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 desparate 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. 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 doning 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 cipher text 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 posessing 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 before hand.

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. 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 (Underhill & Bradfield, 2009, p. 57). 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** (Underhill & Bradfield, 2009, p. 58). 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 of 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 $A \cap B = \emptyset$ (Underhill & Bradfield, 2009, p. 58).

So what is probability then? Kolmogorov, often considered the Father of probability, defined it as follows (Underhill & Bradfield, 2009, p. 60):

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

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

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 2.5, 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. □

TODO:

1. Calculating probabilities from first principles
2. Conditional probabilities
3. independent events

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

$$Pr(B|A) = \frac{Pr(A \cap B)}{Pr(A)} \tag{2.7}$$

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

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

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*

$$Pr(A \cap B) = Pr(A) \times Pr(B) \tag{2.9}$$

Proof: A and B are independent, thus

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

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

$$Pr\left(\bigcap_{i=1}^{n} A_i\right) = Pr(A_1) \times ... \times Pr(A_n) \tag{2.10}$$

Proof: TODO

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

Can I use the term boolean vector even though I haven't defined such a vector over a vector space? Other papers seem to do so? At the moment, I'm just using n-tuple instead of n-vector

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

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

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

$$A \oplus' B = (a_1 \oplus b_1, a_2 \oplus b_2, ..., a_n \oplus b_n) \tag{2.11}$$

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

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

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 a 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 amount 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 2 paired algorithms, E for encryption, and D for decryption. Both algorithms accept 2 inputs, an input box of size n bits, and a key of size k bits and both yield a 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: (Menezes et al., 1996) For any K, an input key of bit length k, and P is a string of input bits of length n, let us consider a function

$$E_K(P) := E(K, P) : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n \tag{2.12}$$

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:

$$E_K^{-1}(C) := D(K, C) : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n \tag{2.13}$$

such that

$$\forall K \ D_K(E_K(P)) = P \tag{2.14}$$

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 is split up into various subkeys, and 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 an Substitution box (S-box), a Permuation box (P-box) and a Key-mixing stage, in that order.

Looking at the diagram below, 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 $S : \{0,1\}^n \to \{0,1\}^n$ which maps boolean n-tuples to boolean n-tuples, is called an **Substituion-box** (S-box). In this instance, n is called the block size, and we refer to an S-box with block size n, as an n-bit S-box.

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-tuples, which just maps every element to itself.

$$S_{id} : \{0,1\}^n \to \{0,1\}^n \tag{2.15}$$

where

$$S_{id}(x) = x \tag{2.16}$$

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

S-boxes provide good confusion. What about diffusion? P-boxes. Refer to S-box followed immediately by P-box as SP-box.

2.4 Differential Properties of SP-boxes

Next will be discussed the various differential properties of S-boxes which allow cryptanalysts to mount an attack against a block cipher. These include the

property that XORs do not affect differentials, as well as ways to find relationships between input differentials and output differentials of an S-box.

2.4.1 Probabilities of Differential Trails

The previous properties discussed can be combined to give us insight into the probabilities of differentials trails, or in plain English: Given an input differential, how likely or probable is it that a certain output differential occurs. This section will discuss the theory behind probabilities of differentials.

All of this however, is based on the assumption that these probabilities are linearly independent. If they are not, then we can't be sure of what the final probability is.

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 in various subsections, and then putting it all together in the form of a theoretical discussion of why it works.

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:

1. We can choose some plaintext.
2. We know or can generate the corresponding ciphertext.
3. We know the S-boxes
4. 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, we will choose some plaintext and generate the corresponding cipher text by assumption 2. We now have an fairly large number of plaintext inputs and their corresponding ciphertext outputs. What we are wanting to do however, is search for any pairs of plaintext (P, P') with XOR difference ΔP, which results in the corresponding ciphertext pair (C, C') having XOR difference ΔC with high probability. In this way, we will be exploting the fact that S-boxes do not affect these so called differentials. To be clear however in the above, $E_K(P) = C$ and $E_K(P') = C'$. Futhermore, $P \oplus P' = \Delta P$ and $C \oplus C' = \Delta C$ and the pair $(\Delta P, \Delta C)$ is known as a differential.

If we perform this method over all the inputs that we have, we can tabulate the trails that occur with high probability. In a perfect cipher, the probablity of any particular output difference occuring for any particular input difference is $\frac{1}{2^n}$ (Heys, 2002, p. 19) where n is the block size, so we are looking for pairs with probabilities much higher than this.

Thinking back to the structure of our SPN, if there is an S-box after the last round, we can simply invert it and get what the input would be to said S-box, so either way we are left facing the last round key.

In summary, our basic methodology is as follows:

1. Choose some plaintext/ciphertext pairs.
2. Compute the differentials and find differential trails with high probabilities
3. Extract the last round key in one of two ways
4. Rinse and repeat

Let us discuss each of these in greater details then.

3.1 Choosing Plaintext/Ciphertext Pairs

3.2 Computing Differentials

So how do we analyze an S-box to retrieve its differentials? Well since the S-boxes are publicly known, we can iterate through all possible inputs X to a particular

S-box, and for each input, iterate through all possible input differences ΔX to get input pairs $(X, X' = X \oplus \Delta X$. Then, by sending the inputs, X and X' through the S-box, we can calculate each possible output difference ΔY 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 S-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 this S-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. This is 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 S-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:

ΔX values

X	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 ΔY for a given X, the row label, and ΔX, the column label. To work out the ΔY 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 ΔY occurs for a particular input difference ΔX, 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 S-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.

But we can do this process for each round, and chain together high-probability differentials between rounds, giving us a **differential trail**.

3.3 Statistical Analysis on Differentials

3.4 Breaking Each Round Key

3.5 Combining it all together

3.6 The Theory Behind It or Why It Works

Chapter 4

Conclusion

In the conclusion, I will wrap up what has been discussed in my paper, and mention what can be improved upon in the future.

References

Heys, H. M. (2002, July). A tutorial on linear and differential cryptanalysis. *Cryptologia*, *26*, 189–221. Available from `http://portal.acm.org/citation.cfm?id=763194.763197`

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

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