-Boxes we get the output mask for the substitution phase of round 2, which is:

$$\mathbf{b}_2 = \langle 0,1,1,0,1,0,1,1,1,0,1,1,1,0,1,1\rangle$$

Now that the output mask for the substitution phase is known, the output mask for the whole round can be determined by applying the linear transformation to the substitution phase output mask. We get:

$$\mathbf{a}_3 = \langle 0,1,1,1,1,0,0,0,1,1,1,1,0,1,1,1\rangle$$

And we get the first round masks, which have the following ELP value:

$$ELP(\mathbf{a}_2, \mathbf{a}_3) = \prod_{i=1}^{r} LP\left(\mathbf{a}_2^i, \mathbf{b}_2^i\right)$$

$$= 1.0000 \times 0.5625 \times 0.5625 \times 0.5625 = 0.177978515625$$

For the third round masks, we did exactly the same thing, except that this time the exhaustive search was on the output S-Box masks only, since the input masks are already determined. The masks were the following:

$$\mathbf{a}_3^1 = \langle 0,1,1,1\rangle, \mathbf{a}_3^2 = \langle 1,0,0,0\rangle, \mathbf{a}_3^3 = \langle 1,1,1,1\rangle, \mathbf{a}_3^4 = \langle 0,1,1,1\rangle$$

$$\mathbf{b}_3^1 = \langle 0,0,0,1\rangle, \mathbf{b}_3^2 = \langle 0,0,1,0\rangle, \mathbf{b}_3^3 = \langle 0,1,0,1\rangle, \mathbf{b}_3^4 = \langle 1,0,1,0\rangle$$

And these masks have the following LP values:

$$LP\left(\mathbf{a}_3^1, \mathbf{b}_3^1\right) = 0.2500, LP\left(\mathbf{a}_3^2, \mathbf{b}_3^2\right) = 0.2500$$

$$LP\left(\mathbf{a}_3^3, \mathbf{b}_3^3\right) = 0.2500, LP\left(\mathbf{a}_3^4, \mathbf{b}_3^4\right) = 0.5625$$

And in a similar way to that of round 2, we get round 3 masks, which are:

$$\mathbf{a}_3 = \langle 0,1,1,1,1,0,0,0,1,1,1,1,0,1,1,1\rangle$$
$$\mathbf{a}_4 = \langle 0,0,0,1,0,0,1,0,0,1,0,1,1,0,1,0\rangle$$

And their corresponding ELP value is:

$$ELP(\mathbf{a}_3, \mathbf{a}_4) = \prod_{i=1}^{r} LP\left(\mathbf{a}_3^i, \mathbf{b}_3^i\right)$$

$$= 0.2500 \times 0.2500 \times 0.2500 \times 0.5625 = 0.0087890625$$

Finally, for round 4, the masks:

$$\mathbf{a}_4^1 = \langle 0,0,0,1\rangle, \mathbf{a}_4^2 = \langle 0,0,1,0\rangle, \mathbf{a}_4^3 = \langle 0,1,0,1\rangle, \mathbf{a}_4^4 = \langle 1,0,1,0\rangle$$

$$\mathbf{b}_4^1 = \langle 0,1,0,0\rangle, \mathbf{b}_4^2 = \langle 1,0,1,0\rangle, \mathbf{b}_4^3 = \langle 0,0,1,1\rangle, \mathbf{b}_4^4 = \langle 0,1,0,1\rangle$$

The corresponding LP values:

$$LP\left(\mathbf{a}_4^1, \mathbf{b}_4^1\right) = 0.2500, LP\left(\mathbf{a}_4^2, \mathbf{b}_4^2\right) = 0.5625$$

$$LP\left(\mathbf{a}_4^3, \mathbf{b}_4^3\right) = 0.2500, LP\left(\mathbf{a}_4^4, \mathbf{b}_4^4\right) = 0.2500$$

The complete masks:

$$\mathbf{a}_4 = \langle 0,0,0,1,0,0,1,0,0,1,0,1,1,0,1,0\rangle$$

$$\mathbf{a}_5 = \langle 0,1,0,0,1,0,0,1,0,1,1,0,0,0,1,1\rangle$$

And the corresponding ELP value:

$$ELP(\mathbf{a}_4, \mathbf{a}_5) = \prod_{i=1}^{r} LP\left(\mathbf{a}_4^i, \mathbf{b}_4^i\right)$$
$$= 0.2500 \times 0.5625 \times 0.2500 \times 0.2500 = 0.0087890625$$

Combining out results so far, we have now a 2 through 4 round characteristic $\Omega_2^4 = \langle \mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_4, \mathbf{a}_5 \rangle$, which is able to attack 7 effective round 1 effective subkey bits. And it which has the following $ELCP$ value:

Round 2	S-Box Input Mask	$\mathbf{a}_2^1 = \langle 1,1,1,0 \rangle$	$\mathbf{a}_2^2 = \langle 1,0,0,0 \rangle$	$\mathbf{a}_2^3 = \langle 0,0,0,1 \rangle$	$\mathbf{a}_2^4 = \langle 0,1,0,1 \rangle$
	S-Box Output Mask	$\mathbf{b}_2^1 = \langle 0,1,1,0 \rangle$	$\mathbf{b}_2^2 = \langle 1,0,1,1 \rangle$	$\mathbf{b}_2^3 = \langle 1,0,1,1 \rangle$	$\mathbf{b}_2^4 = \langle 1,0,1,1 \rangle$
	LP Value	1.0000	0.5625	0.5625	0.5625
	Complete Input Mask	$\mathbf{a}_2 = \langle 1,1,1,0,1,0,0,0,0,0,0,1,0,1,0,1 \rangle$			
	Substitution Output Mask	$\mathbf{b}_2 = \langle 0,1,1,0,1,0,1,1,1,0,1,1,1,0,1,1 \rangle$			
	Complete Output Mask	$\mathbf{a}_3 = \langle 0,1,1,1,1,0,0,0,1,1,1,1,0,1,1,1 \rangle$			
	ELP Value	0.177978515625			
Round 3	S-Box Input Mask	$\mathbf{a}_3^1 = \langle 0,1,1,1 \rangle$	$\mathbf{a}_3^2 = \langle 1,0,0,0 \rangle$	$\mathbf{a}_3^3 = \langle 1,1,1,1 \rangle$	$\mathbf{a}_3^4 = \langle 0,1,1,1 \rangle$
	S-Box Output Mask	$\mathbf{b}_3^1 = \langle 0,0,0,1 \rangle$	$\mathbf{b}_3^2 = \langle 0,0,1,0 \rangle$	$\mathbf{b}_3^3 = \langle 0,1,0,1 \rangle$	$\mathbf{b}_3^4 = \langle 1,0,1,0 \rangle$
	LP Value	0.2500	0.2500	0.2500	0.5625
	Complete Input Mask	$\mathbf{a}_3 = \langle 0,1,1,1,1,0,0,0,1,1,1,1,0,1,1,1 \rangle$			
	Substitution Output Mask	$\mathbf{b}_3 = \langle 0,0,0,1,0,0,1,0,0,1,0,1,1,0,1,0 \rangle$			
	Complete Output Mask	$\mathbf{a}_4 = \langle 0,0,0,1,0,0,1,0,0,1,0,1,1,0,1,0 \rangle$			
	ELP Value	0.0087890625			
Round 4	S-Box Input Mask	$\mathbf{a}_4^1 = \langle 0,0,0,1 \rangle$	$\mathbf{a}_4^2 = \langle 0,0,1,0 \rangle$	$\mathbf{a}_4^3 = \langle 0,1,0,1 \rangle$	$\mathbf{a}_4^4 = \langle 1,0,1,0 \rangle$
	S-Box Output Mask	$\mathbf{b}_4^1 = \langle 0,1,0,0 \rangle$	$\mathbf{b}_4^2 = \langle 1,0,1,0 \rangle$	$\mathbf{b}_4^3 = \langle 0,0,1,1 \rangle$	$\mathbf{b}_4^4 = \langle 0,1,0,1 \rangle$
	LP Value	0.2500	0.5625	0.2500	0.2500
	Complete Input Mask	$\mathbf{a}_4 = \langle 0,0,0,1,0,0,1,0,0,1,0,1,1,0,1,0 \rangle$			
	Substitution Output Mask	$\mathbf{b}_4 = \langle 0,1,0,0,1,0,1,0,0,0,1,1,0,1,0,1 \rangle$			
	Complete Output Mask	$\mathbf{a}_5 = \langle 0,1,0,0,1,0,0,1,0,1,1,0,0,0,1,1 \rangle$			
	ELP Value	0.0087890625			
ELCP Value		0.00001374841667711735			
Expected Data Complexity		1.16×10^6			

Table 4.1: Our first linear characteristic

$$ECLP\left(\Omega_2^4\right) = \prod_{i=1}^{4} ELP\left(\mathbf{a}_i, \mathbf{a}_{i+1}\right)$$
$$= 0.177978515625 \times 0.0087890625 \times 0.0087890625$$
$$= 0.000013748416677117348$$

This means that, for applying algorithm 2 with a 78.5% chance of success to recover the first subkey, we need:

$$\mathrm{P} = \frac{16}{ECLP\left(\Omega_2^4\right)} = \frac{16}{0.000013748416677117348} = 1.16 \times 10^6$$

known plaintext-ciphertext pairs. Let's sum up our results in Table 4.1.

The problem is that the data complexity associated with this characteristic is 1.16×10^6, which is impossible to get hold of since that there are only $2^{16} = 65536$ possible plaintexts. We intentionally wrote this case to show that there are several criteria for selecting the appropriate characteristic.

Round 2	S-Box Input Mask	$\mathbf{a}_2^1 = \langle 1,1,1,0\rangle$	$\mathbf{a}_2^2 = \langle 0,0,0,0\rangle$	$\mathbf{a}_2^3 = \langle 0,0,0,0\rangle$	$\mathbf{a}_2^4 = \langle 0,1,0,1\rangle$
	S-Box Output Mask	$\mathbf{b}_2^1 = \langle 0,1,1,0\rangle$	$\mathbf{b}_2^2 = \langle 0,0,0,0\rangle$	$\mathbf{b}_2^3 = \langle 0,0,0,0\rangle$	$\mathbf{b}_2^4 = \langle 1,0,1,1\rangle$
	LP Value	1.0000	1.0000	1.0000	0.5625
	Complete Input Mask	$\mathbf{a}_2 = \langle 1,1,1,0,0,0,0,0,0,0,0,0,0,1,0,1\rangle$			
	Substitution Output Mask	$\mathbf{b}_2 = \langle 0,1,1,0,0,0,0,0,0,0,0,0,0,1,1,1\rangle$			
	Complete Output Mask	$\mathbf{a}_3 = \langle 0,0,0,1,1,0,0,0,1,0,0,1,0,0,0,1\rangle$			
	ELP Value	0.5625			
Round 3	S-Box Input Mask	$\mathbf{a}_3^1 = \langle 0,0,0,1\rangle$	$\mathbf{a}_3^2 = \langle 1,0,0,0\rangle$	$\mathbf{a}_3^3 = \langle 1,0,0,1\rangle$	$\mathbf{a}_3^4 = \langle 0,0,0,1\rangle$
	S-Box Output Mask	$\mathbf{b}_3^1 = \langle 0,0,1,0\rangle$	$\mathbf{b}_3^2 = \langle 0,0,1,0\rangle$	$\mathbf{b}_3^3 = \langle 0,0,0,1\rangle$	$\mathbf{b}_3^4 = \langle 0,0,0,1\rangle$
	LP Value	0.2500	0.2500	0.5625	0.2500
	Complete Input Mask	$\mathbf{a}_3 = \langle 0,0,0,1,1,0,0,0,1,0,0,1,0,0,0,1\rangle$			
	Substitution Output Mask	$\mathbf{b}_3 = \langle 0,0,1,0,0,0,1,0,0,0,0,1,0,0,0,1\rangle$			
	Complete Output Mask	$\mathbf{a}_4 = \langle 0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,1\rangle$			
	ELP Value	0.0087890625			
Round 4	S-Box Input Mask	$\mathbf{a}_4^1 = \langle 0,0,0,0\rangle$	$\mathbf{a}_4^2 = \langle 0,0,0,0\rangle$	$\mathbf{a}_4^3 = \langle 1,1,0,0\rangle$	$\mathbf{a}_4^4 = \langle 0,0,1,1\rangle$
	S-Box Output Mask	$\mathbf{b}_4^1 = \langle 0,0,0,0\rangle$	$\mathbf{b}_4^2 = \langle 0,0,0,0\rangle$	$\mathbf{b}_4^3 = \langle 0,1,1,1\rangle$	$\mathbf{b}_4^4 = \langle 0,0,1,0\rangle$
	LP Value	1.0000	1.0000	0.2500	0.5625
	Complete Input Mask	$\mathbf{a}_4 = \langle 0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,1\rangle$			
	Substitution Output Mask	$\mathbf{b}_4 = \langle 0,0,0,0,0,0,0,0,0,1,1,1,0,0,1,0\rangle$			
	Complete Output Mask	$\mathbf{a}_5 = \langle 0,0,0,0,0,0,1,0,0,0,1,1,0,0,1,0\rangle$			
	ELP Value	0.140625			
ELCP Value		0.0006952285766601562			
Expected Data Complexity		23014			

Table 4.2: The modified characteristic

To solve this problem, we used another characteristic that attacks fewer effective subkey bits, but with a higher ELP value so that the associated data complexity is realizable. We derived the new characteristic in exactly the same way as the previous one, except that we started with different round 1 masks. Applying the same techniques, we get thr characteristic described in table 4.2.

Applying the linear cryptanalysis using this characteristic is slower than that of the first characteristic, since attacks 2 bits fewer effective subkey bits than the first one, that is 4 times slower. However, it has a reasonable data complexity.

Now we proceed to extract the first round subkey. We have generated 25000

plaintext-ciphertext pairs, and we assume that they are known to everyone on the network. Although this is not a realistic situation, but we will work in this environment which represent the worst case scenario.

Suppose that we have at disposal those 25000 known plaintext-ciphertext pairs $\langle p_i, c_i \rangle ; 1 \leq i \leq 25000$ plus an encrypted target message c_t that we wish to break. If this message (which we represented as a string) was larger than 16 bits, we break into 16 bit parts and apply the next procedure to each part.

First, we loop through all the possible values for the round 1 effective key bits, there are $2^5 = 32$ total guesses. For each guess $\mathbf{g}_j ; 1 \leq j \leq 32$, we loop through all the known plaintext-ciphertext pairs and increment a counter $C_{\mathbf{g}_i}$ every time the following relation holds true:

$$\mathbf{a}_2 \cdot \psi(p_i; \mathbf{g}_j) = \mathbf{a}_5 \cdot c_i$$

Since there are 32 possible guesses, and 25000 known plaintext-ciphertext pairs, the above condition is tested $32 \times 25000 = 800000$ times. After we get hold of all the counters $C_{\mathbf{g}_i} ; 1 \leq i \leq 32$, we choose the guess that maximizes $(2C_{\mathbf{g}_i} - 12500)^2$ to be the correct guess.

The source code or the above operation is available at:

`https://github.com/Jay-611/Linear-Cryptanalysis-of-SPNs/blob/master/CryptanalysisImplementation/CryptanalysisImplementation.java`

5 Conclusions

5.1 Summary

Theoretically, we have presented a mathematical study of block ciphers in general and substitution-permutation networks. Besides a detailed mathematical study of linear cryptanalysis on substitution-permutation networks. This study is a first of a kind in the Syrian Arab Republic.

Practically, we have successfully implemented an SPN encryption scheme plus client/server model.

And we have successfully implemented linear cryptanalysis on the SPN encryption scheme that we built. Algorithm 2 succeeded in extracting 5 round 1 subkey bits. More subkey bits can be extracted by:

1. Using multiple linear characteristics: Applying Algorithm 2 using another characteristic that attacks another subset of the target subkey may extract more subkey bits.
2. Attacking a different round: We have used Algorithm to attack the first round subkey. We can use it with another suitable characteristic to attack the final round subkey, which the method Matsui used to break DES.

There are several reasons why a full break was not possible on our SPN scheme.

1. The small block size: Our SPN implementation uses a 16-bit block size, which is small even for old block ciphers. This small block size limits the abilities of the linear characteristics. Plus, it gives a relatively small upper on the number of known plaintext-ciphertext pairs.

2. No use of a key schedule algorithm: A key schedule is a potential weakness for every block cipher architecture, because it means that the subkeys are not independent, and that a relation that relates the subkey bits to the original key bits can be derived. We emitted the key schedule so that we can assume that our SPN is a Markov cipher.

3. The key is very large relative to the block size: Since there is no key schedule, the key is assumed to the combination of the four 16-bit subkeys. Therefore the key is four times larger than the block size, which limits the ability of Algorithm 2 to derive a large part of the full key to the point where exhaustive search becomes feasible.

Based on our procedure for finding the appropriate linear characteristics, we can derive find several ways for establishing security for SPNs.

1. Increasing the number of round: Since the expected linear characteristic probability $ELCP$ is taken as a product of different ELP values over all the rounds, and since ELP values are bounded in the interval $[0, 1]$, increasing the number of rounds results in smaller values of $ELCP$, and therefore higher data complexity.

2. Carefully selecting the S-Boxes: The ELP value of a single round masks is taken as a product of the LP values for all that rounds S-Boxes. Therefore, we have to make sure that there is no input and output masks for an S-Box so that the associated LP value is relatively high.

3. Carefully designing the key schedule: Matsui had the advantage that the 26 effective key bits that he extracted corresponded to 26 original key bits, which is considered a weakness in the DES key schedule. If the key schedule was designed so that no relation between the subkey bits and the original key, the attacker ends in a situation similar to the one we encountered, where he has no option but to extract all the subkeys.

5.2 Future Work

This project can be considered as a base for a variety of possible future work.

1. Using multiple linear approximations, which may get us one step closer to the full break.
2. Applying the methods above to Feistel networks and other block ciphers architectures.
3. Reducing the data complexity by applying the linear hull effect.

We hope that in the future, this project will be thought of as a base for the beginning of linear cryptanalysis teaching in our Syrian universities. And we will build a string foundation for this field in Syria, based in the National Center for the Distinguished.

List of Figures

List of Tables

Bibliography

[1] Menezes, Alfred J., Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[2] Paar, Christof, and Jan Pelzl. *Understanding cryptography: a textbook for students and practitioners*. Springer Science & Business Media, 2009.

[3] Daemen, Joan, and Vincent Rijmen. *"AES proposal: Rijndael."* (1999).

[4] Keliher, Liam. *Linear cryptanalysis of substitution-permutation networks*. Diss. Queen's University, 2003.

[5] Meier, Willi, and Othmar Staffelbach. "Nonlinearity criteria for cryptographic functions." *Advances in Cryptology—EUROCRYPT'89*. Springer Berlin Heidelberg, 1989.

[6] Matsui, Mitsuru. "Linear cryptanalysis method for DES cipher." *Advances in Cryptology—EUROCRYPT'93*. Springer Berlin Heidelberg, 1993.

[7] Biham, Eli, and Adi Shamir. "Differential cryptanalysis of DES-like cryptosystems." *Journal of CRYPTOLOGY* 4.1 (1991): 3-72.

[8] Vaudenay, Serge. "On the security of CS-cipher." *Fast Software Encryption*. Springer Berlin Heidelberg, 1999.

[9] Kam, John B., and George I. Davida. "Structured design of substitution-permutation encryption networks." *Computers, IEEE Transactions on* 100.10 (1979): 747-753.

[10] Heys, Howard M. "A tutorial on linear and differential cryptanalysis." *Cryptologia* 26.3 (2002): 189-221.

[11] Junod, Pascal. *Linear cryptanalysis of DES*. No. LASEC-STUDENT-2007-001. 2000.

[12] Junod, Pascal, and Serge Vaudenay. "Optimal key ranking procedures in a statistical cryptanalysis." *Fast Software Encryption*. Springer Berlin Heidelberg, 2003.

[13] Nyberg, Kaisa. "Linear approximation of block ciphers." *Advances in Cryptology—EUROCRYPT'94*. Springer Berlin Heidelberg, 1994.