# Differential Cryptanalysis of Block Ciphers

Matthew Laten

Third Year Maths Project

Supervisor: Dr Christine Swart

“Two can keep a secret if one is dead...” - Anon

# Abstract

In this paper we discuss the mathematics behind attacking a Block Cipher by means of Differential Cryptanalysis. To this end, we lay down some background knowledge in the fields of Probability Theory, Boolean Algebra and Block Ciphers themselves first, along with examples and the necessary mathematical theorems. We then move on to explaining what Differential Cryptanalysis is, how it works, and how we can use it against Substitution-Permutation Networks. We conclude with a walkthrough of the process by means of an example.

# Contents

# Chapter 1

# Introduction

Imagine a world without encryption. Anyone with access to a computer could steal your money, impersonate you, or learn your secrets. Privacy would be dead. The inability to secure information might even lead to police states, where governments will have to resort to desperate measures to protect their secrets. Fortunately, we do not live in such a world. We live in a world where encryption is a very real part of our daily lives. Whether banking online, shopping for new items or chatting to your friends on your favourite social network, you are indirectly using various forms of encryption to keep your messages safe, and indicate to computer servers that you are who you say you are.

So if our data is all secure and encrypted, why should we worry more about the subject of Cryptography? In short, our data isn't secure. Attackers find more and more ingenious ways of breaking implementations and protocols, even when the encryption method is said to be secure. The premier algorithm for encrypting electronic data from 1979 onwards, known as the Data Encryption Standard (DES), was publicly broken in 1997 (Curtin & Dolske, 1998). Today's encryption standards are tomorrow's broken algorithms. So we, as cryptographers, have the responsibility to make encryption more secure; even in some cases, to improve the manner in which these algorithms are implemented! How can we do this? By donning a black hat and thinking like an attacker. In this paper, we will be looking at an attack on block ciphers, specifically known as differential cryptanalysis. But first, we should probably define some unfamiliar terminology.

## 1.1 Terminology

**Remark:** Note, this paper assumes that you have an adequate knowledge of 3rd year university level Mathematics, but a knowledge of Cryptography or Probability theory is not required, as all concepts necessary for understanding this paper will be explained.

As you might already know, **Cryptography** is the discipline concerned with keeping data secret, or more formally, it is the study and practice of techniques and algorithms for securing the transference of data in the presence of third parties, known as attackers or adversaries. **Cryptanalysis** however, deals with the breaking of these techniques and retrieval of the secret data.

In cryptography, **encryption** refers to the process of converting plaintext, or data that is easily understandable, into ciphertext, that is unintelligible data. The reverse process of converting the ciphertext back into plaintext is known as **decryption**. In most cases, this encryption or decryption occurs with the aid of a **key**, a parameter that determines the functional output of the cryptographic algorithm.

Furthermore, there are two main types which come up when discussing key-based cryptography, namely Symmetric-key, and Asymmetric-key or Public-key cryptography. In the case of **Symmetric-key**, the same key is used for encryption and decryption, while with **Asymmetric-key**, a key is made available to the public for encryption, and only those possessing the secret or private key will be able to decrypt messages encrypted with the matching public key.

A **block cipher** is a type of Symmetric-key encryption cipher that operates on a fixed-length "block" of data. This is in contrast to a **stream cipher**, which operates on a potentially infinite stream of data. For the purposes of this paper, we will define a block cipher in more depth later, but at the moment we have enough of a vocabulary to delve into a bit of supporting knowledge.

# Chapter 2

# Background

We will start by looking at some basic Probability Theory, and then move on to Boolean Algebra. To conclude this section, we will introduce Block Ciphers and the interesting properties surrounding them. We will only define concepts that are needed as stepping stones to explaining Differential Cryptanalysis of Block Ciphers, and thus exclude some fundamental theorems to certain sections that are not needed.

## 2.1 Probability Theory

What does it mean for an event to have a probability of occurring? You probably have some intuitive understanding of what this means. For example, you will probably be aware that when you flip a regular coin, you have a 50% chance that it will land with heads facing up, and a 50% chance that it will land with tails facing up. It is also easy to see that if you roll an unweighted die, you have a 1 in 6 chance of landing on a particular number that was chosen beforehand.

What if I ask you what the probability of you getting an even number on a die is after you roll it? Most people would say there is a 50% chance, since half of the numbers are even and half of the numbers are odd. With this very intuitive understanding of probability, we will define probability more rigorously below.

Firstly, a **random experiment** is a procedure where the outcome cannot be determined before the procedure is completed (Underhill & Bradfield, 2009,

p. 57). In our examples above, tossing a coin or rolling a die can be considered random experiments. The set of all possible outcomes to a random experiment is called the **sample space** and a particular instance of conducting the random experiment is known as a **trial**. An **event** in this context is a subset of the sample space. So the coin landing with heads facing upwards, or the die landing on a 3, or even the die landing on an even number would all be examples of events occurring. However, an event that is a singleton in terms of being a subset of the sample space is called an **elementary event** . Thus, only getting heads on a coin toss, or getting a 3 on a die roll can be considered elementary events. Finally we will end off with a definition about how events relate to each other.

**Definition:** For $S$, some sample space, let $A, B \subset S$ be events. Then they are called **mutually exclusive** if $A \cap B = \emptyset$.

So what is probability then? Kolmogorov, often considered the father of probability, defined it as follows:

**Definition:** Suppose $S$ is a sample space for a random experiment. Then, for all events $A \subset S$, we define the **probability** of $A$, denoted $Pr(A)$, to be a real number with the following properties:

1. $0 \leq Pr(A) \leq 1$
2. $Pr(S) = 1$ and $Pr(\emptyset) = 0$, where $\emptyset$ is the empty set or **null event**.
3. For $A, B \subset S$, if $A \cap B = \emptyset$ then $Pr(A \cup B) = Pr(A) + Pr(B)$

Now that we have made precise the definition of probability, we can look into calculating probabilities of events occurring. Eventually we will use this to show how we can calculate the probability of a set of independent events. But first, a few theorems:

**Theorem 2.1.1.** *If $A_1, A_2, ..., A_n$ are pairwise mutually exclusive (in other words, for $i \neq j, A_i \cap A_j = \emptyset$), then*

$$Pr(A_1 \cup A_2 \cup ... \cup A_n) = \ Pr(A_1) + Pr(A_2) + ... + Pr(A_n) \tag{2.1}$$

*This can be written concisely as*

$$Pr\left(\bigcup_{i=1}^{n} A_i\right) = \sum_{i=1}^{n} Pr(A_i) \tag{2.2}$$

**Proof:** This proof can be obtained by the repeated use of Axiom 3.

We know that for any $A_i, A_j \in \{A_1, A_2, ..., A_n\}$, $A_i$ and $A_j$ are mutually exclusive. Thus, by use of Axiom 3, we have

$$Pr\left(\bigcup_{i=1}^{n} A_i\right) = Pr\left(\bigcup_{i=1}^{n-1} A_i\right) + Pr(A_n) \tag{2.3}$$

But it can also be noted that

$$Pr\left(\bigcup_{i=1}^{n-1} A_i\right) = Pr\left(\bigcup_{i=1}^{n-2} A_i\right) + Pr(A_{n-1}) \tag{2.4}$$

Substituting into 2.3, we get

$$Pr\left(\bigcup_{i=1}^{n} A_i\right) = Pr\left(\bigcup_{i=1}^{n-2} A_i\right) + Pr(A_{n-1}) + Pr(A_n) \tag{2.5}$$

Repeating this process, it is easy to see that

$$Pr\left(\bigcup_{i=1}^{n} A_i\right) = Pr(A_1) + ... + Pr(A_{n-1}) + Pr(A_n) \tag{2.6}$$

The result follows. □

Recalling that elementary events are mutually exclusive, if we know the probability of each elementary event, we can calculate the probability of any event by summing the probabilities of the elementary events it contains. This however can be tedious, so we look for a better way of calculating probabilities of a certain class of problems. They are the class of problems where by elementary events are equiprobable, that is having an equal probability of occurring. Thus, if there are $N$ elementary events in our sample space, each elementary event should have a probability of $\frac{1}{N}$ of occurring. Thus, to calculate the probability of an arbitrary event $A$ occurring, we could sum $\frac{1}{N}$ $n(A)$ times, where $n(A)$ denotes

the number of elementary events contained in $A$, since $A$ is just the union of elementary events contained in $A$. It is easy to see then that

$$Pr(A) = \frac{1}{N} \times n(A) \tag{2.7}$$

Now that we have a fast way of computing probabilites of arbitrary events for which we know the probability of elementary events, we can consider events that are independant of one another.

**Definition:** Let $A$ and $B$ be two events in a sample space $S$. Then the **conditional probability** of the event B given that the event A has occurred, denoted by $Pr(B|A)$, is

$$Pr(B|A) = \frac{Pr(A \cap B)}{Pr(A)} \tag{2.8}$$

What follows directly from the definition of conditional probability, is what it means for events to be independent.

**Definition:** Events are called **independent** if the outcome of one event does not affect the outcome of another. That is, for $A, B \subset S$

$$Pr(B|A) = Pr(B). \tag{2.9}$$

What however happens when we want to find the probability of two independent events both occurring? The next theorem tells us.

**Theorem 2.1.2.** *Let $A, B \subset S$ be two independent events. Then the probability of both $A$ and $B$ occurring, $Pr(A \cap B)$, is given by*

$$Pr(A \cap B) = Pr(A) \times Pr(B). \tag{2.10}$$

**Proof:** $A$ and $B$ are independent, thus

$$\begin{aligned} Pr(B|A) &= Pr(B) \\ \frac{Pr(A \cap B)}{Pr(A)} &= Pr(B) \\ Pr(A \cap B) &= Pr(A) \times Pr(B) \quad \square \end{aligned}$$

However, for Block Ciphers we are interested in the probability of multiple independent events occurring together. Thus, we finish off this section of probability theory by proving the following very useful theorem.

**Theorem 2.1.3.** *Let $A_1, ..., A_n$ be independent events, then*

$$Pr\left(\bigcap_{i=1}^{n} A_i\right) = Pr(A_1) \times ... \times Pr(A_n) \tag{2.11}$$

**Proof:** We prove this inductively by first examining the case of $n = 2$. We know

$$Pr\left(\bigcap_{i=1}^{2} A_i\right) = Pr(A_1) \times Pr(A_2) \tag{2.12}$$

from Theorem 2.1.2 Now we examine the case where $n = m$ for some arbitrary $m \in \mathbb{N}$. Since $A_1, ..., A_m$ are independent events, it is easy to see that $\bigcap_{i=1}^{m-1} A_i$ and $A_m$ are independent events. Thus we can see that

$$Pr\left(\bigcap_{i=1}^{m} A_i\right) = Pr\left(\bigcap_{i=1}^{m-1} A_i\right) \times Pr(A_m) \tag{2.13}$$

holds. However,

$$Pr\left(\bigcap_{i=1}^{m} A_i\right) \times Pr(A_{m+1}) = Pr\left(\bigcap_{i=1}^{m-1} A_i\right) \times Pr(A_m) \times Pr(A_{m+1}) \tag{2.14}$$

yields

$$Pr\left(\bigcap_{i=1}^{m+1} A_i\right) = Pr\left(\bigcap_{i=1}^{m} A_i\right) \times Pr(A_{m+1}). \tag{2.15}$$

Expanding out, we get

$$Pr\left(\bigcap_{i=1}^{m} A_i\right) = (...(Pr(A_1) \times Pr(A_2)) \times Pr(A_3)) \times ...Pr(A_m). \tag{2.16}$$

and the result follows by induction for every $n \in \mathbb{N}$.

## 2.2 Boolean Algebra

In order to understand how many Block Ciphers work, we will have to take a look at Boolean Algebra, that is, the logical calculus of truth values. In particular we will look at the operation known as the 'exclusive or', commonly known as XOR and represented by the symbol '$\oplus$'.

### 2.2.1 XOR

**Definition:** The XOR of two boolean values is true if either one of the values is true, and is false if both are true, or both are false.

In simpler terms, we can view it as a function that takes two inputs, and returns true if either the one value or the other is true, but not both. As we are dealing with True and False values, we will use the more compact notation of representing $True$ as 1, and $False$ as 0.

Thus, the truth table for the XOR operation is given as follows:

| $A$ | $B$ | $A \oplus B$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

This can be compactly noted in the operation table below:

| | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

What the above table is saying, is that $0 \oplus 0 = 1 \oplus 1 = 0$. Likewise, $0 \oplus 1 = 1 \oplus 0 = 1$.

**Remark:** It is easy to see that this operation is equivalent to addition in $\mathbb{F}_2$, that is, the finite field of order 2.

### 2.2.2 Boolean Vectors

We can examine collections of truth values in a particular order as an $n$-vector of boolean values.

This results in us being able to define XOR on these vectors:

**Definition:** Let $A = (a_1, a_2, ..., a_n), B = (b_1, b_2, ..., b_n) \subset \{0, 1\}^n$ be two Boolean $n$-vectors. Then,

$$A \oplus' B = (a_1 \oplus b_1, a_2 \oplus b_2, ..., a_n \oplus b_n) \tag{2.17}$$

where $\oplus'$ represents the component-wise XOR of boolean vectors.

**Remark:** Since it will never be ambiguous as to whether we are XORing boolean values or boolean vectors, we will drop the use of $\oplus'$ and just use $\oplus$ to denote XOR of both values and vectors.

### 2.2.3 Bits

We have already seen how it is convenient to represent truth values as 1s and 0s in the previous subsection. We will now look at the primary way of compactly representing Boolean vectors - namely that of bit strings.

Firstly, let's take a look at some Computer Science terminology:

- A **bit** or binary digit is variable which holds either a 0 or a 1.
- A **bit string** is a sequence of 1 or more bits.

We can see that the definition of a bit fits well with our representation of boolean values in the previous section. In fact, in the space of Boolean Algebra, the term bit is actually used to denote truth values given by this numerical representation.

Thus, if we consider a Boolean $n$-vector $(a_1, a_2, ..., a_n)$, we can represent this as a bit string of length $n$, of the form $a_1a_2...a_n$. Furthermore, our definition of XOR applies to bits as well and a bit string can be bit-wise XOR'd with a string of the same length, by taking the component-wise XOR of the corresponding Boolean vector. This results in the following definition of a bit-wise XOR:

**Definition:** Let $A = a_1a_2...a_n$ and $B = b_1b_2...b_n$ be two bit strings of length $n$. Then $A \oplus B$ is defined to be the bit string representation of $A' \oplus B'$ where $A' = (a_1, a_2, ..., a_n), B' = (b_1, b_2, ..., b_n) \subset \{0, 1\}^n$, the Boolean vector representation of $A$ and $B$.

**Example:** Suppose we wanted to XOR 101011 with 011010. We could convert the bit strings into their boolean tuple representation and component-wise XOR them.

Thus,

$$\begin{aligned} 101011 \oplus 011010 &= (1,0,1,0,1,1) \oplus (0,1,1,0,1,0) \\ &= (1 \oplus 0, 0 \oplus 1, 1 \oplus 1, 0 \oplus 0, 1 \oplus 1, 1 \oplus 0) \\ &= (1,1,0,0,0,1) \\ &= 110001 \end{aligned}$$

You might notice that this operation can be described as taking the first bit string, and flipping a bit wherever you see a 1 as the corresponding bit in the second bit string.

**Remark:** Often, for the sake of brevity, we will use the decimal or hexadecimal representation of a number rather than it's binary representation. The reader is assumed to have knowledge of number systems and converting between variously based number systems.

## 2.3 Block Ciphers

We will be considering an attack on Block Ciphers later, and thus it makes sense to introduce Block Ciphers as part of the background. In short, Block Ciphers can be defined as algorithms that operate on a fixed number of bits, using some sort of symmetric key. (Menezes et al., 1996)

Alright, let's take a step back and try understand what that means. Given some plaintext, a block cipher will convert fixed-size blocks of that plaintext into the same size blocks of ciphertext during the encryption stage. The way in which it does this depends on the type of block cipher in discussion, but usually this involves XORing in a secret key in some way. Furthermore, since this key is symmetric, it can be used with the same block cipher to reverse the process, and convert ciphertext that has been generated by this particular block cipher, back into plaintext.

In particular, a block cipher is a combination of two paired algorithms, E for encryption, and D for decryption. Both algorithms accept two inputs, an input box of size $n$ bits, and a key of size $k$ bits and both yield an $n$-bit output block.

How could we make this definition more precise though? With our intutive understanding of a Block Cipher, we can define it mathematically as follows:

**Definition:** If $K$ is an input key of bit length $k$, and $P$ is a string of input bits of length $n$, let us consider a function

$$E_K(P) := E(K, P) : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n \tag{2.18}$$

For each $K$ and output of $E_K(P) = C$, the function $E_K(P)$ is required to be an invertible mapping on $\{0,1\}^n$, with the inverse defined as:

$$E_K^{-1}(C) := D(K, C) : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n \tag{2.19}$$

such that

$$\forall K \ D_K(E_K(P)) = P \tag{2.20}$$

holds.

Then the pair $(E_K, D_K = E_K^{-1})$ constitutes a block cipher.

**Remark:** In the above:

- $k$ is known as the **key size**
- $n$ is known as the **block size**
- $C$ is known as the **ciphertext**
- $P$ is known as the **plaintext**

There are different types of block ciphers, but for the purposes of this paper, we will focus on a specific type of iterated block cipher, known as a Substitution-Permutation Network (SPN). There are other types of block ciphers, such as Feistel ciphers, but these are beyond the scope of our discussion. What makes SPNs different from other block cipher implementations, is the way the symmetric key is mixed in with the plaintext to form ciphertext.

Iterated block ciphers are block ciphers in which the secret key used to generate the various subkeys or "round keys" that are mixed in over multiple

rounds. At each round, a "round function" takes the previous round's output as input, with the first round using the plaintext as input. Thus, these round functions are chained together to form our iterated block cipher. In the case of an SPN, the round function is made up of a Key-mixing stage, a Substitution box (S-box) and a Permuation box (P-box), in that order.

Looking at figure 2.1, we can see this in action. In the next subsections we will discuss what S-boxes and P-boxes are.

### 2.3.1 Vectorial Boolean Functions (S-boxes)

A large part of understanding Block Ciphers, and how to attack them, will be tied up in understanding S-boxes (which are types of Vectorial Boolean Functions). This section will deal with introducing and explaining them, along with a few examples.

**Definition:** Any bijection $S : \{0,1\}^n \to \{0,1\}^m$ which maps boolean $n$-vectors to boolean $m$-vectors, is called a **Substitution-box** (S-box). Where $n = m$ we call $n$ the block size, and we refer to an S-box with block size $n$, as an $n$-bit S-box. In the case that $n \neq m$, we will refer to such an S-box as an $(n \times m)$-bit S-box.

**Remark:** For the sake of convenience, we will continue to discuss Block-Ciphers in terms of $n$-bit S-boxes, unless otherwise stated.

What immediately follows from the fact that S is a bijection, is that the inverse $S^{-1}$ exists, and that we can reverse the S-box.

**Example:** A very simple, but not very useful, S-box would be one operating on $n$-vectors, which just maps every element to itself (the so called identity S-box).

$$S_{id} : \{0,1\}^n \to \{0,1\}^n \tag{2.21}$$

where

$$S_{id}(x) = x \tag{2.22}$$

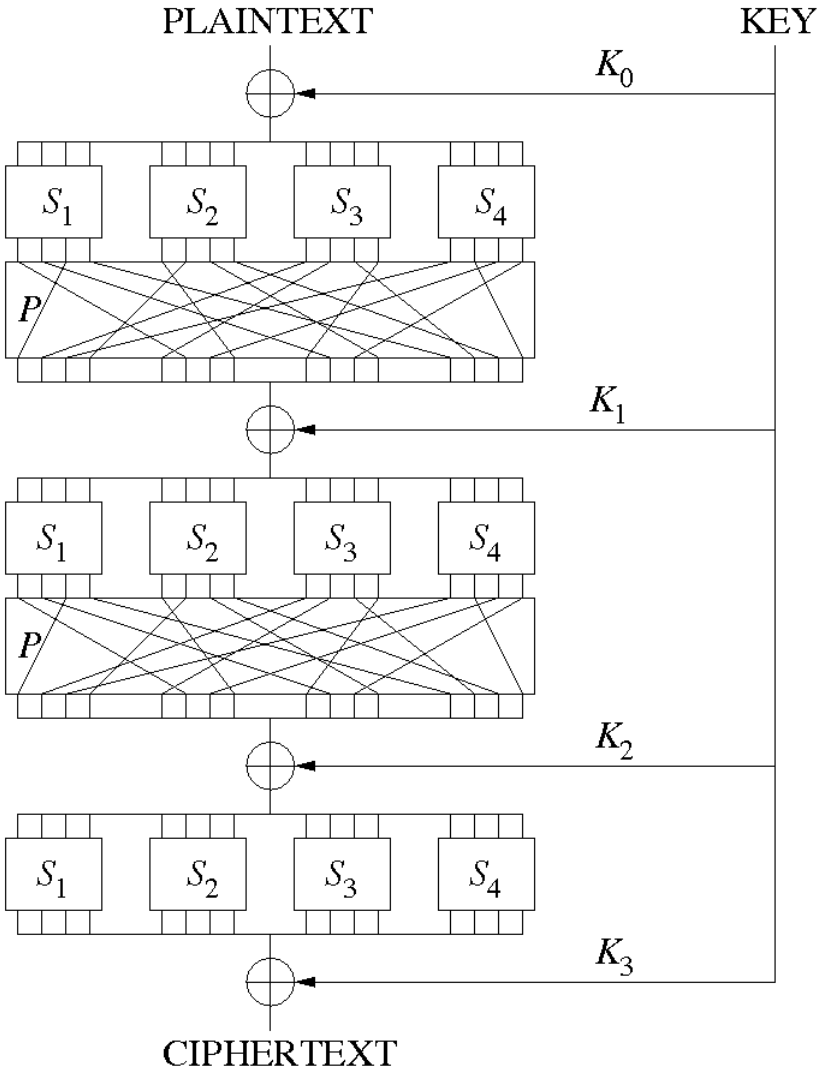


Figure 2.1: Substitution-Permutation Network (Gabore, 2009)

**Example:** Let's consider a simple 3 bit S-box. Since there are only 3 bits, we have $2^3 = 8$ possible inputs. Since S-boxes are bijections, we can only have 8 possible outputs. Thus, we can think of an S-box as a type of look-up table.

| $x$ | $y = S(x)$ |
|---|---|
| 000 | 010 |
| 001 | 110 |
| 010 | 000 |
| 011 | 100 |
| 100 | 011 |
| 101 | 001 |
| 110 | 111 |
| 111 | 101 |

We can however reverse the S-box, taking $x$ to be the subject of our formula. So instead of $y = S(x)$, we get $x = S^{-1}(y)$. Looking at the previous table like this, we get:

| $y$ | $x = S^{-1}(y)$ |
|---|---|
| 000 | 010 |
| 001 | 101 |
| 010 | 000 |
| 011 | 100 |
| 100 | 011 |
| 101 | 111 |
| 110 | 000 |
| 111 | 110 |

### 2.3.2 P-boxes

From the previous section, we can see here is that S-boxes serve to obscure the relationship between plaintext and ciphertext, thereby providing good **confusion** for the Block Cipher as defined by Shannon (1949). Another property that we

want for a good Block Cipher would be that small changes in our plaintext will result in completely different ciphertext. More formally, we want that redundancy, or non-message bits, in the statistics of our plaintext be dissipated in the statitics of our ciphertext, known as Shannon's property of **diffusion**. To achieve that, we use something called a Permutation-box.

**Definition:** A **Permutation-box** or P-box is a method of bit-shuffling used to permute or transpose bits across bit strings.

Since a P-box permutes bits over bit strings, we can put a P-box immediately after an S-box to send output bits from one S-box to a different input location for the next round. We refer to such an S-box followed immediately by a P-box as an SP-box.

# Chapter 3

# Differential Attacks

So far we have taken a look at a number of sections, which for the most part look fairly disjoint. In this section, we will tie these concepts together and discuss the process whereby a differential attack can be mounted against a block cipher. Perhaps the easiest way to do this would be by means of working through the process on an example in various subsections.

So before we begin, we would want to clarify what assumptions need to be made in order to mount a successful attack against the type of Block Cipher discussed in the previous chapter.

We can reduce them to the following list:

- We can choose some plaintext.
- We know or can generate the corresponding ciphertext.
- We know the SP-boxes.
- We do not know the key.

Accepting those assumptions, we are ready to note down our algorithm for attacking Block Ciphers, and specifically a SPN, with Differential Cryptanalysis. So let's explain how we would attack an SPN then...

Firstly, for each round of our Iterated Block Cipher, we will want to search for any pairs of inputs $(X, X')$ with XOR difference $\Delta X$, which results in the cor-

responding output pair $(Y, Y')$ having XOR difference $\Delta Y$ with high probability. To be clear however in the above, $X \oplus X' = \Delta X$ and $Y \oplus Y' = \Delta Y$. The pair $(\Delta X, \Delta Y)$ is known as a **differential**. We can do this, since although SP-boxes affect inputs and outputs, they do not affect differentials. This enables us to to build up a differential characteristic for the Block Cipher.

Thinking back to the structure of our SPN, if there is an SP-box after the last round, we can simply invert it and get what the input would be to said SP-box, so either way we are left facing the last round key. Then, since we know high-probability inputs for the last round from our differential characteristic, we can XOR the inputs with our know ciphertext to extract the key.

In summary, once we have chosen our plaintext/ciphertext pairs, our basic methodology for breaking the last round is as follows:

1. For each round up to $r - 1$, compute the differentials on the SP-box.
2. Find differential characteristics with high probabilities.
3. Extract the last round key.

We will use this strategy iteratively to break each round key, so let us discuss each of these points in greater detail.

## 3.1 Computing Differentials for Each Round

So, for each round, how do we analyze an SP-box to retrieve its differentials? Well since the SP-boxes are publicly known, we can iterate through all possible inputs $X$ to a particular SP-box, and for each input, iterate through all possible input differences $\Delta X$ to get input pairs $(X, X' = X \oplus \Delta X$. Then, by sending the inputs, $X$ and $X'$ through the SP-box, we can calculate each possible output difference $\Delta Y$ given by $\Delta Y = S(X) \oplus S(X')$. Once we have all of these output differences, we can tally the output differences that occur often for a given input difference, and record them in a differential table.

This might be best explained using an example, so let us consider the SP-box represented by the following table (in hexadecimal):

| $X$ | $S(X)$ |
|---|---|
| 0 | 3 |
| 1 | 5 |
| 2 | 2 |
| 3 | 1 |
| 4 | 6 |
| 5 | 7 |
| 6 | 4 |
| 7 | 0 |
| 8 | A |
| 9 | C |
| A | B |
| B | 8 |
| C | 9 |
| D | F |
| E | D |
| F | E |

You might notice that in this SP-box, function values do not seem to be particularly well distributed. In fact, any value less than 8 is mapped to corresponding value less than 8, and consequently values greater than or equal to 8 are mapped to values greater than 8. In real life, this would correspond to a bad permutation function, but we purposefully use this SP-box so that we get clear outliers in terms of high-probabities differentials when we compute the differential table.

Thus, if we step through possible inputs, $\{0, ..., 15\}$ to our SP-box, and for each input, XOR it with all the possible input differences $\{0, ..., 15\}$, we get the following pairs $(X, X \oplus \Delta X)$ represented by the row and column headings of the table below:

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

Each value in the above table is the $\Delta Y$ for a given $X$, the row label, and $\Delta X$, the column label. To work out the $\Delta Y$ by hand, use the following formula:

$$\Delta Y = S(X) \oplus S(X \oplus \Delta X) \tag{3.1}$$

By counting the number of times a particular output difference $\Delta Y$ occurs for a particular input difference $\Delta X$, we can generate the following table, where the column headings are output differences and the row headings are input differences:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 2 | 0 | 6 | 2 | 0 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 6 | 2 | 0 | 6 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 6 | 2 | 0 | 0 | 2 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 2 | 2 | 4 | 0 | 2 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 2 | 2 | 2 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 4 | 0 | 2 | 2 | 2 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 6 | 0 | 2 | 4 | 2 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 10 | 0 | 0 | 0 | 0 | 2 | 2 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 6 | 0 | 0 | 2 | 2 | 4 |
| A | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 2 | 0 | 2 | 0 | 6 | 0 | 2 |
| B | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 2 | 6 | 0 | 0 | 4 | 0 |
| C | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 4 | 2 | 2 | 0 | 0 | 6 |
| D | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 10 | 2 | 0 | 0 |
| E | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 2 | 2 | 6 | 2 |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 0 | 2 | 0 | 2 | 4 | 2 | 0 |

Taking a look at the table above, we see that certain input differences map to certain output differences significantly more often than others. For example, the input difference 5 goes to the output difference 5, 10 out of the 16 times, giving us a probability of $\frac{10}{16}$. For a perfect SP-box, we want each input difference go to each output difference $\frac{1}{2^n} = \frac{1}{16}$. Unfortunately, we know that $\Delta Y = S(X) \oplus S(X \oplus \Delta X) = S(X \oplus \Delta X) \oplus S(X)$ so input differences map to output differences in pairs, and thus the smallest number we could get is 2.

Up until now, we have all but ignored the effect of the key, but there is a good reason why: the key does not affect the differentials of a round.

To see this, notice that for inputs $X$ and $X'$ and round subkey $K$, we have:

$$\begin{aligned}(X \oplus K) \oplus (X' \oplus K) &= X \oplus K \oplus X' \oplus K \\ &= X \oplus X' \oplus K \oplus K \\ &= X \oplus X' \\ &= \Delta X\end{aligned}$$

## 3.2 Constructing a Differential Characteristic

In the previous section, we saw how to calculate the differentials for a particular round. However, we can conduct this process for each round, and chain together high-probability differentials between rounds, giving us a **differential characteristic** of the Block Cipher. We do this by using one round's high probability output differential as the next round's input differential.

**Example:** Continuing the example from the previous section, we saw that the following input differences went to the following output differences with high probability:

- $5 \to 5$ with probability $\frac{10}{16}$
- $8 \to 9$ with probability $\frac{10}{16}$
- $D \to C$ with probability $\frac{10}{16}$

Now suppose this was our first SP-box, with our SPN consisting of 4 rounds, we would have the same information for each of these rounds. Perhaps the second SP-box had a differential $9 \to 2$ with probability $\frac{8}{16}$ and the third SP-box had a differential $2 \to 6$ with probability $\frac{6}{16}$. Then, since these probabilities are assumed to be independent, we can multiply them together to get a differential characteristic $8 \to 2$ with probability $\frac{10}{16} \times \frac{8}{16} \times \frac{10}{16} = \frac{600}{4096} = \frac{75}{512}$ for the first 3 rounds. Thus, we are assured that plaintext pairs with a difference of 8 will result in input pairs to round 4 with a difference of 2 with high probability.

**Remark:** If the probabilities are not independent, then we can't multiply them together and the differential attack would not work.

We then use this differential charateristic in the next stage of our attack.

## 3.3 Extracting Each Round Key

How would we extract a key from a round? Well if we knew the input $X$ to a round, the SP-box and the output $Y$, we could send the output back through the SP-box, to get $S^{-1}(Y) = X \oplus K$ and XOR it with $X$ to get the key K since

$$S(X \oplus K) = Y. \tag{3.2}$$

For the last round, we know what the outputs are since we can generate ciphertext from plaintext. Unfortunately, we do not know the specific input to our last round before the key gets mixed in. We do however know what possible plaintext input differences will lead to what possible input differences to round $r$ from our differential characteristic.

All that is then required is to choose plaintext pairs $(P, P')$ satisfying our high-probability differential characteristic, and generate corresponding inputs to round $r$ from the input difference to round $r$. Now for each input pair $(X, X')$, we can XOR the inputs $X$ and $X'$ with the inputs to our SP-box $S^{-1}(C)$ and $S^{-1}(C')$, such that we get:

$$X \oplus S^{-1}(C) = K \tag{3.3}$$

$$X' \oplus S^{-1}(C') = K' \tag{3.4}$$

Thus, we get possible values for the round key $K$ and $K'$. If we have chosen a high probability differential on which we base our input pairs, there should be a $K$ that clearly occurs more often than others. This is then our candidate key for that round. We then check this key against arbitrary plaintext inputs to ensure that we get the expected ciphertext outputs, and once the key is verified, we accept as being the correct key for the round.

Now we repeat the method for our block cipher consisting of $r - 1$ rounds, and extract each round key. Once we have all the round keys, we can easily generate the key and we will then have broken the SPN.

# Chapter 4

# Implementation

In the previous chapter, statistical analysis was conducted on SP-boxes using the following Python3 program:

```
#define an SP-box in terms list with index corresponding to input bit
first = [3,5,2,1,6,7,4,0]
second = [10,12,11,8,9,15,13,14]
sbox = first + second
size = len(sbox)
dt = []

for i in range(size):
    dt.append([0]*size)
    #print('%X & %X \\\\\\hline' % (i, sbox[i]))

def s(x):
    return sbox[x]

def outputs():
    for i in range(size):
        print(get(i))
```

```
def diffs():
    print('   ', end='')
    for i in range(size):
        print(' %X ' % i, end=' ')
    print()
    for x in range(size):
        print('%X' % x,end='| ')
        for delx in range(size):
            dely = s(x) ^ s(x^delx)
            print('%X:%X' %(delx, dely) ,end=' ')
            dt[delx][dely] += 1
        print('')
    print()

def diff_table():
    print('   ', end='')
    for i in range(size):
        print(' %X' % i, end=' ')
    print()
    for i in range(size):
        print('%X' % i,end='| ')
        for j in range(size):
            print('%2d' % dt[i][j] ,end=' ')
        print('')

diffs()
diff_table()
```

# Chapter 5

# Conclusion

Thus, we can see that the basic structure of Block Cipher, and in particular a Substitution-Permutation Network, can be broken with relative ease using Differential Cryptanalysis. Fortunately, there have been numerous improvements made to this simple structure, resulting in ciphers, such as AES, that are impervious to Differential Cryptanalysis. However, the mechanics behind creating a strong Block Cipher are well beyond the scope of our discussion.

# References


Curtin, M., & Dolske, J. (1998, May). *A Brute-Force Search of DES Keyspace.* Available from `http://www.interhack.net/pubs/des-key-crack/`

Gabore, P. (2009, April). *SubstitutionPermutationNetwork2.png.* (Creative Commons Attribution)

King, J. (2012, August). *Differential cryptanalysis tutorial.* Available from `http://www.theamazingking.com/crypto-diff.php`

Menezes, A., van Oorschot, P., & Vanstone, S. (1996). *Handbook of applied cryptography.* CRC Press.

Shannon, C. E. (1949). Communication theory of secrecy systems. *Bell System Technical Journal*, *28*(4), 656–715. Available from `http://netlab.cs.ucla.edu/wiki/files/shannon1949.pdf`

Underhill, L., & Bradfield, D. (2009). *Introstat.* University of Cape Town.